WTAGRAPH: Web Tracking and Advertising Detection using Graph Neural Networks

Zhiju Yang
Colorado School of Mines

Weiping Pei
Colorado School of Mines

Monchu Chen
Appen

Chuan Yue
Colorado School of Mines

***Abstract*—Web tracking and advertising (WTA) nowadays are ubiquitously performed on the web, continuously compromising users' privacy. Existing defense solutions, such as widely deployed blocking tools based on filter lists and alternative machine learning based solutions proposed in prior research, have limitations in terms of accuracy and effectiveness. In this work, we propose WTAGRAPH, a web tracking and advertising detection framework based on Graph Neural Networks (GNNs). We first construct an attributed homogenous multi-graph (AHMG) that represents HTTP network traffic, and formulate web tracking and advertising detection as a task of GNN-based edge representation learning and classification in AHMG. We then design four components in WTAGRAPH so that it can (1) collect HTTP network traffic, DOM, and JavaScript data, (2) construct AHMG and extract corresponding edge and node features, (3) build a GNN model for edge representation learning and WTA detection in the transductive learning setting, and (4) use a pre-trained GNN model for WTA detection in the inductive learning setting. We evaluate WTAGRAPH on a dataset collected from Alexa Top 10K websites, and show that WTAGRAPH can effectively detect WTA requests in both transductive and inductive learning settings. Manual verification results indicate that WTAGRAPH can detect new WTA requests that are missed by filter lists and recognize non-WTA requests that are mistakenly labeled by filter lists. Our ablation analysis, evasion evaluation, and real-time evaluation show that WTAGRAPH can have a competitive performance with flexible deployment options in practice.**

***Index Terms*—web tracking and advertising, privacy, graph neural networks**

I. INTRODUCTION

Web tracking and advertising (WTA) are ubiquitously performed by trackers to collect web users' browsing activities for various purposes such as personalized advertisements and behavioral analytics [1]–[5], during which user privacy has often been compromised. To protect users, a basic approach is to detect and act when WTA are about to take place. Currently, a widely deployed solution is to use blocking tools such as AdBlock Plus [6] and uBlock Origin [7]. These tools can detect and block WTA requests based on filter lists (such as EasyList [8] and EasyPrivacy [9]) that define a set of rules for determining whether an HTTP (or HTTPS) request is WTA-related (i.e., related to either tracking or advertising, or both). However, the maintenance of these manually-curated filter lists often requires a significant amount of human effort, and prior research has also shown that filter lists have shortcomings such as incurring false-positive and false-negative errors [10], [11].

Therefore, researchers have proposed machine learning based solutions to better detect WTA requests especially with fewer false negatives. By extracting features from HTTP traffic and request URLs [12], [13], various machine learning classifiers have been developed to detect WTA-related requests. Similarly, by using syntactic and semantic features of JavaScript code [14]–[16], researchers have shown the effectiveness of different machine learning classifiers on detecting WTA-related JavaScript requests. Recently, Iqbal et al. [17] presented AdGraph, a state-of-the-art approach for WTA detection. However, though each of these prior efforts made important contributions, opportunities exist for researchers to further improve the performance of the machine learning based approach for WTA detection.

In this paper, we propose WTAGRAPH, a web tracking and advertising detection framework based on Graph Neural Networks (GNNs). There has been a surge of success in applying GNNs for addressing problems with graph-structured data in recent years [18]–[21]. Typical GNNs adopt the neighborhood aggregation strategy (i.e., message passing), where the representation of each node is iteratively learned by combining its feature vector and feature vectors aggregated from its neighbors. In practice, many GNN variants have achieved state-of-the-art performance because GNNs can utilize both explicit features of nodes and implicit features learned from the graph for learning better node representations. By nature, HTTP network traffic, where WTA requests are included, can be structured as a graph, in which an edge represents a specific HTTP request and a node represents either the source domain that issues an HTTP request or the destination domain that an HTTP request is sent to. Motivated by the fact that HTTP network traffic can be represented by a graph and GNNs are successful in graph learning tasks, we construct an attributed homogenous multi-graph (AHMG) to represent HTTP network traffic, and formulate WTA detection as a task of GNN-based edge representation learning and classification in AHMG. To effectively perform this task, we propose WTAGRAPH and address a few technical challenges including AHMG construction, attribute assignment, edge representation formulation, and message passing.

We designed and implemented four components in WTAGRAPH so that it can (1) collect HTTP network traffic, DOM, and JavaScript data, (2) construct AHMG and extract corresponding edge and node features, (3) build a GNN model for edge representation learning and WTA detection in the transductive learning setting, and (4) use a pre-trained GNN model for WTA detection in the inductive learning setting.

We evaluated WTAGRAPH on a dataset collected from Alexa Top 10K websites, and showed that WTAGRAPH can effectively detect WTA requests in both transductive and inductive learning settings. For example, it detects WTA requests with 97.90% accuracy, 98.38% precision, 96.25% recall, and 97.30% F_1 score in the transductive learning setting; it detects WTA requests with 97.82% accuracy, 98.00% precision, 96.38% recall, and 97.18% F_1 score in the inductive learning setting. Manual verification results indicate that WTAGRAPH can detect new WTA requests that are missed by filter lists and recognize non-WTA requests that are mistakenly labeled by filter lists. Our ablation analysis, evasion evaluation, and real-time evaluation show that WTAGRAPH can have a competitive performance with flexible deployment options in practice.

Overall, our paper makes the following major contributions: (1) we proposed a novel GNN-based approach for WTA detection, (2) we developed a GNN that directly learns edge representation in AHMG, (3) we designed and implemented an edge representation aggregation strategy in our GNN, (4) we implemented WTAGRAPH that integrates data collection, graph construction, GNN training, and WTA detection into a single framework, and (5) we performed large scale evaluations and showed that WTAGRAPH is effective and useful in both transductive and inductive learning settings.

The rest of this paper is structured as follows. Section II reviews the related work on WTA detection and Graph Neural Networks. Section III gives the formal definition of our AHMG and formulates our task of WTA detection. Section IV describes the design and implementation of WTAGRAPH. Section V describes our data collection process and dataset. Section VI and Section VII present the evaluation results of WTAGRAPH in transductive and inductive settings, respectively. Section VIII discusses the use of WTAGRAPH, limitations, and future work. Section IX concludes this work.

II. Related Work

Our work is closely related to WTA detection and GNNs.

WTA Detection. Web tracking and advertising compromise users' privacy by associating their identities with their browsing activities on different websites. Based on manually-curated filter lists that define a set of rules, blocking tools have been widely deployed to detect WTA requests. However, prior research has shown the shortcomings of this solution regarding maintenance effort, false-positive errors, and false-negative errors [10], [11].

Therefore, researchers have proposed machine learning based solutions to better detect WTA requests. Gugelmann et al. [12] introduced a machine learning based method for classifying WTA requests. Based on a set of HTTP traffic features, the authors trained a classifier that can identify WTA requests with the precision and recall at around 84%. Similarly, Shuba et al. [13] proposed the NoMoAds framework for ad-blocking on the mobile environment. It extracts features from HTTP traffic and trains a decision tree model to detect WTA requests. Bhagavatula et al. [22] developed a k-nearest neighbors classifier that can detect WTA requests using 91 URL features. By using syntactic and semantic features of JavaScript code, Ikram et al. [14] proposed a one-class classifier that can detect WTA-related JavaScript programs. Wu et al. [15] and Kaizer et al. [16] introduced similar classifiers that detect WTA-related JavaScript requests using code behaviors (e.g., API and cookie access). Our approach differs from these solutions significantly because we build a GNN model to better identify WTA requests by utilizing both explicit and implicit features learned from AHMG.

Recently, Iqbal et al. [17] proposed a graph-based framework, named AdGraph, for WTA detection. By instrumenting the Chromium browser, AdGraph first records DOM changes caused by responsible parties (such as some JavaScript code and HTTP request) during a webpage visit. Using the recorded DOM changes, it then builds a graph that captures the context of HTTP requests (e.g., DOM manipulation and JavaScript behavior). Lastly, it extracts 64 features from the context graph for an HTTP request and trains a random forest model to classify WTA requests with 95.33% accuracy, 89.1% precision, and 86.6% recall [17]. Note that building upon AdGraph, the PageGraph [23] component of the Brave browser is under development. By more comprehensively attributing DOM events, PageGraph can perform various tasks such as filter list generation and web compatibility analysis [23].

Our approach also differs from this work significantly in two aspects. First, we construct attributed homogenous multi-graph (AHMG) to represent HTTP network traffic of a set of requests, while AdGraph builds a graph to represent the context of HTTP requests for a single webpage visit. Second, we develop a GNN model to learn edge representations and detect WTA requests, while AdGraph trains a traditional random forest model based on explicitly extracted features to detect WTA requests. We provide a detailed experimental comparison between AdGraph and our work in Section VII-B.

Graph Neural Networks. Graphs are widely used to represent real-world objects and relationships in various domains such as social networks, knowledge graphs, traffic networks, and molecular structures. GNNs are one type of neural network that can directly operate on graphs for performing different learning tasks. A successful category of GNNs in recent years are convolutional GNNs [18]–[20], [24], [25]. Inspired by the success of convolutional neural networks (CNNs) [26], these GNNs achieve graph convolution operations by adopting the neighborhood aggregation strategy (i.e., message passing), where the representation of each node is iteratively learned by combining its feature vector and feature vectors aggregated from its neighbors. In practice, many GNN variants have achieved state-of-the-art performance for tasks in different domains such as point clouds classification and action recognition in computer vision [27]–[29], recommendation and spam detection in social networks [30]–[32], and molecular fingerprints and drug discovery in chemistry [24], [33], [34].

Motivated by the fact that HTTP network traffic can be represented by a graph and GNNs are successful in graph learning tasks, we in this work formulate WTA detection as a task of edge classification in an HTTP network traffic graph.

Especially, we design a specific GNN model that directly learns edge representation for performing this task.

III. Problem Definition

Our task of WTA detection is to detect whether some HTTP requests issued from a website are for the tracking and/or advertising purpose. Intuitively, one can build a graph to represent HTTP network traffic of one or more websites, in which an edge represents a specific HTTP request and a node represents either the source domain that issues an HTTP request or the destination domain that an HTTP request is sent to. We can hence consider our task of WTA detection as predicting the labels (i.e., WTA vs. non-WTA) of edges in the graph. In this section, we first introduce the definition of the attributed homogenous multi-graph that we aim to explore in this work. We then formally define WTA detection as a task of edge classification on such a graph.

Attributed Homogenous Multi-Graph. An attributed homogenous multi-graph (AHMG) is defined as a directed graph $G = (V, E, X, X^e)$, where each node $v \in V$ has a feature vector $x_v \in X$, and each edge $e_{(u,v,i)} \in E$ has a feature vector $x^e_{(u,v,i)} \in X^e$. Since multiple edges directed from node u to node v are permitted in G, we use index i to indicate the i^{th} edge among those multiple edges. In addition, self-loop (i.e., an edge directed from a node to the node itself) is also permitted in G.

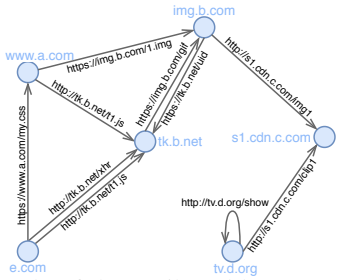


Fig. 1: An example of the attributed homogenous multi-graph constructed from HTTP network traffic.

AHMG is a natural representation of the HTTP network traffic on the web. Figure 1 provides an example. Each node represents a fully qualified domain name (FQDN) [35], while an edge represents a specific HTTP request issued from the source FQDN to the destination FQDN. Consistent with the characteristics of HTTP network traffic on the web, multiple edges exist between two FQDNs (e.g., *e.com* and *tk.b.net*) and they represent different HTTP requests, while self-loops (e.g., *http://tv.d.org/show*) also commonly exist in an AHMG. Besides, the properties of FQDNs and HTTP requests can be considered naturally as attributes of nodes and edges, respectively.

AHMG Edge Representation Learning and Classification. With the definition above, we can formally define our task of WTA detection as an AHMG edge representation learning and classification problem. Given an AHMG $G = (V, E, X, X^e)$, the goal is to design a model $f_\omega : x^e \rightarrow \mathbb{R}^d$ $(d \ll |x^e|)$ that learns the d-dimensional representation of each edge $e_{(u,v,i)}$ in G. The learned edge representation can be utilized in different tasks, such as edge clustering and edge classification. Specifically, our task in this work is to classify edges as WTA or non-WTA in an AHMG built from HTTP network traffic using the learned edge representations.

