

PHISHXGRAPH: Robust Phishing Website Detection using Cross-Layer Interaction Graphs

Jaeho Bae¹, Junghoon Kim¹, Doowon Kim², Seongil Wi¹

¹ UNIST

² University of Tennessee, Knoxville

Abstract. Phishing remains a persistent and evolving threat to web security, continuously adapting to circumvent traditional defenses. Existing phishing detection methodologies often rely on features extracted from only a limited set of surface-level resources, such as URL strings, HTML content, or visual logos. This fragmented view limits their ability to generalize across diverse phishing webpages and leaves them vulnerable to resource-specific adversarial attacks.

In this paper, we propose PHISHXGRAPH, a robust phishing detection framework that learns the underlying functional logic of phishing webpages through cross-layer interaction graphs. It models the relationships among HTML structure, JavaScript execution, network requests, and browser storage as a heterogeneous directed graph. It then classifies each webpage using a machine learning model trained on graph-derived features. On a large-scale dataset with 106K webpages, PHISHXGRAPH achieves an accuracy of 98.64% and an F1 score of 97.28%, significantly outperforming five state-of-the-art phishing detectors. Under diverse adversarial attacks, it achieves the lowest attack success rate, averaging 2.57% compared with 21.58% for the five baselines, which demonstrates its strong robustness.

Keywords: Phishing detection · Web security · Graph-based analysis

1 Introduction

Phishing remains one of the most pervasive and impactful cyber threats in the modern web ecosystem. By masquerading as legitimate entities, attackers deceive users into surrendering sensitive credentials, leading to significant financial and personal damages that continue to escalate in 2025 [7, 20]. To counter these threats, early defenses relied heavily on blacklists [1, 5]; however, these methods often fail to protect users during the critical early hours of a phishing page’s lifecycle due to rapid URL rotation [27, 38, 39, 44].

To address this limitation, numerous machine learning (ML)-based phishing detection techniques have been proposed [15, 24, 35, 42, 48, 52]. These methods aim to extract discriminative features from phishing webpages. For example, prior work has considered features from lexical URL patterns [24, 42, 48], HTML content [30, 50, 52], and network infrastructure [18, 21]. More recently, reference-based approaches have also been explored [10, 29, 32–34]. In particular, these

methods compare suspect pages against brand-specific resources from legitimate websites, such as logos.

However, existing approaches still share a fundamental limitation: they observe only a partial surface of a webpage. As shown in Table 1, prior approaches typically focus on only one or a few resource types, such as URLs, HTML, network infrastructure, or brand logos. This design (1) limits generalizability across diverse phishing webpages and (2) remains vulnerable to resource-specific adversarial attacks (e.g., image perturbation [17, 19, 40]).

Contributions. In this paper, we propose PHISHXGRAPH, a robust phishing detection framework based on cross-layer interaction graphs. Our key idea is to detect the underlying invariant phishing patterns (e.g., credential harvesting and exfiltration) by considering not only a broad set of resource types, but also the interactions among them.

During the rendering process, PHISHXGRAPH constructs a heterogeneous directed graph, called a cross-layer interaction graph, that captures interactions across four key layers: HTML, JavaScript (JS), network requests, and browser storage. From this graph, PHISHXGRAPH extracts content, structural, and behavioral features, which are then used by a machine learning model to classify webpages as phishing or benign.

We compared PHISHXGRAPH against five state-of-the-art baselines on a large-scale dataset of 106K webpages, including 53K legitimate pages and 53K phishing pages. PHISHXGRAPH achieved an F1 score of 97.28%, outperforming URLNet (92.74%), LBP (84.10%), StackModel (89.68%), Phishpedia (75.56%), and PhishIntention (74.27%). Our feature analysis further shows that cross-resource interactions contribute substantially to detection, highlighting the importance of cross-layer interaction graphs for phishing detection.

Under adversarial attacks, PHISHXGRAPH achieves the lowest attack success rate, averaging 2.57% compared with 21.58% for the five baselines. We further demonstrate the practical utility of PHISHXGRAPH in a 15-day real-world study, where it proactively identified 1,716 emerging phishing webpages before they appeared on major blacklists, including OpenPhish [8], PhishTank [1], and Google Safe Browsing (GSB) [5].

Our key contributions are summarized as follows:

1. We propose PHISHXGRAPH, a phishing detection framework that models HTML, JavaScript, network, and browser storage interactions as a cross-layer interaction graph.
2. We demonstrate the superior performance of PHISHXGRAPH over state-of-the-art phishing detection tools on 106K webpages.
3. We show that PHISHXGRAPH is robust against adversarial attacks and effective at identifying emerging phishing campaigns in the wild.
4. To support open science and reproducible research, we release PHISHXGRAPH at <https://github.com/WebSec-Lab/PhishXGraph>.

2 Background and Related Work

We describe existing phishing detection approaches in three categories: blacklist-based approaches, machine learning (ML)-based approaches, and reference-based approaches.

Blacklist-based approaches. Blacklist-based approaches maintain lists of known phishing URLs and block access when a visited URL matches an entry [1, 5]. However, such approaches suffer from an inherent limitation: they miss previously unseen phishing pages and are easily evaded through URL rotation [44]. Oest *et al.* [38] further showed that delays in blacklist updates leave users unprotected during the early lifetime of a phishing page.

ML-based approaches. Numerous ML-based techniques have been proposed to identify unknown phishing websites [24, 30, 35, 42, 48, 50, 52]. CANTINA [52] and CANTINA+ [50] incorporate TF-IDF content analysis, search engine validation, and HTML/DOM indicators. Ma *et al.* [35] and Sahingoz *et al.* [42] used lexical URL and host-based features. URLNet [24] learns character- and word-level representations from URL strings using two CNN branches. Texception [48] models URL strings with multi-scale convolutions over character and word embeddings. StackModel [30] combines lexical URL features and HTML structure features (tag-length statistics, hidden-element flags, link structure) into a stacking ensemble.

Another recent line of research focuses on identifying phishing websites by leveraging graph-level features [12, 16, 22, 36, 46, 49]. PhishGNN [12] models a website as a hyperlink graph and combines it with URL features. BPE [22] and LBP [16] propagate phishing labels over a heterogeneous network graph via belief propagation, making detection depend on the hosting infrastructure. GraPhish [36] instead constructs graphs from encrypted TLS traffic features to detect phishing from network-level flows. HUGPhish [49] represents HTML content as graphs based on HTML tags, extracts graph embeddings with a GNN, and combines them with URL and handcrafted HTML features. SpecularNet [46] models the DOM as a tree together with the domain name and applies a hierarchical graph autoencoder.

Reference-based approaches. There has been a surge of research on identifying phishing websites using reference data from legitimate brands [10, 29, 32–34]. In general, these approaches compare a suspect page against a reference database of brand resources collected from known legitimate websites. PhishZoo [10] uses Scale-Invariant Feature Transform (SIFT) descriptors to match the visual appearance of a suspect page’s logo against reference logo profiles of legitimate brands. Phishpedia [32] combines R-CNN logo localization with Siamese-network brand matching. PhishIntention [33] extends Phishpedia by additionally incorporating a credential-request-page (CRP) classifier. DynaPhish [34] and KnowPhish [29] extend this pipeline with dynamic knowledge expansion and large-scale brand knowledge bases.

