PhishLang: A Real-Time, Fully Client-Side Phishing Detection Framework Using MobileBERT

Sayak Saha Roy
The University of Texas at Arlington
sayak.saharoy@mavs.uta.edu

Shirin Nilizadeh
The University of Texas at Arlington
shirin.nilizadeh@uta.edu

***Abstract*—We present PhishLang, the first fully client-side anti-phishing framework implemented as a Chromium-based browser extension. PhishLang enables real-time, on-device detection of phishing websites by utilizing a lightweight language model (MobileBERT). Unlike traditional heuristic or static feature-based models that struggle with evasive threats, and deep learning approaches that are too resource-intensive for client-side use, PhishLang analyzes the contextual structure of a page's source code, achieving detection performance on par with several state-of-the-art models while consuming up to 7 times less memory than comparable architectures. Over a 3.5-month period, we deployed the framework in real-time, successfully identifying approximately 26k phishing URLs, many of which were undetected by popular antiphishing blocklists, thus demonstrating PhishLang's potential to aid current detection measures. On the other hand, the browser extension outperformed several anti-phishing tools, detecting over 91% of the threats during zero-day. PhishLang also showed strong adversarial robustness, resisting 16 categories of realistic problem space evasions through a combination of parser-level defenses and adversarial retraining. To aid both end-users and the research community, we have open-sourced both the PhishLang framework and the browser extension.**

I. Introduction

Phishing attacks have led to numerous data breaches and credential theft incidents, resulting in financial losses surpassing $52 million and affecting over 300,000 users in the past year alone [4], [81]. These scams are particularly effective due to sophisticated social engineering techniques [24] that allow attackers to imitate legitimate websites. In response, researchers have explored the various techniques employed by attackers [17], [6], [31], using them to develop countermeasures that utilize distinctive phishing indicators, such as URL characteristics, network behavior, and heuristic features in source code [54], [77]. A key advantage of feature-based phishing detection models is their resource efficiency. These models are lightweight and easily scalable, making them well-suited for real-world deployment where millions of URLs must be analyzed daily [57]. As a result, they continue to be favored by security vendors for their in-house detection pipelines, where fast inference is essential for identifying malicious URLs early [55], [61]. However, the effectiveness of these models is highly reliant on their feature sets, which become outdated [56], unavailable [55], or adversarially exploited by attackers over time [93], [71]. Updating the models requires not only engineering new detection rules or strategies but also collecting revised groundtruth datasets. This creates a significant lag in detection, during which novel phishing campaigns go undetected or are caught too late [57], leaving end-users exposed to attacks. Thus, recent efforts have focused on models that rely on unsupervised or readily available features. Among these, several screenshot-based detection approaches have gained prominence [5], [47], that utilize the visual appearance of websites to flag suspicious activity. While these models report high accuracy in controlled settings, they are rarely deployed at scale due to significant resource demands, both in terms of memory and computational load of processing images, coupled with the ease with which adversaries can manipulate visual elements to evade detection [93], [90].

Another line of work involves using similarity-based algorithms to detect phishing by identifying structural or content-level overlap with known malicious websites [50], [65], [95]. For instance, PhishSim [65] classifies websites based on the compressibility patterns of HTML DOMs. However, such methods implicitly assume that future phishing pages will resemble past ones, an assumption that fails against zero-day attacks or novel phishing kits with unseen templates [57]. Moreover, because these approaches lack semantic understanding, they often misclassify benign but structurally similar pages and struggle to detect cleverly obfuscated threats [91], [92].

A more promising direction approach involves the use of language models (LM), which offer a deeper semantic understanding of webpage content, while relying on a much lighter modality, i.e., text. However, existing LM-based approaches face several critical limitations that prevent real-time deployment. For example, models fine-tuned on static features such as BERT-based frameworks [27], [32] suffer from the same limitations of traditional feature-based models. Prompt-based methods using commercial LLMs such as ChatGPT [39], [15] offer a feature-agnostic alternative and have demonstrated strong performance in phishing detection. However, these models are proprietary and accessed via paid APIs, making them cost-prohibitive for large-scale, real-time deployment. Some hybrid frameworks, such as PhishLLM [48], further combine commercial LLMs with vision-based components, compounding both the computational burden and financial cost. While a few efforts have explored using open-source LLMs, such as Llama for phishing detection [34], [84], these models typically require significantly more computational resources and are slow at scale. Moreover, they often underperform compared

Network and Distributed System Security (NDSS) Symposium 2026
23-27 February 2026, San Diego, CA, USA
ISBN 979-8-9919276-8-0
https://dx.doi.org/10.14722/ndss.2026.231037
www.ndss-symposium.org

to their commercial LLM counterparts [41].