IV. Design

In this section, we first present the high-level architecture of our WTA detection framework WTAGRAPH. We then elaborate on the design rationale and implementation of its four components.

A. High-level Architecture of WTAGRAPH

After we formulated our problem as a task of edge representation learning and classification in an AHMG, we identified the following three challenges in our task.

- *How to extract and assign representative attributes for nodes and edges in an AHMG?* As explained in Figure 1, domain names and URLs are naturally attributes of nodes and edges. To achieve better learning and classification performance, it is crucial to effectively convert them into feature vectors and identify other representative features.
- *How to design and build a GNN model that directly learns edge representations in an AHMG?* There is no existing GNNs, to the best of our knowledge, for direct edge representation learning in an AHGM. It is therefore necessary to design a new GNN model that can learn edge representations in our graph.
- *How to effectively aggregate and propagate edge representations in that GNN?* The success of GNNs in other applications suggests that an effective aggregation and propagation strategy will be essential for our GNN.

Therefore, we propose WTAGRAPH to address these challenges and perform our task of WTA detection. Specifically, WTAGRAPH is designed for two usage settings: transductive learning setting and inductive learning setting. In the ***transductive learning setting***, WTAGRAPH is expected to detect WTA requests in the testing part of all observed data. That is, the edges in a given AHMG are split into training and testing datasets; WTAGRAPH is trained, during which the labels of edges in the training dataset and the whole structure of AHMG are observed, to infer edge labels in the testing dataset. In this setting, our GNN goes through both the forward pass and back-propagation for model training as well as for inferring unlabeled edges in the testing dataset. In practice, this usage setting corresponds to the application scenarios in which a batch of HTTP requests of multiple websites or webpages are added for prediction (i.e., their labels are unknown or uncertain); they belong to the testing dataset and are added to the existing AHMG (with labeled edges) to form a new AHMG for an entire transductive training/retraining and inferring process. For example, one can leverage WTAGRAPH's predictions in the transductive learning setting to help improve filter lists.

In the ***inductive learning setting***, WTAGRAPH is expected to detect WTA requests that have never been seen during the training process. That is, WTAGRAPH pre-trained (in the transductive learning setting above) on a pre-built AHMG

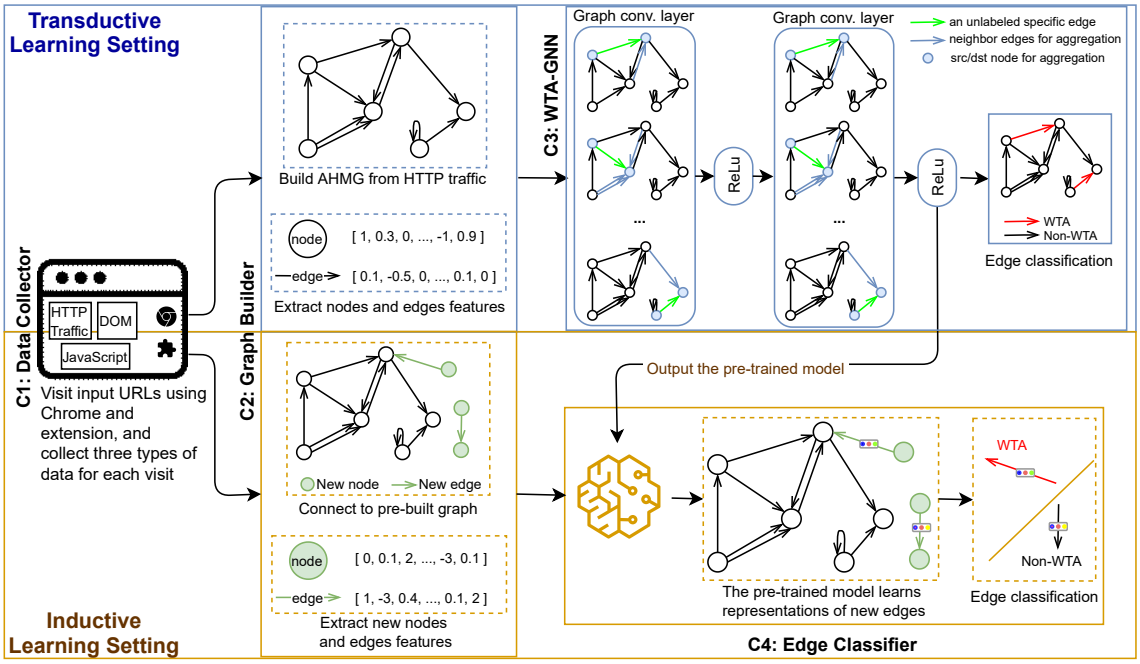


Fig. 2: The high-level architecture of our WTAGRAPH framework.

will be used to predict new or unseen edges by simply going through the forward pass. In practice, this usage setting corresponds to the application scenarios in which a pre-trained model is utilized to immediately predict new HTTP requests. In more details, the requests encountered in each webpage visit will immediately form a new testing AHMG, which will be connected (i.e., extended) for just one hop to the pre-built AHMG mentioned above if they share some nodes; this extended yet still much smaller testing AHMG will be the input to the pre-trained WtaGraph, and will simply go through the forward pass for edge representation learning and inference in real-time.

As shown in Figure 2, we design four components in WTAGRAPH for WTA detection in both transductive and inductive learning settings. The *data collector* component uses a Google Chrome browser extension to collect HTTP network traffic, document object model (DOM), and JavaScript API access for each webpage visit. The *graph builder* component builds an AHMG from the collected HTTP network traffic and extracts the corresponding node and edge features to address the first challenge. The *WTA-GNN* component is a novel GNN model for edge representation learning and WTA detection in the transductive learning setting. It captures the local graph structure, neighbor nodes' features, and neighbor edges' features for learning each edge representation. In our design of WTA-GNN we address the last two challenges. The *edge classifier* component uses a pre-trained WTA-GNN for WTA detection in the inductive learning setting.

B. Data Collector

The data collector component is designed to record the HTTP traffic (including all the redirected requests as in [17], [36]–[38]), DOM, and JavaScript API access for each webpage visit. The collected data can be then leveraged to build an AHMG and to extract the features of its of nodes and edges. We implemented the core functions of this data collector in a Google Chrome browser extension. In the data collection process, this extension can be installed in Google Chrome that is automated by Selenium [39] to crawl a list of websites.

In more details, we implemented the browser extension with the following capabilities. First, it can intercept and record all HTTP traffic during a webpage visit using Google Chrome's *webRequest* API. Second, it can automatically scroll down a webpage to load dynamic content (if any). Third, it can save the page source at the end of the visit so that DOM-related features can be extracted. Fourth, it can monitor JavaScript API access by instrumenting popular APIs that can be used for tracking and/or advertising. Lastly, it can save all crawled data to a NoSQL database on a server for later analysis.

C. Graph Builder

The graph builder component is designed to construct an AHMG from data crawled by the data collector and extract node and edge features of AHMG.

Graph Construction. In the transductive learning setting, the graph builder constructs an AHMG as exemplified in Figure 1 by converting all intercepted HTTP requests as edges and corresponding source and destination FQDNs of those requests as nodes. This AHMG is typically large, containing requests collected from many (e.g., thousands as in our experiments) websites. In the inductive learning setting, the graph builder converts the newly collected HTTP traffic of a single webpage visit into a new testing AHMG as described in Section IV-A. Depending on the existence of shared nodes between a new testing AHMG and the pre-built AHMG, three possible types of edges are added to the new testing AHMG. First, in the

case that both source and destination FQDNs exist as nodes in the pre-built AHMG, we add a new edge between the two nodes to represent a new HTTP request. Second, in the case that only one of two FQDNs exists as a node in the pre-built AHMG, we add a node representing the missing FQDN and a new edge between the two nodes. Third, in the case that both source and destination FQDNs do not exist in the pre-built AHMG, we add two nodes representing the two missing FQDNs and a new edge between them.

As one can imagine, isolated subgraphs could exist in an AHMG. For example, *x.com* may only send requests to *y.com* and there is no request between them and other FQDNs, thus creating an isolated subgraph that represents the unique traffic between these two FQDNs. Note that isolated subgraphs are not obstacles to the design and implementation of our WTA-GNN because typical GNNs can handle such cases (Section IV-D).

Feature Extraction. After an AHMG is constructed in either learning setting, the graph builder extracts two and three categories of features for nodes and edges, respectively.

The first node feature category is *character embeddings*. As explained in Section III, the FQDN strings can be considered naturally as attributes of nodes. Hence, a straightforward way to represent an FQDN as an input to our WTA-GNN is to use one-hot encoding. Each character in the FQDN string will be represented by a vector of bits (e.g., 26 bits if we only encode the lowercase alphabet), among which a 1-bit indicates the presence of a specific character. A representation of an FQDN, therefore, can be a big vector of concatenated one-hot vectors especially if the FQDN contains dozens of characters and if more characters (e.g., uppercase and special characters) need to be encoded.

Therefore, we decide to use the character embeddings yielded by the character CNN embedding model [40]. After training over one billion words [41], the character CNN model produces an embedding for each of 256 characters in the vocabulary. The embedding size of each character is 16. For example, the character 'a' is represented by the following vector of 16 real numbers:

```
[1.1014, -0.6760, 0.6962, ..., 1.0212]
```

In more details, we first set the length of an FQDN to 30 characters. FQDNs longer than 30 characters would be truncated from the 31^{st} character, and FQDNs shorter than 30 characters would be padded with a <PAD> token to length 30. We then concatenate all character embedding vectors following the character order in the FQDN. Note that for the <PAD> token, we fill its embedding vector with 16 zeros. Using this method, an FQDN string is represented by a feature vector of size 480.

The second node feature category is *DOM*. The DOM feature of a node is a vector of bits representing the presence of HTML tags in the DOM. This feature could help identify WTA requests because many WTA-related domains may not have DOM data (e.g., no DOM is available for an endpoint domain that only accepts POST requests). Note that for those FQDNs that have no DOM information collected, we fill the feature vector with all zeros.

The first edge feature category is also *character embeddings*. We truncate or pad a request URL to 200 characters, and concatenate each character embedding vector following the character order in the request URL. This brings each edge with a feature vector of size 3,200 that represents characters in the URL. Note that while an FQDN string is more related to a node, a request URL is more specific to an edge in our AHMG. Hence, we consider the corresponding character embeddings of a request URL as edge features.

The second edge feature category is *JavaScript API access*. The JavaScript feature of an edge is a vector of bits representing the access to instrumented APIs by the script received from a JavaScript file request. The features in this category could help distinguish between functional and WTA-related JavaScript because WTA-related JavaScript is known for accessing certain APIs to perform tracking and/or advertising [1], [42]. In addition, we consider the presences of `async` and `defer` attributes in the `script` HTML tag of a JavaScript file request as two binary features. The high information gain of these two features has been evaluated and confirmed by authors in [17]. For non-JavaScript requests, we fill their JavaScript feature vectors with all zeros.

The third edge feature category is *request statistics*. For a request, we extract the following informative attributes: source frame, order, timing, type, method, HTTP cookies, and URL length. These attributes could also help characterize a WTA request. For example, a WTA request usually carries data in its URL (e.g., in the query section), which makes its URL length often longer than that of a non-WTA request. These attributes can be extracted from the saved HTTP traffic. Specifically, by using the chrome.webRequest API [43] in our data collector extension, we can extract the details of each HTTP request. For example, given an HTTP request, the source frame feature is a bit that indicates whether the request happens in the main frame or not, and can be extracted from the `frameId` property; the order feature is an integer derived by checking the current request's index in a sequence of requests ordered by the `timeStamp` property. This ends up with a feature vector of size 34 for an edge.

D. WTA-GNN

Given a constructed AHMG and its node and edge features, our task is to predict whether an edge (representing a specific HTTP request) is WTA-related.

Design Choices, Decision, and Justification. To perform this task, there are at least four potential solutions. First, one can train a traditional machine learning classifier, such as a random forest classifier, to predict labels of edges. Second, one can build a traditional neural network, e.g., a multilayer perceptron (MLP), to classify edges by learning their representations. Third, following the typical link prediction (i.e., predicting the possible presence of an edge between two nodes) approach [44], [45], one can build a GNN model to learn node representations, and predict the label of each edge using the derived

edge representation (e.g., by concatenating or averaging the corresponding source and destination node representations). Fourth, one can build an end-to-end GNN model that directly learns edge representations and predicts their labels.