Table 1 summarizes the webpage resources considered by previous approaches. Because these approaches detect phishing webpages by observing only a subset of webpage resources, their detection capability has limited generalizability across

Table 1: Surface coverage of phishing detection approaches. A ✓ indicates that the method extracts features from the corresponding resource, while ▲ indicates partial observation of that resource.

Category	Research Work	Webpage Resources						Cross-resource Interactions
		URL	HTML	JS	Storage	Network	Image	
ML-based Approaches	CANTINA [52]	✓	✓	—	—	—	—	—
	CANTINA+ [50]	✓	✓	—	—	—	—	—
	Ma <i>et al.</i> [35]	✓	—	—	—	—	—	—
	Sahingoz <i>et al.</i> [42]	✓	—	—	—	—	—	—
	URLNet [24]	✓	—	—	—	—	—	—
	Texception [48]	✓	—	—	—	—	—	—
	StackModel [30]	✓	✓	—	—	▲ [‡]	—	—
	PhishGNN [12]	✓	✓	—	—	—	—	—
	BPE [22]	✓	—	—	—	✓	—	—
	LBP [16]	✓	—	—	—	✓	—	—
	GraPhish [36]	—	—	—	—	✓	—	—
	HUGPhish [49]	✓	✓	—	—	—	—	—
	SpecularNet [46]	✓	✓	—	—	—	—	—
Reference-based Approaches	PhishZoo [10]	✓	✓	—	—	—	✓	—
	Phishpedia [32]	✓	—	—	—	—	✓	—
	PhishIntention [33]	✓	✓	—	—	—	✓	—
	DynaPhish [34]	✓	✓	—	—	—	✓	—
	KnowPhish [29]	✓	✓	—	—	—	✓	—
Cross-layer Graph (Ours)	PHISHXGRAPH	✓	✓	✓	✓	✓	▲ [†]	✓

[†] We analyze only the request associated with an image, not the image content itself.

[‡] It considers only hyperlinks statically present in the HTML.

diverse phishing webpages and remains vulnerable to resource-specific adversarial attacks. We discuss these limitations in detail in §3.

3 Motivation

Motivating example. Figure 1a shows an example of a phishing webpage (metube[.]com) consisting of `index.html` and `app.js`. The page contains a login form on Lines 2–5 in `index.html`. When a victim enters a password into the input field on Line 3 and clicks the ‘Sign in’ button on Line 4, the `onsubmit` JS event handler (Lines 1–6 in `app.js`) executes. The handler concatenates the victim’s password with a value retrieved from `localStorage`³ (Line 5) and sends the combined value to the attacker’s web server via `fetch` (Line 3).

As shown in Table 1, previous approaches detect phishing webpages by observing only a subset of webpage resources. This design introduces two key limitations: (1) lack of generalizability and (2) lack of robustness.

Limitation #1: lack of generalizability. Existing approaches ignore phishing patterns manifested in resources outside their observation scope, and therefore have limited generalizability across diverse phishing webpages (§5.3). For instance, PhishIntention [33] observes URL information (e.g., metube[.]com), HTML information (e.g., the login form and the `password` input field in Lines 2–5 of

³ Phishing pages often use browser storage for fingerprinting-based cloaking [43, 51] and multi-stage phishing orchestration [47, 53].

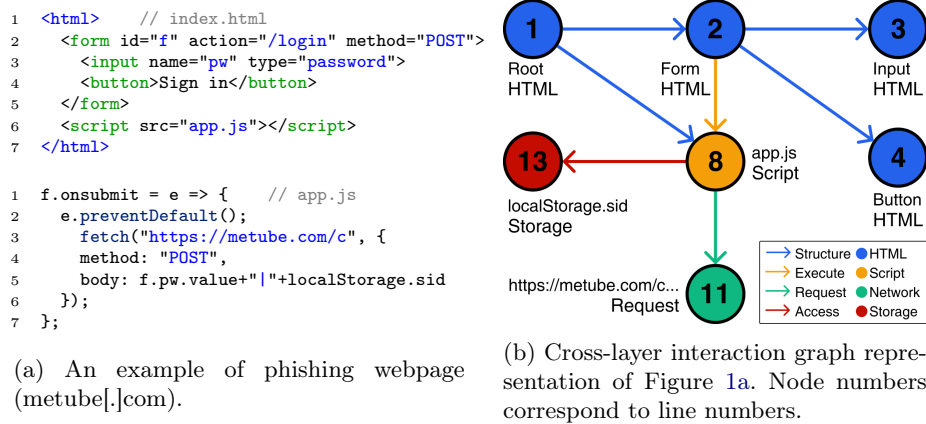


Fig. 1: Motivating example.

index.html), and image information (e.g., brand-like visual elements shown on the page). However, it does not observe the underlying phishing behavior expressed through JS execution (e.g., the `onsubmit` handler in Line 1 of `app.js`), browser storage access (e.g., reading a value from `localStorage` in Line 5 of `app.js`), or network communication (e.g., transmitting the sensitive data via `fetch` in Line 3 of `app.js`).

Limitation #2: lack of robustness. Previous approaches are not robust against resource-specific evasion attacks (§5.4). For example, URL-focused approaches can be evaded when phishing pages use URL obfuscation techniques such as URL shortening or the addition of legitimate-looking words [11, 41]. Image-based approaches are vulnerable to image-targeted attacks, including logo masking, manipulation, and font substitution [17, 19, 40]. Approaches that rely on HTML-level signals can likewise be evaded by modifying keywords, injecting benign text, or altering tag structures without changing page functionality [11]. Notably, even approaches that consider multiple resource types (e.g., PhishIntention [33]) can still be bypassed by combining evasion attacks across those resources (§5.4).

Our approach. To overcome these limitations, we propose PHISHXGRAPH, a robust phishing website detection framework. Our key idea is to detect phishing behaviors not only by considering a broad set of webpage resource types (i.e., HTML, JS, storage, and network) but also by modeling the interactions among them (Table 1).

In particular, PHISHXGRAPH leverages a cross-layer interaction graph that represents the HTML, JS, storage, network activity of a webpage as a single heterogeneous directed graph. Figure 1b illustrates the cross-layer interaction graph corresponding to the phishing webpage in Figure 1a. We construct this graph by dynamically recording the webpage’s behavior and creating nodes for observed HTML, JS, storage, and network entities⁴. We then represent the rela-

⁴ Each node is annotated with attributes relevant to phishing detection, such as the tag name for an HTML node and the destination URL for a Network node (§4.1).

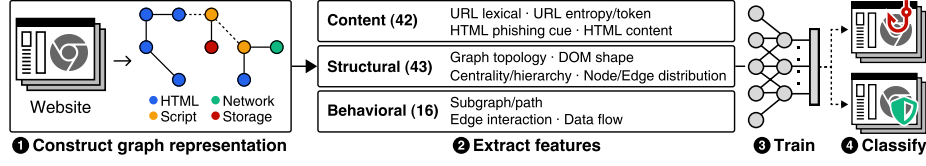


Fig. 2: PHISHXGRAPH workflow.

tionships among these entities as typed edges. For example, we use an Execute edge to denote that an HTML node triggers `app.js`, an Access edge to denote that `app.js` retrieves information from `localStorage`, and a Request edge to denote that the script transmits data outward.