Compounding these issues, many of these academic models [49], [47], [45], [39], [65] have not been evaluated under adversarial conditions or benchmarked against operational baselines like anti-phishing blocklists. As a result, their performance in real-world threat landscapes remains largely untested.

Another major challenge faced by security vendors stems from the sheer volume of URLs that must be processed to find malicious ones that can be added to blocklists [56]. This often results in situations where a vendor is capable of detecting a malicious URL, but has not been able to evaluate it yet, leaving end users temporarily unprotected [57]. One promising solution is to shift detection to the client side, enabling instant decisions at the point of user interaction. However, client-side environments are heavily resource-constrained and require a delicate balance between detection accuracy and computational efficiency. For example, Google's client-side phishing detector, while optimized for speed, relies on significantly weaker models and has been evaluated to have a detection rate of only 14% against real-world threats [64], highlighting the trade-offs currently required in such settings.

To address these limitations, we introduce PhishLang, an efficient and transparent phishing detection framework utilizing MobileBERT, a lightweight language model. Instead of relying on static feature sets, PhishLang uses linguistic patterns in the website source code by focusing on the actionable portions of the page, such as forms, links, and scripts, capturing subtle structural and textual cues attackers use, while avoiding resource-heavy or redundant artifacts from the website. This allows the model to operate with significantly lower resource requirements, using up to almost seven times less memory than prevalent open-source phishing detection models, while also outperforming them in detection efficiency. Deploying the framework in real-time over 3.5 months, PhishLang identified 25,796 phishing URLs, many of which were missed by popular anti-phishing systems, thus highlighting its potential to significantly improve threat coverage. It also successfully detected evasive phishing attacks across five major threat families with an average detection accuracy of 94.7%.

Crucially, PhishLang is not only efficient *but also resilient*. We evaluate its robustness against 16 realistic adversarial attacks that simulate problem-space perturbations. We found that PhishLang demonstrates strong out-of-the-box protection against the majority of the adversarial attacks. And for those that were initially successful in evading detection, we conducted a systematic analysis of their evasion strategies and developed targeted countermeasures, either by enhancing the HTML parser or incorporating adversarial training, resulting in further improvement of the tool's robustness.

Thus, not only can PhishLang be easily deployed on a real-time system, but to the best of our knowledge, we have also developed the first open-source, Chromium-based browser extension that *operates entirely on the client side*, running locally on the user's device without relying on an online blocklist. It is designed to be highly efficient, with minimal memory usage, allowing it to run smoothly even on low-end hardware without compromising detection performance. By benchmarking the extension against popular anti-phishing tools, we find that it significantly surpasses these tools in identifying new *zero-day* threats. Figure 1 provides a high-level overview of the PhishLang pipeline, illustrating the end-to-end flow of our framework, including model construction and evaluation, real-world deployment for live threat detection, comparison with security blocklists, adversarial robustness analysis, and fully client-side implementation.