However, there are some shortcomings in the first three solutions. In either the first or the second solution, mainly the edge features are used for the prediction. Either solution would miss the opportunity to leverage features of neighbor nodes and latent graph structural features that could intuitively contribute to better classification performance. The third solution is also inappropriate for solving our problem, even though it leverages GNN to explore both the node and graph structural features. Typical link prediction GNNs are built to process simple graphs, while our graph is a multi-graph. As a consequence, multiple edges between the same pair of nodes in our graph will have the same edge representation if we derive their representations simply from node representations. This will lead to inaccurate classification results. We evaluate and compare these three solutions in Section VI-A.

Therefore, we set out to explore the fourth solution to perform our task. We build a specific GNN model that directly learns edge representations by incorporating edge features, node features, and latent graph structural features. Our model, referred to as WTA-GNN, can be used in both transductive and inductive learning settings.

WTA-GNN Design. Typical GNNs (e.g., GCN [18], GAT [20], and GraphSage [19]) learn node representations in a homogenous simple graph by incorporating neighbor node features and graph structural features for tasks such as node classification, node clustering, and link prediction. To the best of our knowledge, no study has been conducted specifically for edge representation learning on AHMG. Therefore, inspired by the design of typical GNNs, we design WTA-GNN for edge representation learning on our AHMG.

The node representation learning process of a typical GNN can be boiled down to the following two steps. First, it aggregates neighbor nodes' representations for each node. Second, it propagates each node's representation by incorporating its representation and the aggregated representations. Various GNN studies have demonstrated that such an aggregation and propagation process can yield better performance on graph-related tasks. We hypothesize that a similar aggregation and propagation process could be directly applied for edge representation learning to help achieve a good performance on our task of WTA detection in an AHMG.

Hence, we design the edge representation learning process in our WTA-GNN as follows. First, it aggregates neighbor edges' representations for each edge. Specifically, we consider the incoming edges of the same destination node as a given edge's neighbors for aggregation. The intuition behind this design is that HTTP requests sent to the same FQDN likely have the same purpose especially from the WTA vs. non-WTA perspective. Hence such an aggregation could help propagate and learn an edge representation for accurate classification. Second, it incorporates the representations of two nodes of each edge. The intuition behind this design is that node representations could further encode graph structural information and help classify an edge. Third, similar to the node representation learning, it propagates each edge representation by incorporating its representation and aggregated edge and node representations through multiple layers of computation.

Algorithm 1: WTA-GNN edge representation learning

Input: AHMG $G = (V, E, X, X^e)$; node feature vector $x_v \in X$; edge feature vector $x^e_{(u,v,i)} \in X^e$; # of layers L; node weight matrix W_V; edge weight matrix W_E; activation function σ; neighborhood function N; aggregation function Agg

Output: edge representations $z_{e_{(u,v,i)}}, \forall e_{(u,v,i)} \in E$

1 $h^0_{e_{(u,v,i)}} \leftarrow x^e_{(u,v,i)}, \forall e_{(u,v,i)} \in E$

2 $s^0_v \leftarrow x_v, \forall v \in V$

3 **for** $l = 1...L$ **do**

4 **for** $v \in V$ **do**

5 $s^l_{N(v)} \leftarrow Agg\left(s^{l-1}_u, \forall u \in N(v)\right)$

6 $s^l_v \leftarrow \sigma\left(W^l_V \cdot \left(s^{l-1}_v + s^l_{N(v)}\right)\right)$

7 **end**

8 **for** $e_{(u,v,i)} \in E$ **do**

9 $h^l_{N(e_{(u,v,i)})} \leftarrow Agg\left(h^{l-1}_j, \forall j \in N(e_{(u,v,i)})\right)$

10 $t^l \leftarrow Average\left(s^l_u,\ s^l_v\right)$

11 $h^l_{e_{(u,v,i)}} \leftarrow \sigma\left(W^l_E \cdot Concat\left(h^{l-1}_{e_{(u,v,i)}} + h^l_{N(e_{(u,v,i)})},\ t^l\right)\right)$

12 **end**

13 **end**

14 $z_{e_{(u,v,i)}} \leftarrow h^L_{e_{(u,v,i)}}, \forall e_{(u,v,i)} \in E$

Algorithm 1 details the edge representation learning process in WTA-GNN. It takes as input an AHMG $G = (V, E, X, X^e)$ and the learnable weight matrices W_V and W_E, and outputs a d-dimensional edge representation $z_{e_{(u,v,i)}}$ for each edge. In more details, L defines the number of layers in the neural network and l in the outer loop denotes a specific layer. Note that the base case $l = 0$ defines the initial input layer, where features extracted by the graph builder component are provided to WTA-GNN (Lines 1 and 2). In each layer l, WTA-GNN first learns node representations in the first inner loop (Lines 4 to 7), and then learns edge representations in the second inner loop (Lines 8 to 12). In the first inner loop, each node representation s^l_v is produced by applying an activation function to the weighted sum of neighbor nodes' representations (aggregated in Line 5) and its own representation from layer $l-1$. In the second inner loop, WTA-GNN first aggregates neighbor edges' representations into a single vector $h^l_{N(e_{(u,v,i)})}$ (Line 9), then combines the representations of the edge's two nodes into a vector t^l by averaging s^l_u and s^l_v (Line 10), and lastly produces edge representation $h^l_{e_{(u,v,i)}}$ by applying an activation function to the weighted concatenated representation vector (Line 11). Specifically, WTA-GNN concatenates the combined vector t^l (which represents a partial edge representation inherited merely from the two nodes) and the sum of

an edge representation $h^{l-1}_{e_{(u,v,i)}}$ and the aggregated neighbor edges' representation $h^{l}_{N(e_{(u,v,i)})}$.

As shown in Figure 2, WTA-GNN is trained in the transductive learning setting, where edges in the input AHMG are split into the training set and testing set. During the training process, we apply the cross-entropy loss function to the output representation $z_{e_{(u,v,i)}}$ for edges in the training set, and tune the weight matrices W_V and W_E via stochastic gradient descent. Cross-entropy loss is popularly used in GNNs (such as [18], [20]) for classification tasks. In our case, it measures the distance between raw (i.e., without normalization) edge classification logits and ground truth labels. The training process minimizes the average cross-entropy loss across all training edges. The trained WTA-GNN can output representations of edges in the testing set and predict their labels. Furthermore, the trained WTA-GNN can be used in the inductive learning setting to immediately predict the labels of new edges in a testing AHMG constructed for each webpage visit.

WTA-GNN Implementation. We implement WTA-GNN to have two layers (i.e., L=2) using the GNN framework in DGL [46]. Furthermore, we implement the aggregation function Agg as a mean aggregator, which takes the elementwise mean of vectors in $\left\{s^{l-1}_u, \forall u \in N(v)\right\}$ and $\left\{h^{l-1}_j, \forall j \in N(e_{(u,v,i)})\right\}$ in Lines 5 and 9, respectively. We implement the neighborhood function N to select the complete set of neighbors of a node or edge. In fact, one can choose different aggregation and neighborhood functions in Algorithm 1. For example, one might implement the neighborhood function N to only select a subset of neighbors such as in [19]. Note that for edges or nodes that have no neighbors (e.g., in an isolated subgraph that only has one edge or even one node), neighbor selection and aggregation will not happen; hence the corresponding edge or node representation is purely learned from its own initial features.

E. Edge Classifier

The last component of WTAGRAPH is the edge classifier. It loads the WTA-GNN model pre-trained in the transductive learning setting to predict the labels of new edges in a testing AHMG. Specifically, it classifies a new edge as WTA or non-WTA based on the WTA-GNN output $z_{e_{(u,v,i)}}$. The vector $z_{e_{(u,v,i)}}$ in Algorithm 1 is of size two in our design, which corresponds to two possible classes (i.e., WTA and non-WTA) of an edge. The edge classifier takes the class that has the larger logit value as the classification result of an edge.

V. DATA COLLECTION AND DATASET

Data Collection. As explained in Section IV-B, we implemented the data collector component of WTAGRAPH as a Google Chrome browser extension. In our data collection, we installed the data collector in the instrumented Chromium browser [47] of AdGraph [17]. In this way, two separate data collection operations can be performed during the same webpage visit with no interference to each other. We selected the top 10K websites from the Alexa top one million list dated on June 8th, 2020 for data collection. For each website, we automatically visited its homepage using the instrumented Chromium browser installed with WTAGRAPH's data collector from July 23rd, 2020 to August 6th, 2020. In each webpage visit, we waited for the homepage to finish loading and scrolling (triggered by WTAGRAPH's data collector) or at most 120 seconds. The instrumented Chromium browser collected webpage execution context data, while WTAGRAPH's data collector saved the HTTP traffic, DOM, and JavaScript API access information.

Ground Truth. To train and evaluate WTAGRAPH, we created the ground truth labels of all collected HTTP requests by leveraging the same seven filter lists[1] as used in AdGraph [17]. Specifically, they are EasyList [8], EasyPrivacy [9], Anti-Adblock Killer [48], Warning Removal List [49], Blockzilla [50], Peter Lowes's List [51], and Fanboy Annoyances List [52]. We did not cover the Squid Blacklist used in [17] as it was no longer available at that time. Note that filter lists could have false positives and false negatives as studied in [10], [11], but using their outputs as ground truth labels for model training and quantitative evaluation is still reasonable and viable especially for Top 10K websites as analyzed in [17]. We have manual verification on a set of sampled classification results as detailed in Section VI-B and Appendix A. For each collected HTTP request, we labeled it as WTA or non-WTA based on whether it would be blocked by any of these seven filter lists.

Dataset Statistics. Table I summarizes the statistics of HTTP requests collected by WTAGRAPH's data collector and by AdGraph's Chromium browser [47] from Alexa Top 10K websites in the same visits. There are over 1.55 million and 834K requests collected by WTAGRAPH and AdGraph, respectively; using the same seven filter lists, 39% and 15% of these requests collected by WTAGRAPH and AdGraph are labeled as WTA requests, respectively. From the perspective of WTAGRAPH, it missed around 15% of requests collected by AdGraph. From the perspective of AdGraph, it missed over 952K (61%) of requests collected by WTAGRAPH, and over half of those missed requests are WTA requests. We investigated this data discrepancy and found that WTAGRAPH collected data correctly, while AdGraph mistakenly collected some URLs that are not HTTP requests in practice and simply missed some types of HTTP requests due to the potential incompleteness of its Chromium instrumentation. More details about this discrepancy analysis are in Appendix B.

We refer to the dataset (in the first column of Table I) collected by WTAGRAPH's data collector as our Top-10K dataset. This dataset will be used in Section VI and Section VII. The overlapped data (in the third column of Table I) between our Top-10K dataset and the dataset collected by AdGraph during the same webpage visits will be used in Section VII-B for comparing WTAGRAPH with AdGraph.

VI. EVALUATION IN TRANSDUCTIVE SETTINGS

In this section, we present the performance of WTAGRAPH on detecting WTA requests in transductive learning settings.

[1] All of them are dated on June 8th, 2020.

Table I: Statistics of HTTP requests collected by WTAGRAPH and by AdGraph from Alexa Top 10K websites.

Targeted Websites	WTAGRAPH Collected		AdGraph Collected		Overlapped Between Them		Missed By AdGraph	
	# Requests	# (%) Blocked by Filter Lists	# Requests	# (%) Blocked by Filter Lists	# Requests	# (%) Blocked by Filter Lists	# Requests	# (%) Blocked by Filter Lists
Top-10K	1,559,602	614,001 (39%)	834,439	129,004 (15%)	607,124	117,884 (19%)	952,478	496,117 (52%)

We describe its overall performance, perform manual verification, and provide the results of ablation studies on WTAGRAPH variants. Recall that transductive learning settings correspond to the application scenarios in which a batch of HTTP requests of multiple websites or webpages are added for prediction as described in Section IV-A.

A. Overall Performance

Baseline Models. Section IV-D introduced three other potential solutions besides WTAGRAPH to our problem of WTA detection. We implemented two of them, a typical link prediction GNN model and a traditional multilayer perceptron (MLP), as baseline models for comparison with WTAGRAPH. Note that we present the comparison between a traditional machine learning classifier (i.e., a random forest model used in AdGraph [17]) and WTAGRAPH in Section VII-B.

For the link prediction GNN model, we implemented a two-layer GCN [18] model that learns node representations and generates edge representations by averaging the learned node representations of each edge's two nodes. For the MLP, we implemented it with one hidden layer; it takes as input the raw request features (i.e., both node and edge features extracted by WTAGRAPH) and outputs the learned feature vector of size two for each request. Besides, we proposed and implemented an advanced link prediction GNN as a baseline model. It is similar to WTA-GNN but does not aggregate neighbor edges' representations during the learning process. We detailed the experimental setup in Appendix C. We evaluated these three baseline models and WTAGRAPH on our Top-10K dataset through the stratified 10-fold cross-validation.