We then classify the webpage as phishing or benign using an ML model trained on 101 features derived from these graphs. By jointly modeling relevant webpage resources and the interactions that realize phishing flows, PHISHXGRAPH effectively distinguishes phishing webpages from benign ones while remaining robust against resource-specific adversarial attacks.

4 Design

In this section, we provide the design of PHISHXGRAPH. Figure 2 shows the overall workflow of PHISHXGRAPH. PHISHXGRAPH **1** constructs a cross-layer interaction graph that captures the interactions among a webpage’s HTML, JavaScript (JS), network requests, and storage operations; **2** extracts graph-derived features from this graph; **3** trains an ML model on those features; and **4** uses the trained model to classify whether the webpage is phishing. Note that model training (**3**) is performed offline on labeled webpages. Once training is complete, classification (**4**) can be applied online to newly observed webpages.

We describe the graph representation (§4.1), its construction process (§4.2), the 101 features extracted from the constructed graph (§4.3), and the ML model used for phishing classification (§4.4).

4.1 Graph Representation

In the browser, a webpage is represented as a Document Object Model (DOM) tree, which forms a hierarchy of HTML elements. PHISHXGRAPH augments this tree with additional nodes and edges for network requests, JS execution, and storage operations. The resulting heterogeneous directed graph captures cross-layer interactions during page execution. Table 2 summarizes the node and edge types used in our cross-layer interaction graph and their descriptions.

Nodes. The graph contains four node types: HTML, Script, Network, and Storage. (1) HTML nodes represent DOM elements (e.g., `form`, `input`, `anchor`, and `image`). Each HTML node is annotated with its attributes (e.g., `src`, `action`, `class`, `id`, and a `has_event_handler` flag). (2) Script nodes represent both `<script>` elements and dynamically loaded scripts. Each Script node is annotated with attributes (e.g., `src` and an `is_inline` flag). (3) Network nodes

Table 2: Node and edge types in the cross-layer interaction graph.

Category	Type	Description
Node	HTML	One node per DOM element in the page
	Script	One node per script entity
	Network	One node per unique requested URL
	Storage	One node per (storage type, key) item
Edge	Structure	One edge per DOM parent-child relationship
	Execute	One edge from a DOM element to its contained script
	Request	One edge per observed HTTP(s) request
	Access	One edge per storage access, linked to the initiating script

represent distinct remote resources. We create one Network node per unique URL requested during page execution, and each node records the URL, domain, `resource_type`, HTTP status, and an `is_third_party` flag. (4) Storage nodes represent individual storage items. We create one Storage node per (storage type, key) tuple, where the storage type can be a cookie, `localStorage`, or `sessionStorage`.

Edges. The graph contains four types of directed edges: Structure, Execute, Request, and Access. (1) Structure edges reflect the DOM parent-child relationship. (2) Execute edges connect an HTML node to its corresponding Script node; for dynamically loaded scripts, the edge originates from the initiating Script node. (3) Request edges represent HTTP requests. We create one Request edge per observed request from the initiating node to the corresponding Network node. For redirects (i.e., HTTP 3xx responses), a Request edge connects the original Network node to the redirected one, yielding a Network-to-Network edge. (4) Access edges represent storage read and write operations. Each access by a script produces an edge from the corresponding Script node to the accessed Storage node, where the initiating script is identified from the captured call stack.

4.2 Graph Construction

The cross-layer interaction graph is constructed dynamically during a webpage visit in Chromium. Specifically, PHISHXGRAPH uses the Playwright [9] together with the Chrome DevTools Protocol (CDP) [4] to capture all four layers and their interactions. Below, we describe how the nodes and edges for each layer are constructed.

Building the HTML layer. We build the HTML layer from the rendered DOM after the page completes its initial dynamic loading and JS execution. We then parse the DOM with BeautifulSoup [2] to construct HTML nodes and Structure edges. Capturing the rendered DOM is important because phishing websites often use JS to dynamically inject or modify phishing content, including credential forms, to evade static analysis [51].

Building the JS layer. Each `<script>` tag becomes a Script node linked to its containing HTML node. For dynamically loaded scripts, we identify script-type network requests that are not associated with any `<script>` tag in the rendered

Table 3: Feature categories extracted from the cross-layer interaction graph.

Category	Subcategory	# Features	Description
Content (42)	URL lexical	19	URL structure and special-symbol indicators
	URL entropy/token	5	URL randomness and suspicious-token indicators
	HTML phishing cue	8	Phishing-related HTML-level cues
	HTML content	10	General HTML-level properties
Structural (43)	Graph topology	9	Overall graph shape (density, connectivity)
	DOM shape	4	Overall DOM-tree shape (depth, breadth)
	Centrality/hierarchy	12	Structural role of nodes
	Node type distribution	11	Cross-layer node composition
	Edge type distribution	7	Cross-layer edge composition
Behavioral (16)	Subgraph/path	8	Form reachability and exfiltration path lengths
	Edge interaction	2	Direct and reachable suspicious connections
	Data flow	6	Exfiltration and redirect patterns
Total		101	

DOM. We then create a corresponding Script node for each such request and connect it to the Script node that initiated the load.

Building the network layer. We construct Network nodes from actual HTTP requests observed in Chromium, rather than from statically parsed HTML URLs. This includes `XMLHttpRequest` (XHR), `fetch`, and `beacon` requests issued by JS, which are common exfiltration channels in modern phishing pages [28]. Requests to the same URL are merged into a single Network node. For each observed request, we create a Request edge from its initiator, i.e., the referencing HTML node or the initiating Script node, to that corresponding Network node. We also preserve 3xx redirect chains between successive Network nodes.

Building the storage layer. We inject a JS hook before page execution to monitor accesses to browser storage APIs, including `document.cookie`, `localStorage`, and `sessionStorage`, and record each access together with its call stack. We create a distinct Storage node for each encountered (`storage type`, `key`) tuple, and each observed access creates an Access edge from the corresponding Script node to that Storage node. This key-level granularity allows PHISHXGRAPH to distinguish legitimate session management from suspicious behavior, such as a script reading a cookie immediately before issuing an external request.

4.3 Feature Extraction

PHISHXGRAPH extracts 101 features from the cross-layer interaction graph. These features are organized into three categories: (1) content, (2) structural, and (3) behavioral. Table 3 summarizes these categories. Table 9 in Appendix presents 35 of the 101 features⁵. We now describe each category in detail.

Content features (42). Content features capture properties of individual resources. In detail, they are derived primarily from the attributes of URL and

⁵ The full list of 101 features is available at <https://github.com/WebSec-Lab/PhishXGraph>

HTML nodes. URL lexical captures the structural characteristics of URLs, such as the URL length (`url_length`) and the presence of special symbols (`has_at_symbol`). URL entropy/token features capture the randomness of URL strings (e.g., URL’s Shannon entropy) or the presence of suspicious tokens (e.g., “login”). Together, these features capture URL patterns commonly associated with phishing, including unusually long, irregular, or raw-IP URLs [24, 35].