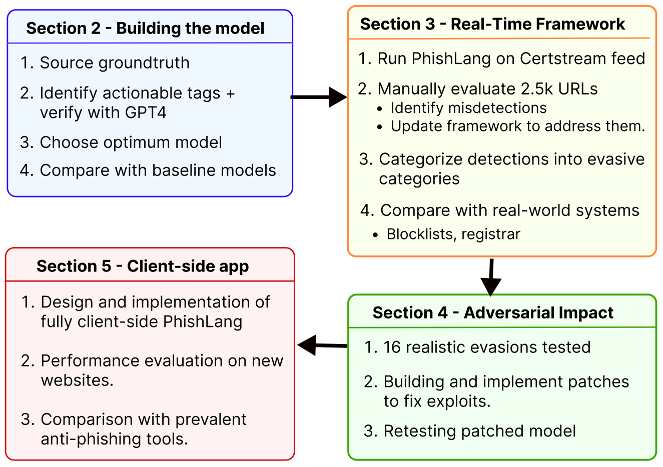
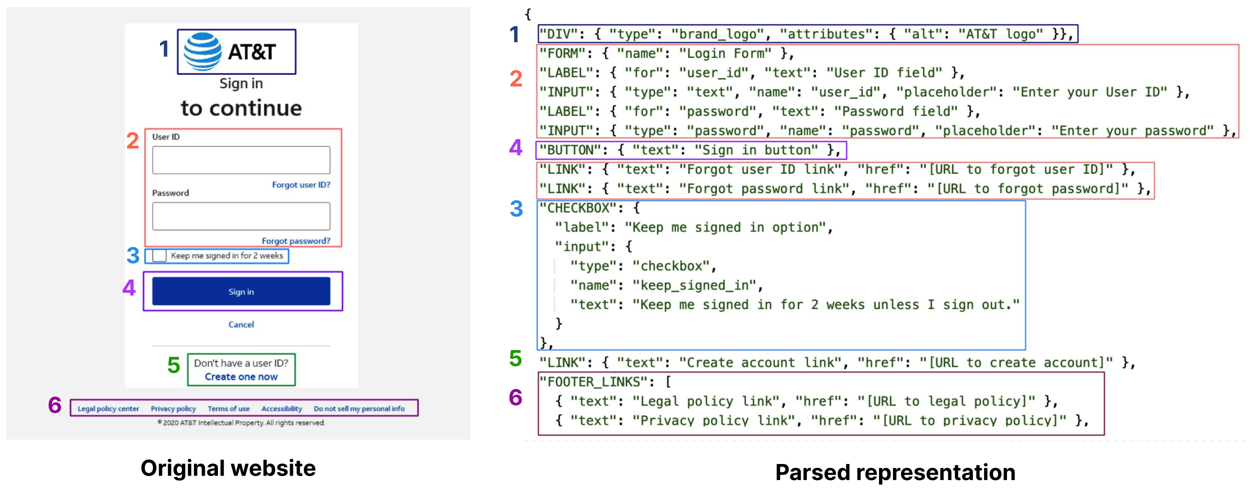

Section 2 - Building the model
1. Source groundtruth
2. Identify actionable tags + verify with GPT4
3. Choose optimum model
4. Compare with baseline models
Section 3 - Real-Time Framework
1. Run PhishLang on Certstream feed
2. Manually evaluate 2.5k URLs
• Identify misdetections
• Update framework to address them.
3. Categorize detections into evasive categories
4. Compare with real-world systems
• Blocklists, registrar
Section 4 - Adversarial Impact
1. 16 realistic evasions tested
2. Building and implement patches to fix exploits.
3. Retesting patched model
Section 5 - Client-side app
1. Design and implementation of fully client-side PhishLang
2. Performance evaluation on new websites.
3. Comparison with prevalent anti-phishing tools.

Fig. 1: Overview of the PhishLang pipeline

The primary contributions of our work are:

1) We built a lightweight open-source phishing detection framework that can not only outperform several popular machine learning-based models, but also provide a better trade-off between inference speed, space complexity, and performance, making it more viable for real-world implementations.
2) Running PhishLang on a live URL stream of URLs from 28th September 2023 to January 11th, 2024, we identified 25,796 unique phishing websites. Lower coverage of these attacks by both blocklists and URL hosting providers, especially for evasive attacks, led to PhishLang reporting and successfully bringing down 74% of these threats, indicating its usefulness to the anti-phishing ecosystem.
3) We evaluate PhishLang against 16 realistic adversarial attacks that make perturbations in the problem space without modifying the layout of the website, developing six countermeasures (patches) that further improve the model's robustness.
4) We implement PhishLang as a client-side Chromium browser extension that can be run on low-end hardware. By benchmarking against widely used anti-phishing tools, we find that PhishLang significantly outperforms all of them in detecting *zero-day phishing attacks* directly within the user's browser, achieving a 91.4% detection rate - more than double that of the next-best performing tool.
5) To aid end-users and the research community, we open-source both the PhishLang framework and browser extension at: https://github.com/UTA-SPRLab/phishlang.



```
{
"DIV": { "type": "brand_logo", "attributes": { "alt": "AT&T logo" }},
"FORM": { "name": "Login Form" },
"LABEL": { "for": "user_id", "text": "User ID field" },
"INPUT": { "type": "text", "name": "user_id", "placeholder": "Enter your User ID" },
"LABEL": { "for": "password", "text": "Password field" },
"INPUT": { "type": "password", "name": "password", "placeholder": "Enter your password" },
"BUTTON": { "text": "Sign in button" },
"LINK": { "text": "Forgot user ID link", "href": "[URL to forgot user ID]" },
"LINK": { "text": "Forgot password link", "href": "[URL to forgot password]" },
"CHECKBOX": {
  "label": "Keep me signed in option",
  "input": {
    "type": "checkbox",
    "name": "keep_signed_in",
    "text": "Keep me signed in for 2 weeks unless I sign out."
  }
},
"LINK": { "text": "Create account link", "href": "[URL to create account]" },
"FOOTER_LINKS": [
  { "text": "Legal policy link", "href": "[URL to legal policy]" },
  { "text": "Privacy policy link", "href": "[URL to privacy policy]" },
```

Fig. 2: Example of a parsed snippet for a phishing website, with the parsed features mapped. The parsing focuses on extracting actionable features from the website.

II. Building the Model

Language Models (LMs) are highly effective at capturing the semantics of natural language, enabling them to recognize subtle linguistic cues and structural patterns in text [94], [74]. Importantly, this capability extends beyond human language: researchers have successfully applied models like BERT for source code analysis, where contextual embeddings between tokens support tasks such as code completion [18] and malware detection [68], [89].

Building on this foundation, our goal was to develop a model that surpasses prior code-similarity-based phishing detection approaches, which typically depend on low-level syntactic matching [69], [50] or unsupervised clustering of raw HTML [65], [52]. In a way, these methods operate more like pattern-matching tools, flagging phishing pages based on superficial resemblance to previously observed structures without truly understanding the intent or function of the underlying code. Consequently, their learned representations are tightly coupled to surface-level artifacts in the training data, including specific tag sequences, layout conventions, or boilerplate components commonly reused across phishing kits. As prior adversarial studies have shown [90], [91], [26], even modest changes, such as reordering elements, inserting inconsequential filler content, or obfuscating tags, can effectively bypass detection. These approaches also often fail to generalize to zero-day phishing campaigns or novel attack templates that depart from familiar patterns [56].

In contrast, our approach utilized the deeper semantic representations produced by language models to focus on actionable components in the source code where phishing behaviors most often manifest. Specifically, PhishLang is not concerned with matching entire code templates or page layouts, but instead focuses on understanding what types of content are typically stored within specific HTML tags and how those contents interact across tags in both phishing versus benign websites. The goal is to capture behavioral intent and contextual relationships, for example, how a deceptive login prompt in a `<form>` tag might be paired with a misleading `<title>` or a suspicious `<a>` link.

To enable this, we needed to train PhishLang on a large corpus of real-world website source code, for which we utilized Lin at el's PhishPedia dataset [47], which includes 30K phishing and 30K benign websites. PhishPedia is not only the largest publicly available phishing dataset but is also notable for its reliability, having been used in several prior literature [49], [30]. Each entry in this dataset typically includes associated metadata, such as HTML source code, screenshots, and OCR text. The phishing websites in this dataset were originally obtained from OpenPhish [58], a widely recognized antiphishing blocklist. These websites were preprocessed to eliminate inactive sites and false positives, and the researchers also implemented strategies to counteract evasive tactics like cloaking and conducted manual verifications to correct any inaccuracies in target brand representations. Conversely, the benign URLs were sourced from the Alexa Rankings list [3], a (now defunct) online database that ranked websites based on popularity. For training PhishLang, we only used the HTML source code of each site. Among the 30,000 phishing samples in the PhishPedia dataset, 22,419 included valid HTML files. For the benign set, 26,000 out of 30,000 entries had usable HTML.