Overall Results. Table II summarizes the performance of these four models on our Top-10K dataset, in terms of classification accuracy, precision, recall, and F_1 score. Overall, WTAGRAPH outperformed all three baseline models on every evaluation metric. For example, the detection accuracy, precision, recall, and F_1 score of WTAGRAPH are 97.90%, 98.38%, 96.25%, and 97.30%, respectively. The ROC and AUC analysis of these four models is in Appendix D.

Table II: Summary of the models' performance for WTA detection on our Top-10K dataset.

Model	Acc.	Prec.	Recall	F_1
Link Pred. GNN	94.20%	93.63%	91.48%	92.54%
MLP	94.89%	94.57%	92.33%	93.44%
Adv. Link Pred.	97.63%	97.90%	96.03%	96.96%
WTAGRAPH (i.e., WTA-GNN)	**97.90%**	**98.38%**	**96.25%**	**97.30%**

There are two major implications from these results. On the one hand, our proposed WTAGRAPH and advanced link prediction model outperform the rest two baseline models. This implies that benefiting from the neighbor aggregation and implicit graph features, a purposefully designed GNN model can outperform typical models in terms of detecting WTA requests. On the other hand, by comparing the aggregation strategy in WTAGRAPH (i.e., aggregating from both nodes and edges) to that in our advanced link prediction GNN (i.e., only aggregating from nodes), we found that the former results in a better detection performance.

Detailed Performance. Table III summarizes the detailed detection performance of WTAGRAPH on 12 types of HTTP requests in our Top-10K dataset. The HTTP requests collected by WTAGRAPH covered all 12 request types as documented for Google Chrome [53]. As shown on the left side of Table III, the number of WTA requests detected by WTAGRAPH is very close to that by filter lists for each request type. Overall, 38.52% and 39.37% of all requests are detected as WTA by WTAGRAPH and filter lists, respectively. This result implies that WTAGRAPH can be an alternative to filter lists because of its proximate detection rate on WTA requests. Meanwhile, WTAGRAPH has a remarkable performance on the majority of request types. For example, WTAGRAPH achieves over 95% accuracy, precision, recall, and F_1 score for the top four request types (i.e., `image`, `script`, `xmlhttprequest`, and `other` accounting for 86.34% of all requests in the dataset). WTAGRAPH has the relatively worse performance on detecting `main_frame`, `stylesheet`, and `font` requests. For example, the false negative rates for these three types of requests are 69.50%, 16.20%, and 10.41%, respectively.

B. Manual Verification

To verify WTAGRAPH's predictions, we conducted a set of manual verifications on sampled requests. Specifically, from all predictions of 12 request types, we randomly sampled 846 false positives, 941 false negatives, 1,040 true positives, and 1,139 true negatives; we manually assigned each of them a label of WTA, non-WTA, mixed (only used for script resources as in [17]), or undecidable after verification. Due to the space limitation, we detailed the complete verification procedure, criteria, and results in Appendix A.

Overall, from 846 sampled false positives, we found that WTAGRAPH is able to identify WTA requests that are missed by filter lists, especially for types of `image`, `script`, `xmlhttprequest`, `sub_frame`, `ping`, and `media`. Specifically, 320 out of 846 requests are verified as WTA requests, which means WTAGRAPH's classification is correct. For example, while filter lists missed, WTAGRAPH successfully detected the following tracking pixel `https://ss0.baidu.com/6ONWsjip0QIZ8tyhnq/ps_default.gif`. From 941 sampled false negatives, we verified that WTAGRAPH's classification (i.e., a non-WTA prediction) is indeed correct over 90% of the time for `stylesheet`, `main_frame`, `other`, and `font` requests. That is, filter lists mistakenly labeled those requests as WTA. For example, Peter Lowes's List [51] aggressively blocks

Table III: The detailed WTA detection performance of WTAGRAPH on 12 types of HTTP requests in the Top-10K dataset.

Request Type	# Requests	# Blocked by Filter Lists	# Detected by WTAGRAPH	Accuracy	Precision	Recall	F_1	FNR	FPR
image	700,670	240,176	232,539	98.43%	99.28%	96.12%	97.68%	3.88%	0.36%
script	364,787	208,697	205,169	96.51%	97.76%	96.11%	96.93%	3.89%	2.94%
xmlhttprequest	151,755	86,432	85,409	96.54%	97.52%	96.37%	96.94%	3.63%	3.24%
other	129,289	5,534	5,323	99.76%	99.06%	95.28%	97.14%	4.72%	0.04%
stylesheet	63,971	5,044	4,407	98.44%	95.92%	83.80%	89.45%	16.20%	0.31%
sub_frame	63,383	50,999	51,037	97.42%	98.36%	98.44%	98.40%	1.56%	6.75%
font	42,386	1,210	1,112	99.64%	97.48%	89.59%	93.37%	10.41%	0.07%
main_frame	23,554	200	130	99.12%	46.92%	30.50%	36.97%	69.50%	0.30%
ping	15,560	14,594	14,462	97.71%	99.23%	98.33%	98.77%	1.67%	11.59%
media	3,945	1,027	1,087	96.86%	91.54%	96.88%	94.13%	3.12%	3.15%
csp_report	263	88	86	93.92%	91.86%	89.77%	90.80%	10.23%	4.00%
object	39	0	0	100%	-	-	-	-	0.00%
Total	1,559,602	614,001	600,761	97.90%	98.38%	96.25%	97.30%	3.75%	1.03%

any request to `criteo.com`; as a consequence, when one visits `criteo.com` as a first-party website, all requests to `criteo.com`, including the `main_frame` request, will be blocked by this filter list.

From 1,040 sampled true positives and 1,139 sampled true negatives, we observed that the majority (89.67%) of the agreements between WTAGRAPH and filter lists are correct, while they both could be incorrect on around 5% requests (excluding undecidable ones) likely due to the imperfection and incompleteness of filter lists. For example, both WTAGRAPH and filter lists did not detect the following tracking pixel `https://c.evidon.com/a/4.gif`.

C. Ablation Analysis

To figure out how different categories of features and neighbor aggregation strategies impact on WTAGRAPH's performance, we conducted ablation studies on four WTAGRAPH variants: (1) variant without node aggregation, (2) variant without edge URL embedding features, (3) variant without edge JavaScript features, and (4) variant without edge URL statistical features. Note that the variant of WTAGRAPH without neighbor edge aggregation (i.e., the advanced link prediction GNN) has been evaluated and reported in Table II.

Figure 3 presents the performance of these four variants and the original WTAGRAPH. Overall, the performance of four WTAGRAPH variants is slightly worse than that of the original WTAGRAPH. Specifically, without node aggregation, WTAGRAPH's classification accuracy, precision, recall, and F_1 score drop 0.72%, 0.95%, 0.89%, and 0.91%, respectively; without edge URL embedding features, its accuracy, precision, recall, and F_1 score drop 0.59%, 0.71%, 0.81%, and 0.76%, respectively; without edge URL statistical or JavaScript features, WTAGRAPH's classification performance declines slightly with around 0.10%.

These results imply that: (1) WTAGRAPH's performance is contributed by all features and aggregation strategies rather than certain dominating factors; (2) WTAGRAPH is flexible in terms of deployment options in practice, for example, one can remove certain features if there are constraints of feature availability or computational resource, and expect WTAGRAPH to still have a competitive performance; (3) WTAGRAPH can be robust against some WTA detection evasions, such as DGA domains [54] and JavaScript obfuscation [55], because it does not heavily rely on certain specific features.

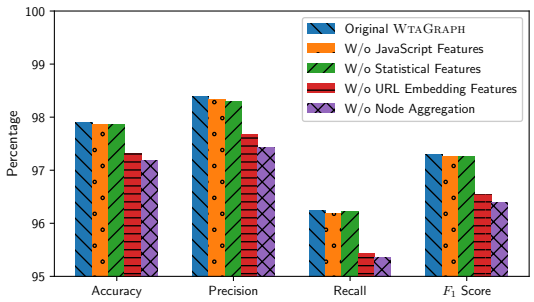


Fig. 3: The performance of WTAGRAPH variants with different categories of features or aggregation strategies ablated.

VII. EVALUATION IN INDUCTIVE SETTINGS

In this section, we evaluate WTAGRAPH in different inductive learning settings. We first present its overall performance. Then, we conduct a detailed comparison between WTAGRAPH and AdGraph [17]. Third, we perform evasion studies to evaluate the robustness of WTAGRAPH. Lastly, we evaluate the real-time performance of WTAGRAPH. Recall that inductive learning settings correspond to the application scenarios in which a pre-trained model is utilized to immediately predict new HTTP requests (encountered in each webpage visit) that form a new testing AHMG as described in Section IV-A.

A. Overall Performance

All the evaluations in the previous section are performed on a full graph in transductive learning settings. To evaluate WTAGRAPH's performance in inductive learning settings, we first transductively trained four WTAGRAPH models on different graphs that are built from randomly selected websites. Specifically, we randomly selected 1K, 3K, 5K, and 7K websites from our Top-10K dataset for transductive training and used each of the remaining (unseen) 9K, 7K, 5K, and 3K websites for inductive testing on their homepages. We built four graphs referred to as the Random-1K, Random-3K, Random-5K, and Random-7K[2] graphs, and pre-trained four WTAGRAPH models on these graphs, respectively. We then tested these four pre-trained WTAGRAPH models on the corresponding remaining websites. In more details, we built a testing AHMG for each individual testing webpage and

[2]For a common data split (80% for training and 20% for testing) and fair comparison with AdGraph in the next subsection, this Random-7K graph indeed contains 7,224 instead of 7,000 websites.

extended it with one-hop neighbors of its nodes and edges in the corresponding pre-built AHMG (Section IV-A). We used the corresponding pre-trained model to predict WTA requests in each testing AHMG.

It is worth highlighting that our testing AHMG construction approach is both reasonable and effective. In inductive learning settings, the input to a pre-trained model for prediction or testing must be an AHMG. Though one can build a testing AHMG that contains only one edge representing an individual request in a webpage visit, WTAGRAPH will perform poorly (as we initially observed) because such an AHMG cannot well leverage implicit graph features and neighbor information. Constructing a testing AHMG for multiple requests of each webpage visit will enable rich feature extraction from both the content and context perspectives; meanwhile, by taking this approach, there is no need or dependency on visiting more webpages or websites. In other words, this is how we address in our WTA detection task the so-called cold-start problem or challenge (i.e., feasibly and properly constructing test graphs) that is common in the inductive learning of GNNs [56], [57]. Note that while AdGraph constructs a totally different graph, its graph construction and feature extraction are also based on multiple requests of a webpage visit; thus, the same input unit (i.e., information collected from each webpage visit) will be provided to WTAGRAPH and AdGraph for a fair comparison.

Table IV: The statistics of training graphs and testing websites in each inductive learning setting. Note that the full graph of this Top-10K dataset has 54,109 nodes and 1,559,602 edges.

Four Training Graphs			Testing Graphs (i.e., Sites)	
# Sites	# Nodes	# Edges	# Sites	# Edges
Random-1K	8,296	149,771	8,982	1,409,831
Random-3K	20,931	482,722	6,982	1,076,880
Random-5K	30,811	787,107	4,982	772,495
Random-7K	44,246	1,234,148	2,758	325,454

Table IV summarizes the statistics of our dataset split in each inductive setting. Taking the Random-1K graph as an example, it only contains 9.60% edges and 15.33% nodes that appeared in the Top-10K dataset. In this inductive learning setting, the WTAGRAPH model trained on the Random-1K graph will be used to infer labels of over 1.4 million requests of 8,982 websites that have never been seen during the training process. Note that a testing AHMG will be created for each of these 8,982 websites and will be independently tested to infer the labels of its edges. On average, each testing AHMG has 27 nodes and 129 edges.