HTML phishing cue features capture phishing-related cues from DOM-level information. For example, this subcategory records whether the page contains credential-related terms (e.g., “login”) and brand-related keywords (e.g., “paypal”) [29, 50, 52]. In contrast, HTML content features capture more general DOM-level properties, such as the overall document length (`html_length`) and the character length of `<title>` text (`title_length`).

Structural features (43). Structural features describe relationships between nodes, capturing the graph structure and the interactions among different layers. Graph topology features summarize the overall shape of the graph. For example, they characterize the density (`graph_density`) and connectivity (`average_degree_connectivity`) of the cross-layer graph. DOM shape features characterize the overall shape of the DOM tree, such as its maximum depth and breadth. Centrality/hierarchy features quantify the structural role of each node in the cross-layer graph. This subcategory captures summary statistics of node centrality (mean, max, and min) as well as the numbers of ancestors/descendants associated with each node.

Node and edge type distributions capture how node and edge types are distributed in the graph, such as the fraction of HTML nodes (`html_node_ratio`) and the fraction of Execute edges (`execute_edge_ratio`). They also measure the fraction of redirect edges (`redirect_edge_ratio`) and the fraction of requests sent to cross-domain destinations (`cross_domain_edge_ratio`).

Note that structural features are designed to capture fundamental structural differences between phishing and legitimate webpages, including graph density, graph depth, and the concentration pattern of outgoing requests. Indeed, our results show that many of these features contribute substantially to discriminating between the two classes (§5.2 and §5.5).

Behavioral features (16). Behavioral features capture suspicious multi-layer data flows in the graph, especially end-to-end exfiltration patterns. Subgraph/path features characterize the prominence of the form-related subgraph, for example, how many nodes it reaches, how deep the form subtree is, and how far input nodes are from the document root. Edge interaction and data flow features further capture phishing behaviors by checking whether suspicious cross-layer paths exist, such as paths from forms or storage nodes to outgoing network requests.

4.4 Classification Model

PHISHXGRAPH classifies each webpage using a 101-dimensional feature vector derived from the cross-layer interaction graph. We use gradient-boosted decision trees (XGBoost [13]) as the classifier. Given the feature vector, the classifier

outputs both a binary label and a phishing probability, allowing downstream systems to apply task-specific decision thresholds.

5 Evaluation

We now evaluate PHISHXGRAPH to answer the following research questions:

- **RQ1.** How accurately does PHISHXGRAPH detect phishing webpages? (§5.2)
- **RQ2.** How does PHISHXGRAPH perform compared with state-of-the-art approaches in terms of detecting phishing websites? (§5.3)
- **RQ3.** How robust is PHISHXGRAPH against evasion attacks? (§5.4)
- **RQ4.** Do individual components of PHISHXGRAPH affect the effectiveness of detection? (§5.5)
- **RQ5.** Can PHISHXGRAPH detect emerging phishing campaigns in the wild ahead of existing blacklists? (§5.6)

5.1 Experimental Setup

We conduct all experiments on a server running Ubuntu 24.04 with an Intel Xeon Platinum 8460Y+ CPU (160 cores) and 512 GB of main memory.

Implementation. We implement PHISHXGRAPH in Python 3.10. We use Playwright for graph construction and XGBoost for classification. The graph construction pipeline captures the rendered DOM, dynamically loaded resources, runtime network requests, and storage operations (§4.2). We set the graph construction timeout to 30 seconds per webpage. On average, each crawl takes 15.2 seconds per page from page visit to feature extraction.

Dataset. We construct a balanced dataset of 53K phishing and 53K benign pages. Phishing samples are collected from PhishTank [1] and OpenPhish [8] over a four-month period (Jan–Apr 2026). Benign samples are collected from the Tranco top-55K list [6, 25]. We adopt an 80/20 stratified train/test split, preserving the class distribution across both partitions. We then perform 5-fold cross-validation on the 80% training set, while reserving the 20% test set exclusively for final evaluation.

5.2 Detection Performance and Feature Analysis

We evaluate how accurately PHISHXGRAPH detects phishing pages. As shown in the last row of Table 5, PHISHXGRAPH achieves 98.64% accuracy, 98.39% precision, 96.19% recall, and 97.28% F1 score. We observed that PHISHXGRAPH produces 138 false positives (FPs) and 385 false negatives (FNs). To better understand these errors, we manually examined each misclassified case and analyzed the root causes of misclassification.

FPs. PHISHXGRAPH misclassified 138 legitimate webpages as phishing. A common pattern across these FPs is that, when visited by our crawler, the pages were only partially rendered because of bot checks, CAPTCHAs, consent windows, which obscured the original content. As a result, the observed graph structure

Table 4: Top-20 features ranked by XGBoost average gain. The Category column indicates content (C), structural (S), or behavioral (B).

# Feature	Category	Gain	# Feature	Category	Gain
1 url_length	C	0.1057	11 descendants_std	S	0.0323
2 num_url_tokens	C	0.0990	12 graph_density	S	0.0322
3 request_node_ratio	S	0.0860	13 graph_clustering_coefficient	S	0.0322
4 closeness_cent_std	S	0.0826	14 cross_domain_edge_ratio	S	0.0322
5 request_edge_ratio	S	0.0801	15 resource_dep_depth	B	0.0221
6 closeness_cent_max	S	0.0716	16 title_length	C	0.0119
7 avg_edge_type_entropy	S	0.0458	17 average_degree_connectivity	S	0.0118
8 html_length	C	0.0363	18 script_node_ratio	S	0.0018
9 ancestor_max	S	0.0333	19 structure_edge_ratio	S	0.0017
10 descendants_max	S	0.0327	20 ancestors_mean	S	0.0011

of these pages deviated from that of typical legitimate webpages. Interestingly, 120 of the 138 FPs (87%) were government portals under `*.gov.*` domains. These pages also exhibited URL-level characteristics that overlap with phishing signals, such as government-related keywords, long URLs, and multiple dots, which likely further contributed to the misclassifications.

FNs. PHISHXGRAPH missed 385 phishing webpages. A common root cause is that, across all observed resource types (HTML, script, network, storage), their graphs are structurally and content-wise indistinguishable from those of legitimate sites. These cases fall into four recurring patterns: (1) 170 (44%) are hosted on legitimate platform subdomains (e.g., `publisher[.]microsoftstart[.]com`); (2) 88 (23%) are self-hosted but filled with benign-looking content so that their DOM matches the size distribution of legitimate sites; (3) 81 (21%) are multi-stage phishing kits in which the credential-harvesting step appears only after several user interactions; and (4) 46 (12%) return near-empty or cloaked responses.

We observed that 44% of FPs and 78% of FNs received phishing probabilities within ± 0.015 of the decision threshold. This suggests that webpages near the threshold form an uncertain region in which phishing and legitimate webpages are difficult to distinguish, even with cross-layer interaction graph features. At the same time, this near-threshold region can serve as a useful signal for prioritizing manual review.

Feature importance analysis. Table 4 lists the top 20 features that contribute most to distinguishing phishing webpages from legitimate ones. We rank features by XGBoost average gain [13], which measures the mean reduction in classification error contributed by a feature. We observed that the gains of the top 20 features are distributed across all three feature categories: 15 structural features (67.7%), four content features (29.7%), and a single behavioral feature (2.6%).