However, a key constraint of language models is their limited input token budget; exceeding this limit leads to dropped information and loss of context. As such, we needed to parse the source code input in such a way so as to retain only the most semantically relevant components.

A. Parsing the source code

A key design challenge in building PhishLang was to represent the HTML input in a way that retained the most semantically meaningful content while staying within the input token budget of modern language models. Much of a website's source code contains aesthetic or structural elements - such as layout containers, styling directives, and formatting tags- that are not useful for phishing detection [35]. Prior work has also shown that phishing websites can be visually polished [5], sometimes even reusing templates from legitimate sites [72], making visual fidelity an unreliable signal [16].

To address this, we aimed to isolate HTML tags that consistently convey user-facing or interactive content, while excluding those used purely for styling or layout. For instance, tags like `<form>`, `<input>`, `<a>`, and `<button>` often reflect a site's behavioral intent, while tags like `<style>` or `<span>` rarely contribute meaningful signals. To determine which tags to retain, two researchers with expertise in phishing and web security manually analyzed a randomly selected subset of 500 phishing websites from the PhishPedia dataset. For each site, they cross-referenced the HTML with the rendered view and screenshot to identify tags that consistently carried semantically rich or interactive content. Going through benign websites was not strictly necessary, as the goal was not to find phishing-specific cues but to select general-purpose tags that typically contain meaningful content across websites. Forms, links, buttons, and headings are foundational to web interaction and appear in both benign and malicious contexts. By analyzing phishing sites alone, we ensured that the retained tags were sufficient for phishing detection, while still being general-purpose.

The inspection revealed that `<h1>`, `<h2>`, and `<h3>` heading tags were typically used to capture user attention with alarming or misleading statements. The `<p>` tag, which holds paragraph text, frequently contained deceptive narratives or persuasive prompts. Hyperlink tags (`<a>`) were identified as critical indicators due to their use in redirecting users to malicious domains or spoofed URLs. List tags such as `<ul>`, `<ol>`, and `<li>` were often used to structure fraudulent instructions or outline fake benefits, while `<form>` tags served as the main mechanism for capturing sensitive user data, making them highly indicative of phishing intent. Similarly, `<input>` elements embedded within forms were frequently used to solicit credentials under false pretenses. The coders also highlighted the importance of the `<title>` tag, which appears in the browser tab and can be manipulated to mimic trusted services, for example, a phishing site might display "Secure Banking Login" to appear legitimate. They also included `<footer>` tags in their analysis as well, noting that phishing sites tend to leave this section sparse or empty [92]. Other selected tags included `<script>` (used to execute malicious behaviors like keylogging or dynamic content injection), `<button>` (triggering phishing actions such as fake logins or downloads), and `<iframe>` (embedding concealed or external phishing content). Additionally, `<meta>` tags involved in auto-redirection, `<a>` tags with JavaScript handlers (e.g., `onclick`), and image-based navigation elements like `<map>` and `<area>` were included for their role in facilitating deception through interactivity or obfuscation.

We use these tags to build a custom parser, which, for a given website's source code, extracts the textual content and selected attributes of each tag, converting them into a structured representation. Each element is represented by its tag type, followed by its content and relevant attributes. Figure 2 illustrates a phishing website and its subsequently generated parsed representation. Even tags that do not contain any elements are preserved with $< EMPTY >$ tag notations. This is because phishing sites frequently utilize empty $< div >$ or $< a >$ tags to simulate legitimacy, misleading the users.

TABLE I: Performance of various language models. Time and Memory denote median time and Memory (DRAM/VRAM based on model) required for inference.

Model	Accuracy	Precision	Recall	F1 Score	Time (s)	Memory
MobileBERT	**0.96**	**0.95**	**0.96**	**0.96**	**0.39**	**74MB**
DistilBERT	0.94	0.94	0.94	0.94	0.85	502MB
DeBERTa	0.83	0.83	0.84	0.84	1.02	1,341MB
FastText	0.62	0.65	0.63	0.63	1.43	201MB
GPT-2	0.68	0.64	0.69	0.68	1.81	922MB
TinyBERT	0.85	0.88	0.82	0.84	0.78	495MB
Llama2 (7B)	0.97	0.96	0.96	0.96	33.71	4,873MB
T5-base	0.89	0.88	0.88	0.88	12.40	1,279MB
Bloom (560M)	0.75	0.74	0.74	0.74	7.49	7,352MB

Text inside the tags is converted to lowercased to reduce the vocabulary size for the LLM.