Figure 4 presents the performance of WTAGRAPH models in four inductive learning settings. The blue bars, plotted for the comparison purpose, represent the performance of the WTAGRAPH in the transductive learning setting as shown in Table II. We observed that except for the WTAGRAPH model trained on the Random-1K graph, the rest WTAGRAPH models can achieve competitive performance in inductive learning settings, comparing with the performance of WTAGRAPH in the transductive learning setting. For example, the WTAGRAPH model on the Random-7K graph can predict the 325,454 requests of 2,758 testing websites with an accuracy 97.82%,

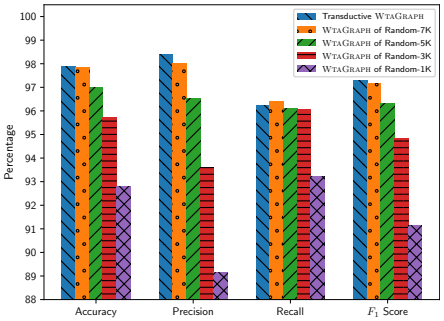


Fig. 4: The performance of WTAGRAPH models (trained on the Random-1K, Random-3K, Random-5K, and Random-7K graphs) in inductive learning settings.

precision 98.00%, recall 96.38%, and F_1 score 97.18%. Besides, we found that the larger the training graph, the better the performance in the inductive learning setting. This is as expected because a larger graph will likely capture more HTTP traffic patterns and include representative edges for aggregation. These results imply that WTAGRAPH is able to perform well in inductive learning settings when the training dataset is not too small. The ROC and AUC analysis for the models in this and next subsections is in Appendix D.

B. Comparison with AdGraph

AdGraph [17] only collected 38.93% (i.e., 607,124) of requests in our Top-10K dataset (summarized in Table I), and does not directly work on our Top-10K dataset because the context information for the requests missed by AdGraph is not available. To feasibly compare our WTAGRAPH with AdGraph, we evaluated their performance on these 607,124 overlapped requests in our Top-10K dataset.

As shown at the top of Table V, we randomly selected 80% websites (i.e., 7,224 of 9,032 overlapped websites) for training and 20% websites for testing. We also divided 1,808 testing websites into two categories: connected and isolated websites. A website is considered as connected if there exists at least one node in its AHMG also appearing in the training graph; otherwise, it is considered as isolated. In other words, the AHMG of a connected website has at least one edge connecting to the training graph. There are 1,714 connected and 94 isolated websites in total. Note that isolated websites barely exist (three isolated websites in total) in our Top-10K dataset[3] as WTAGRAPH collects the complete requests of visited websites and it is more likely for websites to connect with each other based on the common requests.

We trained the WTAGRAPH model and the random forest model in AdGraph using the same training data, and reported their classification performance on the same testing data in Table V. On the one hand, we found that WTAGRAPH outperforms AdGraph in terms of classification accuracy, precision, recall, and F_1 score by 1.46%, 4.05%, 3.99%, and 4.02%, respectively, on predicting the requests of the 1,808 testing

[3]Which differs from the overlapped dataset. Isolated websites also barely exist in our two other datasets (not included in this paper): zero in the random 10K and three in the bottom 10K of the Alexa top one million site list.

Table V: The data split and performance of WTAGRAPH and AdGraph in inductive learning settings on the overlapped dataset.

	Training				**Testing**											
					Overall				Case: Connected				Case: Isolated			
	479,313 requests of 7,224 websites				127,811 requests of 1,808 websites				125,018 requests of 1,714 websites				2,793 requests of 94 websites			
Model	Acc.	Prec.	Recall	F_1	Acc.	Prec.	Recall	F_1	Acc.	Prec.	Recall	F_1	Acc.	Prec.	Recall	F_1
AdGraph	95.71%	88.38%	89.73%	89.05%	94.22%	84.57%	83.77%	84.17%	94.20%	84.87%	83.95%	84.41%	95.13%	11.58%	17.46%	13.93%
WTAGRAPH	97.85%	96.75%	92.20%	94.42%	95.68%	88.62%	87.76%	**88.19%**	95.68%	88.89%	87.92%	88.40%	95.81%	17.98%	26.67%	21.48%

websites. It is worth mentioning that we trained a WTAGRAPH model using the complete data (instead of the overlapped data) of the same 7,224 websites and reported its accuracy 97.82%, precision 98.00%, recall 96.38%, and F_1 score 97.18% right above in Section VII-A. We can see that its performance on the overlapped dataset drops by 2.14% to 9.38% on different evaluation metrics. There are two possible reasons for this performance discrepancy. First, the skewed data (i.e., the overlapped dataset) were used for training WTAGRAPH. For example, only 19% of overlapped requests in the Top-10K dataset are labeled as WTA by filter lists, while 39% of all requests are labeled as WTA (see Table I). Second, the training graph built from incomplete requests is not fully representative of the network traffic in the wild, thus the trained model cannot achieve the same performance as in the previous subsection.

On the other hand, we found that AdGraph missed the collection and correspondingly classification of a large volume of WTA requests. For example, as shown in Table I, among over 614K requests that are labeled as WTA by filter lists in our Top-10K dataset, AdGraph missed over 496K (81%) WTA requests and only collected around 19% of WTA requests.

We further randomly sampled and manually verified 200 disagreements between AdGraph and WTAGRAPH, following the same verification methodology and criteria as described in Appendix A. In more details, we first sampled 100 requests that WTAGRAPH classified as WTA but AdGraph classified as non-WTA. We found that WTAGRAPH's classifications on 62 requests are consistent with their labels from filter lists, while this number is 38 for AdGraph. After manually verifying these 100 requests, we found that among 91 requests whose labels can be determined, WTAGRAPH correctly classified 57 of them while AdGraph correctly classified 34 of them. We further sampled 100 requests that WTAGRAPH classified as non-WTA but AdGraph classified as WTA. We found that WTAGRAPH's classifications on 55 requests are consistent with their labels from filter lists, while this number is 45 for AdGraph. Similarly, after manually verifying these 100 requests, we found that among 84 requests whose labels can be determined, WTAGRAPH correctly classified 51 of them while AdGraph correctly classified 33 of them.

Furthermore, we randomly sampled and manually verified 200 agreements between AdGraph and WTAGRAPH. Specifically, among 100 requests that are classified as WTA by them, all are labeled as WTA by filter lists; we verified that 96 of them are indeed WTA requests while four requests cannot be determined. Among 100 requests that are classified as non-WTA by them, only one is labeled as WTA by filter lists; we verified that 91 of these 100 requests are indeed non-WTA, while eight requests cannot be determined. Besides, the one that is labeled as WTA by filter lists is indeed a WTA request.

In summary, we found that WTAGRAPH outperforms AdGraph from three aspects. First, WTAGRAPH classifies the overlapped requests more accurately than AdGraph. Second, WTAGRAPH's classifications on those sampled disagreements between AdGraph and WTAGRAPH are correct at least 60% of the time. Third, AdGraph missed the opportunity to collect and classify a large volume of WTA requests. There are two possible reasons for WTAGRAPH outperforming AdGraph. First, our WTAGRAPH model can utilize both explicit features of requests and implicit features learned from the graph for detecting WTA requests, while a traditional random forest model in AdGraph only used the explicit features of requests. Second, the AHMG built by WTAGRAPH to represent HTTP network traffic and the relationship between nodes could be more informative and helpful than the graph built by AdGraph to represent the context for an individual webpage in terms of characterizing WTA requests. Note that neither WTAGRAPH nor AdGraph performs well on the 94 isolated testing websites, likely related to the heavy skewness of the data as only 2.15% of 2,793 requests are WTA requests. For WTAGRAPH, another reason is that the edge or node representation in an isolated subgraph is purely learned from its own initial features.

C. Evasion Analysis

Recall our ablation analysis results in Section VI-C imply that WTAGRAPH could be robust against evasions because it does not heavily rely on certain specific features. We now evaluate its robustness under different URL-related evasion techniques in inductive learning settings.

We consider four evasion techniques for evaluation: changing second-level domain, changing lower-level (i.e., 3rd level, 4th level, etc.) domains, expanding URL length, and replacing sensitive keywords. In changing second-level domain, we replace the second-level domain with a randomly generated string of the same length. In changing lower-level domains, we replace each of them (if any) with a randomly generated string of the same length. Because it is possible for motivated trackers to change second-level or lower-level domains to evade detection, we evaluate these two evasion techniques. In expanding URL length, we pad the URL path section at its end with a randomly generated string so that the length from the beginning to the end of the URL path section will be 200 characters. Because WTAGRAPH truncates a URL at length 200 for feature extraction, this padding will force WTAGRAPH to truncate the remaining characters (e.g., in the query string), thus allowing us to evaluate whether the truncated information would help attackers evade our WTAGRAPH. Note that padding the beginning of a URL (i.e., the domain name) is similar to the evasion of changing lower-level domains. In replacing sensitive keywords, we replace the keywords "ad"

and "track" in the URL with randomly generated strings of the same length. We select "ad" and "track" as keywords based on the fact that they are included in 43.36% rules of EasyList [8].

We used the WTAGRAPH model trained on the Random-7K graph (described in Section VII-A) for evasion evaluations. For each labeled WTA URL of a testing webpage, we first adopted the aforementioned evasion techniques individually to modify the URL, and extracted new features for the modified URL. We then tested WTAGRAPH to predict WTA URLs in each testing webpage. Following the same evasion procedure, we also tested AdGraph and filter lists for comparison purposes.

Table VI: The success rates of four evasion techniques targeting WTAGRAPH, AdGraph, and filter lists.

Evasion Technique	Evasion Success Rate		
	WTAGRAPH	AdGraph	Filter Lists
Changing Second-level Domain	3.11%	1.03%	57.78%
Changing Lower-level Domains	2.11%	0.67%	9.13%
Expanding URL Length	2.61%	21.84%	2.45%
Replacing Keywords	0.22%	6.74%	12.54%

Table VI summarizes the success rates of four evasion techniques targeting WTAGRAPH, AdGraph [17], and filter lists. Overall, WTAGRAPH is robust against all these four evasion techniques. For example, only 0.22% and 2.61% WTA URLs successfully evaded WTAGRAPH's detection by replacing sensitive keywords and expanding URL length, respectively. Comparatively, 21.84% WTA URLs evaded AdGraph's detection by expanding URL length. Such a high evasion success rate on AdGraph is expected because URL length is reported in AdGraph as one of two content features that "provided the highest information gain" [17]. Besides, by replacing keywords, 6.74% WTA URLs evaded AdGraph's detection. This is mainly because AdGraph uses the occurrence of WTA-related keywords as one feature, while our WTAGRAPH does not explicitly extract this feature. However, we also found that WTAGRAPH is slightly less robust than AdGraph against the evasions of changing second-level and lower-level domains. This is probably because changing domains will lead to graph structure changes (e.g., adding or deleting nodes and edges), thus affecting WTAGRAPH's detection.

We also found that filter lists are the most vulnerable solution. Specifically, 57.78% WTA URLs evaded filter lists successfully by changing second-level domain. This is because a large portion of rules detect WTA requests by matching second-level domain. For example, over one-third of EasyList [8] rules match second-level domain for WTA detection. By changing second-level domain, it is easy to evade those rules such as "`||ad1data.com^`" and "`||adserved.net^$third-party`". Besides, filter lists are also more vulnerable than WTAGRAPH and AdGraph to changing lower-level domains and replacing keywords.

D. Real-time Performance

To evaluate WTAGRAPH's performance in real-time, we implemented an extension-based WTA detection solution by leveraging the native messaging API of Google Chrome [58] which enables the communication between our browser extension and the native WTAGRAPH application. In more details, we first bundled WTAGRAPH and all its dependencies into a standalone executable that can be started by Google Chrome as a native messaging host in a separate process. We then added a messaging component to our data collection extension (introduced in Figure 2) so that it can send and receive messages to and from the WTAGRAPH executable using the native messaging API. Overall, the real-time WTA detection on a given webpage is performed in the following four steps. First, our extension collects HTTP traffic, DOM, and JavaScript API access while visiting a webpage as we introduced in Sesction IV-B. Second, it sends the collected data to the WTAGRAPH executable running (started by Google Chrome) on the local machine using the native messaging API. Third, WTAGRAPH extracts features, builds a testing AHMG from the collected data, and predicts the label of each collected request. Lastly, WTAGRAPH returns its predictions to the extension using the native messaging API.

Based on this implementation, we measured WTAGRAPH's real-time performance on Alexa Top 1K websites. We selected the WTAGRAPH model trained on the Random-5K graph (described in Section VII-A) for real-time evaluation. This model ensures that among the Top 1K websites, half of them have been seen in the Random-5K training graph (i.e., a transductive learning setting), while the other half of them have never been seen in training and can be tested in the inductive learning setting. Specifically, we measured: (1) how much overhead is introduced by WTAGRAPH comparing to a plain Google Chrome browser? (2) how long does WTAGRAPH take to complete the prediction for a webpage? and, (3) how well does WTAGRAPH predict requests?