Structural features capture fundamental structural patterns of the webpage and therefore play an important role in robust classification. We observed that phishing webpages tend to form flatter/shallower graphs centered around credential collection, whereas legitimate webpages tend to form deeper/denser graphs with richer structure. Features such as `graph_density` and `graph_clustering_coefficient` capture this difference. Also, in general, legitimate webpages often communicate with many third-party services, while phishing webpages usu-

Table 5: Overall detection performance of PHISHXGRAPH and the state-of-the-art phishing detectors.

Category	Tool	Accuracy	Precision	Recall	F1 score
ML-based Approaches	URLNet [24]	94.90%	94.99%	90.59%	92.74%
	LBP [16]	88.00%	93.38%	76.48%	84.10%
	StackModel [30]	94.52%	90.11%	89.26%	89.68%
Reference-based Approaches	Phishpedia [32]	84.27%	83.40%	69.06%	75.56%
	PhishIntention [33]	90.41%	79.64%	69.57%	74.27%
Cross-layer Graph (Ours)	PHISHXGRAPH	98.64%	98.39%	96.19%	97.28%

ally send only a few requests, often to attacker-controlled endpoints. Features such as `request_node_ratio`, `request_edge_ratio`, and `cross_domain_edge_ratio` help distinguish the two classes.

Among the content features, `url_length` and `num_url_tokens` (ranked 1st and 2nd) contribute strongly to classification, because phishing webpages often exhibit longer and more tokenized URLs [24, 35, 42, 48]. Similarly, `html_length` and `title_length` (ranked 8th and 16th) are highly informative, because credential-harvesting pages tend to contain less HTML content and shorter titles than legitimate webpages [31, 37, 50, 52].

The sole behavioral feature in the top 20 is `resource_dep_depth` (ranked 15th), which captures the length of the longest request-dependency chain. Note that the contribution of behavioral features becomes most apparent when they are combined with other feature categories, where they provide complementary signals (§5.5).

5.3 Comparison with State-of-the-Art Phishing Detectors

We compared PHISHXGRAPH against five baselines, including three ML-based approaches (URLNet [24], LBP [16], and StackModel [30]) and two reference-based approaches (Phishpedia [32] and PhishIntention [33]). All baselines were trained and evaluated on the same dataset and under the same experimental setting as PHISHXGRAPH.

Table 5 presents the overall comparison results between PHISHXGRAPH and other approaches. The results clearly show that PHISHXGRAPH outperforms all baselines: compared with PHISHXGRAPH, prior approaches exhibit average performance drops of 14.01% in F1 score and 8.22% in accuracy on average.

ML-based approaches. URLNet and LBP both learn lexical patterns from phishing URLs. Because they rely solely on URL information among the many webpage resources, they suffer an average F1 score drop of 8.86% compared with PHISHXGRAPH. StackModel extracts feature vectors from both the URL and HTML content, achieving 94.52% accuracy, but still shows a 7.6% F1 score drop. Notably, URLNet, LBP, and StackModel capture only a subset of the content features modeled by PHISHXGRAPH. Consequently, they cannot capture phishing behaviors that arise across multiple resources and their interactions.

Table 6: Attack success rate for each attack scenario against phishing detection baselines.

Tool	Attack Scenario						
	URL	HTML	Logo	URL+HTML	URL+Logo	HTML+Logo	URL+HTML+Logo
URLNet	22.45%	–	–	22.45%	22.45%	–	22.45%
LBP	32.10%	–	–	32.10%	32.10%	–	32.10%
StackModel	7.67%	18.77%	–	24.48%	7.67%	18.77%	28.45%
Phishpedia	0.04%	–	37.80%	0.04%	32.52%	32.86%	32.55%
PhishIntention	0.01%	9.50%	30.61%	9.48%	24.42%	24.42%	24.42%
PHISHXGRAPH	1.32%	3.11%	0.00%	4.15%	1.32%	3.11%	4.95%

Reference-based approaches. Phishpedia and PhishIntention rely on recognizable brand cues to match a suspect page against reference data from legitimate brands. When such cues are absent, weak, or hard to localize, their detection performance degrades substantially [19]. This limitation is reflected in our evaluation: Phishpedia and PhishIntention achieve 69.06% and 69.57% recall.

In summary, phishing webpages are too diverse to be reliably handled by prior approaches that observe only one or a few webpage surfaces. In contrast, PHISHXGRAPH considers all relevant webpage resources together with their cross-layer interactions and represents them in a generalizable form as a cross-layer interaction graph. This enables PHISHXGRAPH to effectively distinguish phishing webpages from benign ones across diverse phishing patterns.

5.4 Robustness Analysis

We evaluate the robustness of each phishing detection tool against 10 adversarial phishing evasion attacks [11, 17, 19, 40, 41]. We organize these attacks by the perturbed surface: three URL attacks [11, 41], four HTML attacks [11, 23, 45], and three image (logo-based) attacks [17, 19, 40]. Table 8 in Appendix summarizes the 10 attack variants considered in our evaluation.

For each of the 10 evasion attacks, we generated six adversarial samples for each phishing webpage in the test set. We then evaluated each detection tool on these samples and measured the attack success rate, i.e., the fraction of true positive phishing samples that are misclassified as benign after evasion. In particular, we measure aggregated attack success rate for each perturbation surface, namely URL, HTML, and logo, by jointly considering all attacks generated on that surface. We also consider more sophisticated evasion attacks by combining multiple surfaces, such as URL+HTML and URL+HTML+logo.

Table 6 presents the attack success rate of PHISHXGRAPH and other approaches under various evasion attacks. Lower attack success rate indicates stronger robustness. The symbol – means that the corresponding attack surface is not considered by the detection tool, so attack success rate on that surface is not meaningful.

We observed that all baselines are highly vulnerable when attacks target the resources they primarily use for detection. For example, URLNet shows

Table 7: Results for PHISHXGRAPH’s ablated variants. The capitals in Configuration column indicate content (C), structural (S), or behavioral (B).

Configuration	Accuracy	Precision	Recall	F1 Score
<i>Feature Category Ablation</i>				
C only (42)	92.57%	95.89%	92.26%	94.04%
S only (43)	91.88%	94.79%	93.98%	94.38%
B only (16)	86.28%	90.66%	87.55%	89.08%
C+S (85)	94.84%	94.94%	95.28%	95.11%
C+B (58)	93.58%	92.90%	93.26%	93.08%
S+B (59)	96.06%	95.94%	96.05%	95.99%
C+S+B (101)	98.64%	98.39%	96.19%	97.28%
<i>ML Classifier Selection</i>				
Random Forest	98.60%	96.39%	97.18%	96.78%
SVM	98.42%	96.39%	95.80%	96.09%
MLP	96.50%	95.06%	94.24%	94.65%
XGBoost	98.64%	98.39%	96.19%	97.28%

a high attack success rate of 22.45% against URL-based evasion attacks, while Phishpedia reaches 37.80% against logo-based evasion attacks. Even worse, when evasion attacks are combined across multiple surfaces, the attack success rate increases substantially. In particular, for StackModel, which observes URL and HTML information, the attack success rate under the fully combined attack reaches 28.45%, making evasion much easier.