B. *Choosing the optimum model*

We trained and tested several language models on the (parsed) training data to identify the one that would be most suitable for building our classifier with respect to detection performance, resource consumption, and speed. Specifically, these included nine language models that are considered to provide good performance in binary classification tasks (the number of parameters for the models in parenthesis): DistilBERT(66M) [73], TinyBERT(15M) [37], DeBERTA-base (100M) [33], MobileBERT (25M) [79], FastText (10M) [29], and GPT-2 (117M) [66], Llama2 (7B) [83], Bloom (1.1B) [43] and T5-Base (220M) [67]. Our aim was to choose the model that provides the best trade-off between performance, speed, and memory usage. Each model was fine-tuned using 5-fold cross-validation on 70% of our labeled phishing dataset, with the remaining 30% held out as the test set. Within each fold, models were trained for 10 epochs, and performance was averaged across the folds. For consistency, we used a maximum sequence length of 512 tokens and trained all Transformer-based models using the AdamW optimizer with a learning rate of 2e-5, batch size of 32, and binary cross-entropy loss. AdamW was configured with `weight_decay=0.01` and `eps=1e-8`. All models used their corresponding pretrained tokenizers from HuggingFace Transformers [87], with vocabulary sizes ranging from approximately 30,000 to 50,000 tokens depending on the architecture. FastText was trained using its native implementation with stochastic gradient descent (SGD). Sequences were right-padded and truncated to the fixed maximum length, and a fixed random seed (42) was used to ensure consistency across experiments.

The performance of the models is illustrated in Table I. We found Llama2 to have the best performance out of all the models tested, with an F1 score of 0.96. However, its median inference time of 34 seconds is impractical for real-world usage, where a phishing detection tool would need to process hundreds of thousands of URLs every day, and it also has the highest median memory usage at 4,873.47MB. While the other two LLMs, Bloom and T5, had lower median inference times of 7.49 and 12.40 seconds, respectively, they also performed worse (F1 scores of 0.88 and 0.74, respectively) and had high memory requirements (7351.89MB for Bloom and 1,279.49MB for T5). GPT-2 had the lowest median inference time (1.81 seconds) but also had the worst performance among the LLMs (F1 0.68) and required 922.41MB of memory. Moreover, all four LLMs required inference over GPUs (while they can be used for inference using the CPU, the prediction times will be much slower), and consequently required a large amount of VRAM for said inference per sample. Requiring GPU to provide good inference time is not ideal in a practical setting, as most consumer-end devices might not have access to a GPU.

When considering the Small Language Models (SLMs), MobileBERT had the highest accuracy level at 0.96, with both precision and recall scores of 0.95 and 0.96, respectively. It also demonstrated the fastest prediction times among the SLMs, with a median of 0.39 seconds per prediction, and required the least memory usage at 74MB among all models tested. These attributes make MobileBERT highly suitable for real-time phishing detection, balancing high performance with low computational resource requirements, which is ideal for deployment on devices with limited hardware capabilities, such as mobile and portable systems. Its efficient use of DRAM instead of requiring a GPU makes it even more attractive for widespread, scalable deployment. In comparison, DistilBERT achieved an accuracy of 0.94, with both precision and recall scoring 0.94. While not the quickest, its prediction times were reasonably efficient, with a median of 0.85 seconds per prediction and significantly higher memory usage of 502MB compared to MobileBERT. DeBERTa-base, another SLM, delivered lower precision and recall, also fell short in time efficiency, and had a higher memory usage of 1,341MB. FastText, though more compact with a memory usage of 201MB, suffered from reduced accuracy (0.62) and had a median prediction time of 1.43 seconds. Meanwhile, TinyBERT, the smallest model tested, offered quicker results at 0.78 seconds but at the cost of lower performance metrics, requiring 495MB of memory. All SLMs could provide inference over CPUs, making them suitable for deployment on various platforms. Also, due to the unavailability of commercial LLMs such as ChatGPT and Claude as local implementations, fine-tuning and testing on these models would incur significant costs for training and, especially for inference, given the millions of URLs an anti-phishing tool needs to process daily. However, we compare ChatGPT (3.5 Turbo) and GPT 4 with our model and other ML-based phishing detection tools in the next section. Thus, considering all the tested language models, based on a combination of high precision and recall, fast prediction times, and efficient memory usage, MobileBERT was the most suitable choice for our framework and was thus chosen as the language model used for our PhishLang framework.