From Feb. 23rd to Feb. 28th in 2021, we visited the homepages of the Top 1K websites 10 times using a Google Chrome browser with our WTAGRAPH integrated and a plain Google Chrome browser, respectively, for measuring the overhead. Appendix C details the hardware and software configurations for experiments in this subsection. For each visit of a homepage, we recorded the page load time by measuring the time difference between DOM's *navigationStart* and *loadEventEnd* events in each browser, and recorded the overall execution time of WTAGRAPH in our extension by measuring the time difference between sending the first message (i.e., sending the collected data) and receiving prediction results as the last message. We also saved the prediction results for analysis.

From the perspective of overhead, we found that the page load time in the plain Google Chrome browser is 5,903 ms on average, while that in the Google Chrome browser with WTAGRAPH integrated is 6,419 ms on average. In other words, our implementation introduced a 516 ms page load overhead on average. This overhead is mainly caused by the JavaScript API instrumentation and data collection (e.g., polling JavaScript call stack for tracing API access) in our extension, and can be reduced with an improved implementation.

From the perspective of overall execution time, we found that it only takes 266 ms per webpage on average for WTAGRAPH to complete the predictions of the requests on a

webpage. The message exchange between WTAGRAPH and the extension takes 100 ms per webpage on average, and WTAGRAPH spends the rest 166 ms to extract features, build the graph, and predict the labels of all requests on a webpage.

From the perspective of prediction performance, we collected 2,002,026 requests in total from all the 10 crawls of the Top 1K homepages, and summarized WTAGRAPH's prediction performance in Table VIII of Appendix E. Overall, we found that WTAGRAPH can predict all collected requests with an accuracy 92.80%, precision 94.29%, recall 91.29%, and F_1 score 92.77%. We observed a performance decrease comparing to that in Sections VI-A and VII-A. This is mainly because the WTAGRAPH model used for real-time prediction was trained based on the data collected seven months ago. However, this problem can be addressed by periodically retraining WTAGRAPH. The real prediction performance that WTAGRAPH can achieve in inductive learning settings is what we presented in Section VII-A, and that is the prediction performance a retrained model can achieve in real-time.

In summary, WTAGRAPH can be integrated well with a browser to achieve competitive performance in real-time with a small overhead. It would have less overhead if one further improves the integration implementation.

VIII. DISCUSSION

Suggestions for Better Use of WTAGRAPH. As shown in Section VI-B, WTAGRAPH is able to identify new WTA requests that are missed by filter lists and non-WTA requests that are mistakenly blocked by filter lists. Therefore, WTAGRAPH can be used (e.g., in a web privacy crawler) to refine existing filter lists by augmenting new rules to detect new WTA requests and improving existing rules to make fewer mistakes. Besides an extension-based deployment (as we evaluated in Section VII-D), WTAGRAPH can also be potentially deployed as a browser component to support the built-in WTA detection of a browser, which is similar to PageGraph [23] in the Brave browser. However, since a testing AHMG needs to be constructed for multiple requests of each webpage visit to achieve good detection performance, it is not wise to immediately block the requests in a current webpage visit. Instead, the detection results can be used to block the same requests in the follow-up webpage visits. Overall, it is better to use WTAGRAPH in a crawler for refining filter lists.

Limitations and Potential Future Work. Though the evaluation results in Section VI and Section VII have shown that WTAGRAPH is effective in WTA detection, we acknowledge that it has several limitations. For example, it has a false negative rate and false positive rate on the Top-10K dataset at 3.75% and 1.03%, respectively; it has a relatively low accuracy in detecting WTA requests sent to CDN servers (Appendix A). In addition, similar to that in AdGraph [17], we used the outputs from the filter lists as the ground truth labels in the training process, but filter lists themselves will produce false results as shown in our analysis and in [17]; better ground truth labels (e.g., verified by privacy researchers or engineers) could be helpful to WTAGRAPH, AdGraph, and related work.

To address WTAGRAPH's limitations and further improve its performance, we identified the following potential future work. First, new features for both nodes and edges might help reduce misclassifications of WTAGRAPH. For example, certain DOM features (e.g., tag attributes and JavaScript existence) could help WTAGRAPH correctly detect more `sub_frame` WTA requests. One can design more features specific to each type of requests to better characterize WTA requests. Second, a fine-grained aggregation strategy could help improve WTAGRAPH's performance on detecting WTA requests sent to CDN servers. For example, by only aggregating neighbor edges with the same request type, WTAGRAPH might be able to learn more relevant features from neighbors. Third, fine-tuning our WTA-GNN model could help improve its performance. For example, sub-sampling and negative-sampling techniques could be applied in the training process to potentially reduce the false positives and negatives. Fourth, some techniques from advanced or alternative GNN models (e.g., the attention mechanism from GAT [20] and the local neighborhood sampling method from GraphSage [19]) could be explored to help improve WTAGRAPH's performance. Meanwhile, while existing knowledge graph (KG) based formulation and learning algorithms (such as [59]–[61]) are not easily applicable to our WTA detection problem due to the multi-graph definition and edge representation differences, some KG-based techniques may be helpful for future exploration.

In addition, manually-curated filter lists perform poorly for detecting WTA requests on non-English websites [62] because they usually target English websites. Hence, an evaluation of WTAGRAPH on non-English websites may provide information regarding if WTAGRAPH could help address this problem.

IX. CONCLUSION

In this paper, we proposed WTAGRAPH, a web tracking and advertising detection framework based on Graph Neural Networks. We first formulated WTA detection as a task of edge representation learning and classification in AHMG. We then designed and implemented four components in WTAGRAPH (code available at [63]) so that it can perform edge representation learning and WTA detection. The evaluation results showed that WTAGRAPH can effectively detect WTA requests in both transductive and inductive learning settings. For example, it detects WTA requests with 97.90% accuracy, 98.38% precision, 96.25% recall, and 97.30% F_1 score on the Top-10K websites in the transductive learning setting; it detects WTA requests with 97.82% accuracy, 98.00% precision, 96.38% recall, and 97.18% F_1 score in the inductive learning setting. Manual verification results indicated that WTAGRAPH can detect new WTA requests that are missed by filter lists and recognize non-WTA requests that are mistakenly labeled by filter lists. Our ablation analysis, evasion evaluation, and real-time evaluation show that WTAGRAPH can have a competitive performance with flexible deployment options in practice.

ACKNOWLEDGMENTS

We thank reviewers and shepherds for valuable suggestions. This research was supported by the NSF grant OIA-1936968.

REFERENCES

[1] S. Englehardt and A. Narayanan, "Online tracking: A 1-million-site measurement and analysis," in *Proc. of the ACM SIGSAC conference on computer and communications security*, 2016.

[2] A. Lerner, A. K. Simpson, T. Kohno, and F. Roesner, "Internet jones and the raiders of the lost trackers: An archaeological study of web tracking from 1996 to 2016," in *Proc. of the USENIX Security Symposium*, 2016.

[3] V. Toubiana, A. Narayanan, D. Boneh, H. Nissenbaum, and S. Barocas, "Adnostic: Privacy preserving targeted advertising," in *Proc. of the Network and Distributed System Security Symposium (NDSS)*, 2010.

[4] J. R. Mayer and J. C. Mitchell, "Third-Party Web Tracking: Policy and Technology," in *Proc. of the IEEE Symposium on Security and Privacy*, 2012.

[5] Z. Yang and C. Yue, "A comparative measurement study of web tracking on mobile and desktop environments," in *Proc. of the Privacy Enhancing Technologies Symposium (PETS)*, 2020.

[6] "Adblock Plus," 2021. https://adblockplus.org/.

[7] "uBlock Origin," 2021. https://github.com/gorhill/uBlock.

[8] "EasyList," 2021. https://easylist.to/easylist/easylist.txt.

[9] "EasyPrivacy," 2021. https://easylist.to/easylist/easyprivacy.txt.

[10] P. Snyder, A. Vastel, and B. Livshits, "Who filters the filters: Understanding the growth, usefulness and efficiency of crowdsourced ad blocking," in *Proc. of the ACM on Measurement and Analysis of Computing Systems*, 2020.

[11] M. Alrizah, S. Zhu, X. Xing, and G. Wang, "Errors, misunderstandings, and attacks: Analyzing the crowdsourcing process of ad-blocking systems," in *Proc. of the Internet Measurement Conference*, 2019.

[12] D. Gugelmann, M. Happe, B. Ager, and V. Lenders, "An automated approach for complementing ad blockers' blacklists," in *Proc. of the Privacy Enhancing Technologies Symposium (PETS)*, 2015.

[13] A. Shuba, A. Markopoulou, and Z. Shafiq, "Nomoads: Effective and efficient cross-app mobile ad-blocking," in *Proc. of the Privacy Enhancing Technologies Symposium (PETS)*, 2018.

[14] M. Ikram, H. J. Asghar, M. A. Kaafar, A. Mahanti, and B. Krishnamurthy, "Towards seamless tracking-free web: Improved detection of trackers via one-class learning," in *Proc. of the Privacy Enhancing Technologies Symposium (PETS)*, 2017.

[15] Q. Wu, Q. Liu, Y. Zhang, P. Liu, and G. Wen, "A machine learning approach for detecting third-party trackers on the web," in *Proc. of the European Symposium on Research in Computer Security*, 2016.

[16] A. J. Kaizer and M. Gupta, "Towards automatic identification of javascript-oriented machine-based tracking," in *Proc. of the ACM on International Workshop on Security And Privacy Analytics*, 2016.

[17] U. Iqbal, P. Snyder, S. Zhu, B. Livshits, Z. Qian, and Z. Shafiq, "Adgraph: A graph-based approach to ad and tracker blocking," in *Proc. of the IEEE Symposium on Security and Privacy*, 2020.

[18] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *Proc. of the International Conference on Learning Representations*, 2017.

[19] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Proc. of the Advances in Neural Information Processing Systems*, 2017.

[20] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph Attention Networks," in *Proc. of the International Conference on Learning Representations*, 2018.

[21] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?," in *Proc. of the International Conference on Learning Representations*, 2018.

[22] S. Bhagavatula, C. Dunn, C. Kanich, M. Gupta, and B. Ziebart, "Leveraging machine learning to improve unwanted resource filtering," in *Proc. of the Artificial Intelligent and Security Workshop*, 2014.

[23] "PageGraph in Brave Browser," 2021. https://github.com/brave/brave-browser/wiki/PageGraph.

[24] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *Proc. of the International Conference on Machine Learning*, 2017.

[25] W.-L. Chiang, X. Liu, S. Si, Y. Li, S. Bengio, and C.-J. Hsieh, "Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks," in *Proc. of the ACM International Conference on Knowledge Discovery & Data Mining*, 2019.

[26] Y. LeCun, Y. Bengio, *et al.*, "Convolutional networks for images, speech, and time series," *The handbook of brain theory and neural networks*, 1995.

[27] L. Landrieu and M. Simonovsky, "Large-scale point cloud semantic segmentation with superpoint graphs," in *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[28] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon, "Dynamic graph cnn for learning on point clouds," *ACM Transactions On Graphics*, 2019.

[29] S. Yan, Y. Xiong, and D. Lin, "Spatial temporal graph convolutional networks for skeleton-based action recognition," in *Proc. of the AAAI conference on artificial intelligence*, 2018.

[30] H. Wang, T. Xu, Q. Liu, D. Lian, E. Chen, D. Du, H. Wu, and W. Su, "Mcne: An end-to-end framework for learning multiple conditional network representations of social network," in *Proc. of the ACM International Conference on Knowledge Discovery & Data Mining*, 2019.

[31] W. Fan, Y. Ma, Q. Li, Y. He, E. Zhao, J. Tang, and D. Yin, "Graph neural networks for social recommendation," in *Proc. of the World Wide Web Conference*, 2019.

[32] T. Bian, X. Xiao, T. Xu, P. Zhao, W. Huang, Y. Rong, and J. Huang, "Rumor detection on social media with bi-directional graph convolutional networks," in *Proc. of the AAAI Conference on Artificial Intelligence*, 2020.

[33] D. K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams, "Convolutional networks on graphs for learning molecular fingerprints," in *Proc. of the Advances in Neural Information Processing Systems*, 2015.

[34] Z. Xiong, D. Wang, X. Liu, F. Zhong, X. Wan, X. Li, Z. Li, X. Luo, K. Chen, H. Jiang, *et al.*, "Pushing the boundaries of molecular representation for drug discovery with the graph attention mechanism," *Journal of Medicinal Chemistry*, 2019.

[35] "Fully Qualified Domain Name," 2021. https://en.wikipedia.org/wiki/Fully_qualified_domain_name.

[36] Z. Li, K. Zhang, Y. Xie, F. Yu, and X. Wang, "Knowing your enemy: understanding and detecting malicious web advertising," in *Proc. of the ACM conference on Computer and communications security*, 2012.