In contrast, PHISHXGRAPH jointly models not only all three resources but also the interactions among them, enabling it to capture more fundamental characteristics of webpages. As a result, it remains robust against both surface-specific attacks and combined attacks across multiple surfaces, with overall attack success rate staying below 4.96%. This demonstrates that phishing detection based on cross-layer interaction graphs is highly robust to adversarial attacks.

5.5 Ablation Study

We evaluate the effectiveness of (1) the feature categories and (2) the ML classifier used in PHISHXGRAPH. For the feature category analysis, we compare PHISHXGRAPH (i.e., content+structural+behavioral) against ablation variants that use each category in isolation (e.g., content only) or in partial combinations (e.g., content+structural). For the ML classifier analysis, we compare XGBoost with Random Forest, Support Vector Machine (SVM), and MLP (Multi-Layer Perceptron).

Table 7 presents the comparison results between PHISHXGRAPH and its ablated variants. We observe that the content-only and structural-only variants achieve F1 scores of 94.04% and 94.38%, respectively. This result shows that strong detection performance comes not only from modeling the content of individual resources, but also from capturing the interactions among resources.

Although the behavioral-only variant achieves a relatively lower F1 score of 89.08%, its full benefit emerges when it is combined with the other feature categories. These results indicate that the three feature categories are comple-

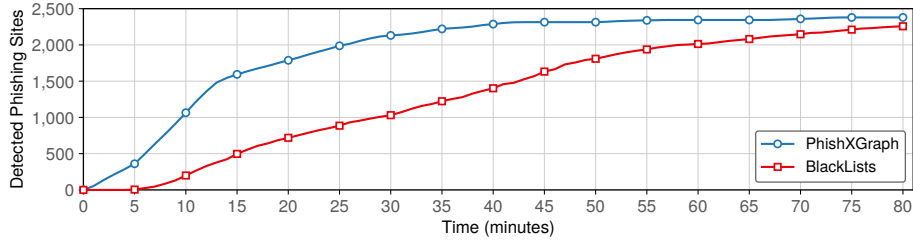


Fig. 3: Detection time of PHISHXGRAPH and blacklists (OpenPhish, PhishTank, and Google Safe Browsing) in a 15-day real-world deployment.

mentary. By integrating all feature categories, PHISHXGRAPH can robustly and generally distinguish phishing webpages from legitimate ones.

In the classifier comparison, XGBoost achieves the strongest overall performance. While Random Forest and SVM yield slightly higher precision, the margin is only about 0.2%. Because the ranking is otherwise stable across all metrics, we use XGBoost as the default classifier in PHISHXGRAPH.

5.6 Emerging Phishing Website Detection

We now evaluate the practicality of PHISHXGRAPH by examining whether it can identify emerging phishing websites ahead of existing blacklists, OpenPhish [8], PhishTank [1], and Google Safe Browsing (GSB) [5]. To this end, we monitored emerging websites collected via CertStream [3] over a 15-day period. CertStream provides a real-time stream of TLS certificates issued through Certificate Transparency logs [39]. This stream enables PHISHXGRAPH to perform proactive classification in near real time.

Over the 15-day period, PHISHXGRAPH identified 2,385 phishing webpages from 254,283 emerging websites. Of these, 1,852 were confirmed as phishing because they were listed in at least one of PhishTank, OpenPhish, and GSB. The remaining webpages were no longer available at the time of our verification, likely due to the short lifespan of phishing webpages [27, 39].

For the 1,852 confirmed phishing webpages, we compared the elapsed time from emergence to PHISHXGRAPH’s detection against the elapsed time from emergence to listing by existing blacklists. As shown in Figure 3, PHISHXGRAPH identified emerging phishing webpages earlier: on average, it flagged a phishing webpage 14 minutes after emergence, whereas the same URLs appeared on a blacklist 36 minutes later. PHISHXGRAPH identified 1,636 webpages within 15 minutes, whereas blacklists required 44 minutes to reach the same count.

Notably, among the 1,852 confirmed phishing webpages, PHISHXGRAPH detected 1,716 earlier than blacklists. These results demonstrate that PHISHXGRAPH can substantially support blacklist maintenance and shorten the period during which end users remain exposed to phishing webpages [38].

6 Discussion

Deployment. PHISHXGRAPH is primarily designed for the efficient, proactive discovery of previously unknown phishing webpages, thereby assisting blacklist compilation and maintenance. PHISHXGRAPH takes 16.6 seconds on average per webpage. This end-to-end latency includes page visit, rendering wait time, graph construction, feature extraction, and model inference. For optimized real-time blocking, a more lightweight detection pipeline (e.g., through feature reduction [26]) is needed; we leave this to future work.

Interaction-gated content. Our analysis of FPs and FNs reveals a recurring pattern: many errors arise when webpages expose only partial content to our crawler, for example until form submission or consent clicks. In such cases, the partially observed page can resemble a legitimate webpage, leading to FNs, or a phishing webpage, leading to FPs. To mitigate this limitation, one promising future direction is to augment PHISHXGRAPH with automated user interaction simulation [14]. This could expose hidden phishing stages and better capture the functional intent of phishing webpages [28].

Potential FPs and FNs. When attackers adversarially mimic the behavior and structure of benign webpages, PHISHXGRAPH may be bypassed. Conversely, when benign webpages exhibit behavior and structure similar to phishing webpages, PHISHXGRAPH may produce FPs. Reducing such FPs and FNs requires richer content and behavioral features to better capture the underlying intent of a webpage. Leveraging LLMs to capture such deeper intent would be an interesting direction, which we leave for future work.

7 Conclusion

We presented PHISHXGRAPH, the phishing detection framework that leverages cross-layer interaction graphs to model the underlying functional and structural properties of webpages. By moving beyond restricted-surface analysis, PHISHXGRAPH captures the phishing patterns in the interactions and data flows across HTML, JavaScript, storage, and network layers. Our evaluation shows that PHISHXGRAPH significantly outperforms state-of-the-art baselines, achieving a 97.28% F1 score. Under adversarial attacks, PHISHXGRAPH achieves the lowest attack success rate, averaging 2.57% compared with 21.58% for the baselines.

Acknowledgments. We would like to thank the anonymous reviewers for their concrete feedback. This work was supported by Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (IITP-2026-RS-2022-00156361 and RS-2025-25394739), National Research Foundation of Korea (NRF) (RS-2025-00561150 and RS-2025-00523578), and the U.S. National Science Foundation (CNS-2440819 and DGE-2335798).