[37] B. Li, P. Vadrevu, K. H. Lee, R. Perdisci, J. Liu, B. Rahbarinia, K. Li, and M. Antonakakis, "Jsgraph: Enabling reconstruction of web attacks via efficient tracking of live in-browser javascript executions.," in *Proc. of the Network and Distributed System Security Symposium (NDSS)*, 2018.

[38] J. Jueckstock and A. Kapravelos, "Visiblev8: In-browser monitoring of javascript in the wild," in *Proc. of the Internet Measurement Conference*, 2019.

[39] "Selenium Web Driver," 2021. http://www.seleniumhq.org/.

[40] R. Jozefowicz, O. Vinyals, M. Schuster, N. Shazeer, and Y. Wu, "Exploring the limits of language modeling," *arXiv preprint arXiv:1602.02410*, 2016.

[41] C. Chelba, T. Mikolov, M. Schuster, Q. Ge, T. Brants, P. Koehn, and T. Robinson, "One billion word benchmark for measuring progress in statistical language modeling," in *Proc. of the Annual Conference of the International Speech Communication Association*, 2014.

[42] P. Laperdrix, W. Rudametkin, and B. Baudry, "Beauty and the beast: Diverting modern web browsers to build unique browser fingerprints," in *Proc. of the IEEE Symposium on Security and Privacy*, 2016.

[43] "chrome.webRequest API," 2021. https://developer.chrome.com/docs/extensions/reference/webRequest/.

[44] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *Proc. of the ACM International Conference on Knowledge Discovery & Data Mining*, 2016.

[45] C. Zhang, D. Song, C. Huang, A. Swami, and N. V. Chawla, "Heterogeneous graph neural network," in *Proc. of the ACM International Conference on Knowledge Discovery & Data Mining*, 2019.

[46] "Deep Graph Library," 2021. https://github.com/dmlc/dgl.

[47] "Instrumented Chromium Broswer in AdGraph," 2021. https://github.com/uiowa-irl/AdGraph.

[48] "Anti-Adblock Killer ," 2021. https://github.com/reek/anti-adblock-killer.

[49] "Warning Removal List," 2021. https://easylist-downloads.adblockplus.org/antiadblockfilters.txt.

[50] "Blockzilla," 2021. https://zpacman.github.io/Blockzilla/.

[51] "Peter Lowes's List," 2021. https://pgl.yoyo.org/adservers/.

[52] "Fanboy Annoyances List," 2021. https://easylist-downloads.adblockplus.org/fanboy-annoyance.txt.

[53] "Request Types Defined in Chrome," 2021. https://developer.chrome.com/extensions/webRequest#type-ResourceType.

[54] "Ad Network Uses DGA Algorithm to Bypass Ad Blockers and Deploy In-Browser Miners," 2021. https://www.bleepingcomputer.com/news/security/ad-network-uses-dga-algorithm-to-bypass-ad-blockers-and-deploy-in-browser-miners/.
[55] H. Le, F. Fallace, and P. Barlet-Ros, "Towards accurate detection of obfuscated web tracking," in *Proc. of the IEEE International Workshop on Measurement and Networking*, 2017.
[56] S. Liu, I. Ounis, C. Macdonald, and Z. Meng, "A heterogeneous graph neural model for cold-start recommendation," in *Proc. of the ACM SIGIR Conference on Research and Development in Information Retrieval*, 2020.
[57] B. Hao, J. Zhang, H. Yin, C. Li, and H. Chen, "Pre-training graph neural networks for cold-start users and items representation," in *Proc. of the ACM International Conference on Web Search and Data Mining*, 2021.
[58] "Native Messaging," 2021. https://developer.chrome.com/docs/apps/nativeMessaging/.
[59] A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston, and O. Yakhnenko, "Translating embeddings for modeling multi-relational data," *Proc. of the Advances in Neural Information Processing Systems*, 2013.
[60] Z. Wang, J. Zhang, J. Feng, and Z. Chen, "Knowledge graph embedding by translating on hyperplanes," in *Proc. of the AAAI Conference on Artificial Intelligence*, 2014.
[61] Y. Lin, Z. Liu, M. Sun, Y. Liu, and X. Zhu, "Learning entity and relation embeddings for knowledge graph completion," in *Proc. of the AAAI conference on artificial intelligence*, 2015.
[62] A. Sjösten, P. Snyder, A. Pastor, P. Papadopoulos, and B. Livshits, "Filter list generation for underserved regions," in *Proc. of The Web Conference*, 2020.
[63] "WtaGraph source code and dataset," 2021. https://github.com/jun521ju/IEEE_SP_2022_WtaGraph.
[64] "DNS Prefetching in Chromium," 2021. https://www.chromium.org/developers/design-documents/dns-prefetching.

APPENDIX

A. Manual Verification

Procedure and Criteria. To get a better understanding of the disagreements between WTAGRAPH and filter lists on the Top-10K dataset in the transductive learning setting (Section VI-A), we followed the criteria presented in [17] to manually verify a sampled set of false positives (i.e., WTAGRAPH classified them as WTA but filter lists labeled them as non-WTA) and false negatives (i.e., WTAGRAPH classified them as non-WTA but filter lists labeled them as WTA). In more details, we first randomly sampled 100 false positives and 100 false negatives of each request type in Table III. If there are less than 100 false positives or false negatives of a request type, we selected all available ones. Then, we manually verified these sampled requests and assigned each of them to one of four labels: (1) WTA, i.e., the request is confirmed to be related to tracking and/or advertising, (2) non-WTA, i.e., the request is confirmed to be not related to tracking or advertising, (3) mixed, i.e., the request (only used for script resources as in [17]) can be both WTA and non-WTA related, (4) undecidable, i.e., the request's label cannot be determined during the manual verification. Similarly, to verify and confirm the agreements (i.e., true positives and true negatives) between WTAGRAPH and filter lists, we conducted another set of manual verifications by following the same sampling procedure and labeling criteria.

In more details, the criteria for determining the label of a request during our manual verification are as follows. As in [17], an `image` request will be labeled as WTA if the image is a tracking pixel (e.g., $1*1$ sized image) or the image has content related to advertising (e.g., having keywords such as sponsored and price); a `script` request will be labeled as WTA if the JavaScript code is used for device fingerprinting, cookie or beacon transmission, ad-related DOM modification, or WTA service communication. We further propose the following criteria for the rest request types that were not sampled in [17]. An `xmlhttprequest`, `ping`, `font`, `stylesheet`, or `csp_report` request will be labeled as WTA if its request body or header carries obvious identifier data (such as user ID and device ID in a cookie or query section). A `sub_frame`, `main_frame`, or `media` request will be labeled as WTA if it has content related to advertising (e.g., an advertisement image or video). In addition, a `sub_frame` or `main_frame` request will also be labeled as WTA if the requested frame embeds with WTA-related JavaScript. An `other` request is usually of one of the above request types; hence it will be labeled by applying the same criteria of the aforementioned request types. For each specific request, it will be labeled as non-WTA if none of the above criteria of its request type applies to it, or will be labeled as undecidable if it cannot be determined during manual verification (e.g., the requested resource is not available at the time of verification). Note that a `script` request could also be labeled as mixed if its JavaScript code serves for both website functional and WTA purposes [17].

Verification on Disagreements. Following the above criteria, in total we randomly sampled and manually verified 846 false positives (accounting for 8.67% of all cases) and 941 false negatives (accounting for 4.09% of all cases). Table VII(a) details our manual verification results of these sampled disagreements between WTAGRAPH and filter lists of each request type. Overall, WTAGRAPH's classification is indeed correct for 60.36% and 37.83% of the sampled false negatives and false positives, respectively. This result implies that the actual performance of WTAGRAPH could be higher than what was reported in Table III (Section VI-A).

Among 846 sampled false positives, we found that WTAGRAPH is able to identify WTA requests that are missed by filter lists for types of `image`, `script`, `xmlhttprequest`, `sub_frame`, `ping`, and `media`. For example, among the 100 `ping` and 100 `media` requests that are labeled as non-WTA by filter lists, 69 and 64 of them are verified as WTA-related requests, respectively.

Meanwhile, we found that there are 444 requests that are verified as non-WTA but classified incorrectly as WTA by WTAGRAPH. There are two potential reasons for WTAGRAPH's misclassification. First, WTAGRAPH's neighborhood aggregation in the learning process might not be sufficiently fine-grained. For example, many of those requests (such as `stylesheet`, `font`, and `script` requests) were sent to CDN servers, which could host both WTA and non-WTA resources; as a consequence, nodes that represent CDN domains in the AHMG would have both WTA and non-WTA edges, and hence resulting in the ambiguous or misleading aggregation from neighbor edges. Second, WTAGRAPH was likely misled by some URL characteristics. For example, many `xmlhttprequest` requests carried multiple parameters in

Table VII: Manual verification results of sampled disagreements and agreements between WTAGRAPH and filter lists.

(a) Verification results of sampled disagreements (i.e., false positives and false negatives).

Request Type	False Positives					False Negatives				
	# Samples	WTA	Non-WTA	Mixed	Undecidable	# Samples	WTA	Non-WTA	Mixed	Undecidable
image	100	52	37	-	11	100	31	53	-	16
script	100	41	48	4	7	100	47	41	3	9
xmlhttprequest	100	37	49	-	14	100	57	29	-	14
other	50	10	36	-	4	100	7	91	-	2
stylesheet	100	0	94	-	6	100	0	94	-	6
sub_frame	100	44	46	-	10	100	65	32	-	3
font	28	0	26	-	2	100	0	100	-	0
main_frame	69	3	62	-	4	100	0	100	-	0
ping	100	69	12	-	19	100	84	8	-	8
media	92	64	27	-	1	32	19	11	-	2
csp_report	7	0	7	-	0	9	0	9	-	0
object	0	0	0	-	0	0	0	0	-	0
Total	846	320	444	4	78	941	310	568	3	60

(b) Verification results of sampled agreements (i.e., true positives and true negatives).

Request Type	True Positives					True Negatives				
	# Samples	WTA	Non-WTA	Mixed	Undecidable	# Samples	WTA	Non-WTA	Mixed	Undecidable
image	100	96	0	-	4	100	1	97	-	2
script	100	98	0	0	2	100	0	90	0	10
xmlhttprequest	100	92	0	-	8	100	4	88	-	8
other	100	100	0	-	0	100	0	96	-	4
stylesheet	100	100	0	-	0	100	0	93	-	7
sub_frame	100	99	1	-	0	100	5	94	-	1
font	100	100	0	-	0	100	0	100	-	0
main_frame	61	14	37	-	10	100	0	100	-	0
ping	100	99	0	-	1	100	31	65	-	4
media	100	93	0	-	7	100	0	94	-	6
csp_report	79	7	31	-	41	100	0	100	-	0
object	0	0	0	-	0	39	0	39	-	0
Total	1,040	898	69	0	73	1,139	41	1,056	0	42

the URL query string for measuring website performance; such a URL characteristic is similar to that of WTA-related `xmlhttprequest` requests which usually carry lots of WTA-related data in the URL query string.

We further sampled 55 (five from each type) out of these 444 requests to evaluate the impact of blocking them. Specifically, for each webpage that issued a sampled request, we visited it five times under two conditions (i.e., with and without the sampled request being blocked). We found that blocking the 55 requests leads to (1) no perceptible difference on 46 webpages, (2) a blank element (e.g., a missing image of a `div`) on five webpages, and (3) complete breakage on four webpages. For example, blocking `https://scout.salesloft.com/i` on `newrelic.com` does not incur any perceptible difference; blocking `https://www.google.com/maps/embed?pb=!...` on `mobinsb.com` results in the missing of an embedded Google Map widget on the webpage; blocking `https://www.zergnet.com/ajax/load_results...` on `zergnet.com` breaks the webpage. The blank elements on five webpages are due to the blocking of two `image` requests, one `xmlhttprequest` request, one `sub_frame` request, and one `media` request. The breakage on four webpages is due to the blocking of three `main_frame` requests and one `xmlhttprequest` request.

Among 941 sampled false negatives, we verified that WTAGRAPH's classification is correct over 90% of the time for `stylesheet`, `main_frame`, `other`, and `font` requests. That is, filter lists mistakenly labeled those requests as WTA. We looked into why filter lists would make such mistakes, and found that some coarse-grained rules in filter lists would aggressively or mistakenly block some resources. For example, Peter Lowes's List [51] aggressively blocks any request to *criteo.com*. As a consequence, when one visits *criteo.com* as a first-party website, all requests to *criteo.com*, including the `main_frame` request, will be blocked by this filter list. We also observed that among the sampled `xmlhttprequest`, `ping`, and `sub_frame` requests, WTAGRAPH tends to mistakenly classify verified WTA requests as non-WTA. Our investigation results indicate that WTAGRAPH missed the opportunity to utilize some important though covert features for classification. For example, while many `sub_frame` resources embed with WTA JavaScript code, the current version of WTAGRAPH did not use JavaScript code features for the corresponding `sub_frame` request classification.

Verification on Agreements. Table VII(b) details manual verification results of sampled 1,040 true positives and 1,139 true negatives, which are the agreements between WTAGRAPH and filter lists. Overall, the majority (93.23%) of agreements between them are verified as correct.

Among 1,040 sampled true positives, we verified that 86.35% (i.e., 898) of these agreements between WTAGRAPH and filter lists are correct as they are indeed WTA requests. Specifically, on the top five dominant request types (i.e., `image`, `script`, `xmlhttprequest`, `other`, and `stylesheet`), WTAGRAPH's classifications are completely correct and consistent with filter lists. However, we observed that the agreements on 37 `main_frame`, 31 `csp_report`, and 1 `sub_frame` requests are questionable according to our labeling criteria. For example, EasyPrivacy [9] labeled these 31 `csp_report` requests as WTA because there were

sent to report-uri.io and report-uri.com; though WTAGRAPH's classification is consistent with EasyPrivacy [9] on these requests, it is questionable to label and predict every request sent to a filter-lists-included domain as a WTA request because of the imperfection of filter lists (recall the aforementioned example of visiting *criteo.com* as a first-party website).

Among 1,139 sampled true negatives, we verified that 92.71% (i.e., 1,056) of these agreements between WTAGRAPH and filter lists are correct as they are indeed non-WTA requests. However, our manual verification also shows that 41 agreed non-WTA requests are indeed WTA requests. That is, both WTAGRAPH and filter lists missed the opportunity to detect these WTA requests.

Summary. These manual verification results have the following major implications. From the perspective of WTAGRAPH, first, the sampled false positives and false negatives indicate that WTAGRAPH could outperform traditional filter lists in terms of detecting new WTA requests and recognizing non-WTA requests that were mistakenly labeled by filter lists. Second, its performance could be improved by refining some design of its components. For example, a fine-grained same-type aggregation strategy (i.e., only aggregating from neighbor edges with the same request type) would be better for learning representations of those requests sent to CDN servers; by designing more features for `ping` requests and incorporating JavaScript code features for `sub_frame` requests, WTAGRAPH may potentially classify those requests more accurately. We leave these improvements as future work. Third, because of the imperfection and incompleteness of filter lists (e.g., missing WTA requests and over-blocking non-WTA requests), the correspondingly trained WTAGRAPH could make mistakes as a filter list does as indicated by some sampled true positives and true negatives. We believe that fine-grained filter lists could help address this problem and improve WTAGRAPH's performance.

From the perspective of filter lists, they inevitably lag behind in the arms race of WTA detection because of the temporal gap between a new WTA request being created on the web and a manually-curated rule being updated to filter lists. Besides, some filter lists should be refined so that they would not aggressively or mistakenly block non-WTA requests especially for first-party websites.

B. Dataset Discrepancy Between WTAGRAPH and AdGraph

We investigated the discrepancy (described in Section V) between data collected by WTAGRAPH and by AdGraph. The results of the manual inspection on 20 sampled websites indicate that there are two reasons for the request miss by WTAGRAPH. First, AdGraph considers all URLs appearing in the `href` attribute of `link` HTML tags as HTTP requests. However, the fact is that a browser will not issue HTTP requests for some of those URLs in the `href` attribute. For example, `<link rel="dns-prefetch" href="//example.com">` will not trigger Chromium to issue an HTTP request to `example.com`, because "Chromium's implementation of DNS prefetching does not use the browser's network stack at all" [64]. Second, AdGraph considers all values in the `src` attribute of `img` HTML tags as URLs. However, the `src` attribute accepts the data URI (e.g., "...") of an image, which will not issue an HTTP request. In summary, WTAGRAPH collects data correctly as what it missed but extracted by AdGraph are not HTTP requests in practice. This is confirmed by the complete match between requests collected by WTAGRAPH and requests displayed on the Network tab of the browser's DevTools.

As for the reason for AdGraph missing a large number of requests that are collected by WTAGRAPH and displayed on the browser's Network tab, we conjecture that AdGraph's Chromium browser instrumentation might not be complete. For example, we found that `ping`, `stylesheet` and `other` types of requests were never recorded by AdGraph's Chromium browser.

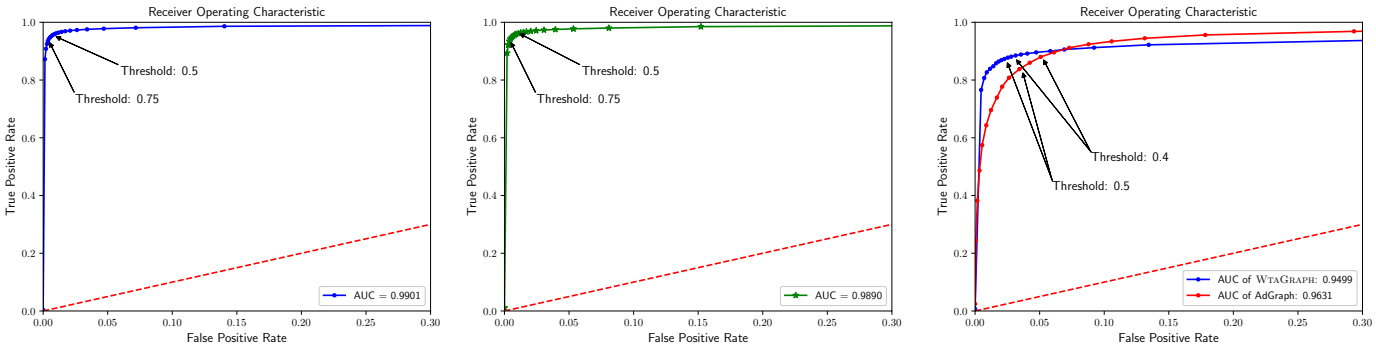
C. Details on the Experimental Setup

Hyperparameters. For all the WTAGRAPH models and variants presented in Sections VI and VII and three baseline models presented in Section VI-A, we trained them for 1000 epochs (enough for stabilizing their performance) with a learning rate of 0.005. We used ReLU as the non-linear activation function and Adam optimizer as the optimization algorithm. All these models have two layers, where the dimension of the hidden layer is 32 and that of the output layer is 2. For the random forest model of AdGraph presented in Section VII-B, we trained it in Weka using the same configuration as described in [17]. Specifically, we configured the random forest with 100 decision trees. Each decision tree is trained using $int(logM + 1)$ features, where M is the total number of features.

We conducted experiments on evaluating the hyperparameter sensitivity of our WTAGRAPH model (Table II) in the transductive learning setting. Specifically, we set different learning rates (i.e., 0.001, 0.005, and 0.0025), training epochs (i.e., 1,000 and 2,000), and dimensions of the hidden layer (i.e., 32, 64, and 128). Overall, we found that our model is not sensitive to these hyperparameters. For example, the accuracy is increased by 0.25% after changing the learning rate from 0.005 to 0.001, and is increased by 0.11% after changing the dimension of the hidden layer from 32 to 64. We finalized the aforementioned hyperparameters by considering the trade-off between training efficiency and performance.

Hardware and Software. For the real-time evaluation in Section VII-D, we ran experiments on a typical desktop machine with 8 Intel i7-3770 CPU cores and 16GB of RAM. This machine is installed with 64-bit Ubuntu 18.04 and a Google Chrome browser (version 88.0.4324.96).

For training and testing all the models (except for the real-time evaluation in Section VII-D), we ran experiments on a server with 40 Intel Xeon CPU cores (E5-2698 v4 @ 2.20GHz), 4 Nvidia Tesla V100 GPUs, and 256GB of RAM. Using this machine, it takes around 0.8 second per epoch and around 15 minutes in total to complete the training process of WTAGRAPH. Besides, WTAGRAPH is implemented with Python 3.6.9, Pytorch 1.5.0, and DGL 0.5.3.



(a) WTAGRAPH: transductive (b) WTAGRAPH: inductive (c) WTAGRAPH and AdGraph: inductive

Fig. 5: ROC curves and AUC values in transductive and inductive learning settings.

D. ROC and AUC Analysis

In both Section VI and Section VII, we reported WTAGRAPH's performance with a default (and commonly chosen) probability threshold of 0.5 in different transductive and inductive learning settings. That is, WTAGRAPH predicts a request as WTA if the probability for this request to be WTA is greater than 50%. Using this default probability threshold, the false positive rate of WTAGRAPH in the transductive learning setting is 1.03% as reported in Table III. We now analyze the impact of different probability thresholds in both transductive and inductive learning settings. Specifically, we first set the threshold from 0 to 1 with a step of 0.05 and derive each model's performance using every threshold. We then derive ROC (Receiver Operating Characteristic) curves and AUC (Area under the ROC Curve) values to further interpret the performance of those models.

We derived the ROC curves for four models (i.e., WTAGRAPH and three baseline models) presented in Section VI-A in transductive learning settings. We found that the AUCs for the link prediction GNN, MLP, advanced link prediction GNN, and WTAGRAPH are 0.9763, 0.9897, 0.9900, and 0.9901, respectively. Similarly, in inductive learning settings, the AUCs for the WTAGRAPH models (Section VII-A) trained on the Random-1K, Random-3K, Random-5K, and Random-7K graphs are 0.9558, 0.9796, 0.9851, and 0.9890, respectively.

Figure 5(a) plots the ROC curve of WTAGRAPH in the transductive learning setting, while Figure 5(b) plots the ROC curve of the WTAGRAPH model trained on the Random-7K graph in the inductive learning setting. Note that because the ROC curves align very close to each other, we did not plot the curves for other models in Figure 5(a) and 5(b). We observed that by using the threshold of 0.5, WTAGRAPH has a TPR of 96.25% and FPR of 1.03% in the transductive learning setting, while it has a TPR of 96.38% and FPR of 1.25% in the inductive learning setting. By increasing the threshold to 0.75, WTAGRAPH has a TPR of 94.23% and FPR of 0.44% in the transductive learning setting, while it has a TPR of 94.82% and FPR of 0.56% in the inductive learning setting. This implies that one can increase the threshold value to achieve a lower FPR along with a relatively good TPR in practice.

Figure 5(c) presents the ROC curves of WTAGRAPH and AdGraph in the inductive learning setting evaluated on the overlapped dataset (Sectin VII-B). While the AUC of AdGraph is slightly higher than that of WTAGRAPH, we should note that there are always some caveats when interpreting AUCs. Especially, we should care more about the left portions of the curves (which correspond to higher threshold values) if avoiding a high false positive rate is critical in practice as in WTA detection. We found that at each threshold starting from 0.4, WTAGRAPH always achieves both a higher TPR and a lower FPR than AdGraph. For example, at the threshold of 0.4, WTAGRAPH has a TPR of 88.50% and FPR of 3.14%, while AdGraph has a TPR of 88.01% and FPR of 5.08%; at the threshold of 0.5, WTAGRAPH has a TPR of 87.76% and FPR of 2.54%, while AdGraph has a TPR of 83.77% and FPR of 3.43%. In WTA detection, the probability or confidence level is often expected to be high (e.g., 50% or even above) to avoid misclassifying and blocking requests.

E. Detailed Real-time Prediction Performance

In Section VII-D, we evaluated WTAGRAPH's performance in real-time. From the perspective of prediction performance, Table VIII summarizes the detailed results.

Table VIII: The real-time prediction performance of WTAGRAPH. The last two rows provide a detailed view in two cases: 500 websites that have been seen during training and 500 websites that have never been seen during training.

	# Sites	# Requests	Acc.	Prec.	Recall	F_1
Overall	1,000	2,002,026	92.80%	94.29%	91.29%	92.77%
Case: Seen	500	998,534	93.68%	95.56%	91.96%	93.73%
Case: Unseen	500	1,003,492	91.92%	93.02%	90.59%	91.79%

We noted in Section VII-D that this prediction performance is lower than what was reported in Section VI-A and Section VII-A mainly because the WTAGRAPH model used for real-time prediction was trained based on the data collected seven months ago. We compared the requests collected on the Top 1K websites in our Top-10K dataset and the requests collected in real-time from one crawl, and found that 85.65% requests and 25.10% FQDNs are different between these two datasets collected seven months apart. In other words, due to the webpage and network traffic pattern changes in the past seven months, the WTAGRAPH model used in this experiment mainly for runtime performance evaluation is relatively obsolete to achieve strong prediction performance.