References

1. An anti-phishing site. <https://www.phishtank.com/>

2. Beautiful Soup. <https://www.crummy.com/software/BeautifulSoup/>
3. CertStream – real-time certificate transparency log update stream. <https://certstream.calidog.io/>
4. Devtools protocol. <https://chromedevtools.github.io/devtools-protocol/>
5. Google safe browsing. <https://safebrowsing.google.com/>
6. Information on the tranco list with id 8lqjv. <https://tranco-list.eu/list/8LQJV>
7. Internet crime report 2025. https://www.ic3.gov/AnnualReport/Reports/2025_IC3Report.pdf
8. Phishing intelligence. <https://openphish.com/>
9. Playwright: Fast and reliable end-to-end testing for modern web apps. <https://playwright.dev/>
10. Afroz, S., Greenstadt, R.: PhishZoo: Detecting phishing websites by looking at them. In: IEEE International Conference on Semantic Computing. pp. 368–375 (2011)
11. Apruzzese, G., Conti, M., Yuan, Y.: SpacePhish: The evasion-space of adversarial attacks against phishing website detectors using machine learning. In: Proceedings of the Annual Computer Security Applications Conference. pp. 171–185 (2022)
12. Bilot, T., Geis, G., Hammi, B.: PhishGNN: A phishing website detection framework using graph neural networks. In: International Conference on Security and Cryptography. pp. 428–435 (2022)
13. Chen, T., Guestrin, C.: XGBoost: A scalable tree boosting system. In: Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp. 785–794 (2016)
14. Eriksson, B., Pellegrino, G., Sabelfeld, A.: Black widow: Blackbox data-driven web scanning. In: Proceedings of the IEEE Symposium on Security and Privacy. pp. 1125–1142 (2021)
15. Garera, S., Provos, N., Chew, M., Rubin, A.D.: A framework for detection and measurement of phishing attacks. In: Proceedings of the ACM Workshop on Recurring Malcode. pp. 1–8 (2007)
16. Guo, W., Wang, Q., Yue, H., Sun, H., Hu, R.Q.: Efficient phishing URL detection using graph-based machine learning and loopy belief propagation. In: IEEE International Conference on Communications. pp. 2671–2676 (2025)
17. Hao, Q., Diwan, N., Yuan, Y., Apruzzese, G., Conti, M., Wang, G.: It doesn’t look like anything to me: Using diffusion model to subvert visual phishing detectors. In: Proceedings of the USENIX Security Symposium. pp. 3027–3044 (2024)
18. Hao, S., Kantchelian, A., Miller, B., Paxson, V., Feamster, N.: PREDATOR: Proactive recognition and elimination of domain abuse at time-of-registration. In: Proceedings of the ACM Conference on Computer and Communications Security. p. 1568–1579 (2016)
19. Ji, F., Lee, K., Koo, H., You, W., Choo, E., Kim, H., Kim, D.: Evaluating the effectiveness and robustness of visual similarity-based phishing detection models. In: Proceedings of the USENIX Security Symposium. pp. 3201–3220 (2025)
20. Kim, D., Zhao, H., Kim, D., Hwang, S.: Unveiling the underground phishing ecosystem: A 12-year longitudinal study of deep and dark web forums. In: Proceedings of the Web Conference. p. 3531–3542 (2026)
21. Kim, D., Cho, H., Kwon, Y., Doupé, A., Son, S., Ahn, G.J., Dumitras, T.: Security analysis on practices of certificate authorities in the HTTPS phishing ecosystem. In: Proceedings of the ACM Symposium on Information, Computer and Communications Security. p. 407–420 (2021)

22. Kim, T., Park, N., Hong, J., Kim, S.W.: Phishing URL detection: A network-based approach robust to evasion. In: *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*. pp. 1769–1782 (2022)
23. Kulkarni, A., Balachandran, V., Divakaran, D.M., Das, T.: From ML to LLM: Evaluating the robustness of phishing web page detection models against adversarial attacks. *Digital Threats* (2025)
24. Le, H., Pham, Q., Sahoo, D., Hoi, S.C.H.: URLNet: Learning a URL representation with deep learning for malicious URL detection. *arXiv preprint arXiv:1802.03162* (2018)
25. Le Pochat, V., Van Goethem, T., Tajalizadehkhoob, S., Korczyński, M., Joosen, W.: Tranco: A research-oriented top sites ranking hardened against manipulation. In: *Proceedings of the Network and Distributed System Security Symposium* (2019)
26. Lee, K., Lim, C., Jin, B., Kim, T., Kim, H.: AdFlush: A real-world deployable machine learning solution for effective advertisement and web tracker prevention. In: *Proceedings of the Web Conference*. pp. 1902–1913 (2024)
27. Lee, K., Lim, K., Kim, H., Kwon, Y., Kim, D.: 7 days later: Analyzing phishing-site lifespan after detected. In: *Proceedings of the Web Conference*. p. 945–956 (2025)
28. Lee, W., Hur, J., Kim, D.: Beneath the phishing scripts: A script-level analysis of phishing kits and their impact on real-world phishing websites. In: *Proceedings of the ACM Symposium on Information, Computer and Communications Security*. pp. 1214–1230 (2024)
29. Li, Y., Huang, C., Deng, S., Lock, M.L., Cao, T., Oo, N., Lim, H.W., Hooi, B.: KnowPhish: Large language models meet multimodal knowledge graphs for enhancing reference-based phishing detection. In: *Proceedings of the USENIX Security Symposium*. pp. 793–810 (2024)
30. Li, Y., Yang, Z., Chen, X., Yuan, H., Liu, W.: A stacking model using url and html features for phishing webpage detection. *Future Generation Computer Systems* (2019)
31. Lim, K., Park, J., Kim, D.: Phishing vs. legit: Comparative analysis of client-side resources of phishing and target brand websites. In: *Proceedings of the Web Conference*. p. 1756–1767 (2024)
32. Lin, Y., Liu, R., Divakaran, D.M., Ng, J.Y., Chan, Q.Z., Lu, Y., Si, Y., Zhang, F., Dong, J.S.: Phishpedia: A hybrid deep learning based approach to visually identify phishing webpages. In: *Proceedings of the USENIX Security Symposium*. pp. 3793–3810 (2021)
33. Liu, R., Lin, Y., Yang, X., Ng, S.H., Divakaran, D.M., Dong, J.S.: Inferring phishing intention via webpage appearance and dynamics: A deep vision based approach. In: *Proceedings of the USENIX Security Symposium*. pp. 1633–1650 (2022)
34. Liu, R., Lin, Y., Zhang, Y., Lee, P.H., Dong, J.S.: Knowledge expansion and counterfactual interaction for reference-based phishing detection. In: *Proceedings of the USENIX Security Symposium*. pp. 4139–4156 (2023)
35. Ma, J., Saul, L.K., Savage, S., Voelker, G.M.: Beyond blacklists: Learning to detect malicious web sites from suspicious URLs. In: *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2009)
36. Manguli, K., Kondaiah, C., Pais, A.R., Rao, R.S.: GraPhish: A graph-based approach for phishing detection from encrypted TLS traffic. *Journal of Information Security and Applications* (2025)
37. Mohammad, R.M., Thabtah, F., McCluskey, L.: Intelligent rule-based phishing websites classification. *IET Information Security* (2014)

38. Oest, A., Safaei, Y., Zhang, P., Wardman, B., Tyers, K., Shoshitaishvili, Y., Doupé, A., Ahn, G.J.: PhishTime: Continuous longitudinal measurement of the effectiveness of anti-phishing blacklists. In: *Proceedings of the USENIX Security Symposium*. pp. 379–396 (2020)
39. Oest, A., Zhang, P., Wardman, B., Nunes, E., Burgis, J., Zand, A., Thomas, K., Doupé, A., Ahn, G.J.: Sunrise to sunset: analyzing the end-to-end life cycle and effectiveness of phishing attacks at scale. In: *Proceedings of the USENIX Security Symposium*. pp. 361–377 (2020)
40. Roh, E., Jeon, S., Son, S., Hong, S.: Evaluating robustness of reference-based phishing detectors. In: *Proceedings of the ACM Symposium on Information, Computer and Communications Security* (2025)
41. Sabir, B., Babar, M.A., Gaire, R., Abuadbbba, A.: Reliability and robustness analysis of machine learning based phishing url detectors. *IEEE Transactions on Dependable and Secure Computing* (2022)
42. Sahingo, O.K., Buber, E., Demir, O., Diri, B.: Machine learning based phishing detection from URLs. *Expert Systems with Applications* (2019)
43. Sanchez-Rola, I., Bilge, L., Balzarotti, D., Buescher, A., Efstathopoulos, P.: Rods with laser beams: Understanding browser fingerprinting on phishing pages. In: *Proceedings of the USENIX Security Symposium*. pp. 4157–4173 (2023)
44. Sheng, S., Wardman, B., Warner, G., Cranor, L.F., Hong, J., Zhang, C.: An empirical analysis of phishing blacklists. In: *Proceedings of the Conference on Email and Anti-Spam* (2009)
45. Song, F., Lei, Y., Chen, S., Fan, L., Liu, Y.: Advanced evasion attacks and mitigations on practical ML-based phishing website classifiers. *International Journal of Intelligent Systems* (2021)
46. Song, T., Casas, P., Meo, M.: Phishing the phishers with SpecularNet: Hierarchical graph autoencoding for reference-free web phishing detection. *arXiv preprint arXiv:2603.01874* (2026)
47. Subramani, K., Melicher, W., Starov, O., Vadrevu, P., Perdisci, R.: PhishInPatterns: measuring elicited user interactions at scale on phishing websites. In: *Proceedings of the ACM Internet Measurement Conference*. pp. 589–604 (2022)
48. Tajaddodianfar, F., Stokes, J.W., Gururajan, A.: Texception: A character/word-level deep learning model for phishing URL detection. In: *IEEE International Conference on Acoustics, Speech and Signal Processing*. pp. 2637–2641 (2020)
49. Vuong, T.L., Do, A.T., Do, V.H.: Enhancing phishing URL detection with graph neural networks: A combination of URL and HTML features. In: *International Conference on Computational Collective Intelligence*. pp. 111–123 (2025)
50. Xiang, G., Hong, J.I., Rosé, C.P., Cranor, L.: CANTINA+: A feature-rich machine learning framework for detecting phishing web sites. *ACM Transactions on Information and System Security* (2011)
51. Zhang, P., Oest, A., Cho, H., Sun, Z., Johnson, R.C., Wardman, B., Sarker, S., Kapravelos, A., Bao, T., Wang, R., Shoshitaishvili, Y., Doupé, A., Ahn, G.J.: CrawlPhish: Large-scale analysis of client-side cloaking techniques in phishing. In: *Proceedings of the IEEE Symposium on Security and Privacy*. pp. 1109–1124 (2021)
52. Zhang, Y., Hong, J.I., Cranor, L.F.: CANTINA: A content-based approach to detecting phishing web sites. In: *Proceedings of the international conference on World wide web*. pp. 639–648 (2007)
53. Zscaler ThreatLabz: Technical analysis of the BlackForce phishing kit (2025), <https://www.zscaler.com/blogs/security-research/technical-analysis-blackforce-phishing-kit>

A Appendix

Table 8: Evasion attacks used in the robustness study, grouped into three attack surfaces and 10 variants.

Attack Surface	Attack Variant	Variant Explanation
URL Perturbation	URL shortening [11]	Substitute with a shortened URL
	URL combosquatting [41]	Substitute path or subdomain
	TLD substitution [41]	Substitute TLD
HTML Perturbation	Inject invisible links [11]	Add invisible internal link
	Object ratio manipulation [11]	Add dummy tags
	DOM structure manipulation [23]	Add benign DOM node or text
	Input Obfuscation [45]	Modify input type from password to text
Logo Perturbation	Logo masking [40]	Mask 50% of logo
	Logo manipulation [19]	15° rotation
	Font substitution [17]	DejaVu Sans re-render

Table 9: (Selected) list of features, grouped by category and subcategory. Type column indicates continuous (C), integer (I), or binary (B) values.

#	Feature Name	Type	Definition
Content features (42)			
URL Lexical (19)			
1	url_length	I	Number of characters in the URL
2	has_at_symbol	B	URL contains '@'
URL Entropy & Token (5)			
3	url_entropy	C	Shannon entropy of the URL's character distribution
4	num_url_tokens	I	Token count after splitting on delimiters and filtering stop words (e.g., http, https, www, com, net, html, php, etc.)
HTML Phishing Cue (8)			
5	urgency_keyword_ratio	C	Ratio of DOM tags containing urgency language (e.g., verify, suspend, expire, urgent, locked, unauthorized, etc.)
HTML Content (10)			
6	html_length	I	Total length of the rendered HTML source
7	title_length	I	Character length of <title> text
Structural features (43)			
Graph Topology (9)			
8	graph_density	C	Ratio of edges to all possible directed edges
9	graph_clustering_coefficient	C	Average clustering coefficient (undirected)
10	average_degree_connectivity	C	Mean of the average neighbor degree per degree class
DOM Shape (4)			
11	dom_depth_max	I	Maximum depth of the DOM tree via structure edges
12	dom_breadth_max	I	Maximum number of children of any HTML node
Centrality & Hierarchy (12)			
13	closeness_centrality_std	C	Std. dev. of closeness centrality
14	closeness_centrality_max	C	Maximum closeness centrality
15	ancestors_mean	C	Mean ancestor count per node (directed)
16	ancestors_max	I	Maximum ancestor count
17	descendants_std	C	Std. dev. of descendant counts
18	descendants_max	I	Maximum descendant count
Node Type Distribution (11)			
19	html_node_ratio	C	Fraction of nodes of type html
20	script_node_ratio	C	Fraction of nodes of type script
21	request_node_ratio	C	Fraction of nodes of type network
Edge Type Distribution (7)			
22	structure_edge_ratio	C	Fraction of edges of type structure
23	request_edge_ratio	C	Fraction of edges of type request
24	execute_edge_ratio	C	Fraction of edges of type execute
25	redirect_edge_ratio	C	Fraction of all edges that are redirect requests
26	cross_domain_edge_ratio	C	Fraction of request edges targeting third-party domains
27	avg_edge_type_entropy	C	Shannon entropy of the edge type distribution
Behavioral features (16)			
Subgraph & Path Analysis (8)			
28	form_subgraph_depth	I	Max depth of form subtrees along structure edges
29	input_to_root_avg_distance	C	Avg. shortest path from input nodes to root (reversed graph)
30	script_to_input_path_length	C	Avg. shortest path from JavaScript nodes to input nodes
31	resource_dependency_depth	I	Max depth of chained request→request edges (incl. redirects)
Edge Interaction Patterns (2)			
32	form_to_external_edge	B	Any form node has a direct third-party request successor
33	script_to_form_path_exists	B	Any JavaScript node can reach a form node via a directed path
Data Flow (6)			
34	form_data_exfil_path	B	Directed path from form → third-party request
35	input_to_request_flow	B	Directed path from input → any request