C. Comparison with other ML models

We compared the performance of PhishLang (MobileBERT) with other popular open-source ML-based phishing detection models. Two of these models rely on the visual features of the website: VisualPhishNet [5] and PhishIntention [49], one relies on both the URL string and HTML representation of the website: StackModel [45], and one that relies on the semantic representation of the URL string only: URLNet [42]. Each of these models was tested on 5k websites: 2.5k randomly selected phishing samples from the PhishPedia dataset and 2.5k randomly sampled benign websites from the same dataset. We also include the commercially available models of ChatGPT (GPT 3.5T and GPT 4) in this comparison. Also, since GPT 3.5T and 4 are general LLM models that can predict any text content, we pass both the full HTML and the parsed versions as artifacts for evaluation.

Prior work on phishing detection frequently reports only the inference time of the proposed models, omitting the often substantial overhead associated with data acquisition (i.e., capturing the website artifacts) and preprocessing, especially in the case of multi-modal models. For instance, Liu et al. [49] report an inference time of up to 0.7 seconds for their PhishIntention framework, without accounting for the significant time and memory space required to capture and store website screenshots and pre-processing them. However, in real-world deployments where phishing detectors may need to process hundreds of thousands of webpages per day, the total operational overhead is governed not only by inference time but by the entire end-to-end pipeline, i.e., the data collection, preprocessing, and ultimately inference. Additionally, the size of the input artifact (e.g., HTML source, URL string, DOM content, image) plays a critical role in memory and bandwidth efficiency. Thus, to provide a more comprehensive and practical evaluation, we report median artifact size as well as the median time required at each stage of the detection pipeline.

Table II illustrates the performance and resource requirements of each model in terms of detection quality, artifact size, and timing overhead across the different stages of the detection pipeline. We find that PhishLang delivers an effective balance of accuracy and efficiency, achieving an F1 score of 0.94 while requiring just 7KB of input and a total median processing time of 0.88 seconds. Its time is evenly distributed across data collection (0.24s), preprocessing (0.31s), and inference (0.39s), making it well-suited for low-latency, client-side deployment. PhishIntention on the other hand, despite achieving the highest F1 score (0.95), requires a substantially higher total time (median) of 9.9 seconds and a large artifact size of 348KB due to its reliance on full-page screenshots. While it's detection stage remains relatively quick (1.08s), most of its time is consumed in data collection (6.52s) and preprocessing (3.27s). Importantly, this total time is significantly higher than the inference only latency (0.7s) reported in its original paper [49]. Visual PhishNet follows a similar pattern. It shows moderate performance (F1 = 0.85) but a large artifact size (217KB) and total latency of 4.51s, again dominated by the data collection (2.51s) and preprocessing (1.38s) stages. This demonstrates that visual models, while performant, introduce hidden costs that must be factored into practical deployment scenarios. Meanwhile, StackModel and URLNet both prioritize lightweight text feature inputs but differ in trade-offs. StackModel, using both the URL and HTML DOM, requires only 1KB of input and completes detection in 1.58s, with most of the time spent in inference. URLNet, relying solely on the URL string, is the fastest (0.71s total) and its artifact size is also the smallest (0.2KB), but its detection performance is the weakest (F1 = 0.73) among the group.

Turning to the general-purpose LLMs, GPT-3.5 and GPT-4, despite not being fine-tuned for phishing detection, show surprisingly strong results. GPT-4, when provided with parsed HTML inputs, achieves an F1 score of 0.93, nearly on par with PhishLang, while having low computational overhead. That being said, these models are constrained by token limits: GPT-3.5 processed only 822 phishing and 1,031 benign samples using full HTML, while GPT-4 handled 1,469 phishing and

TABLE II: Comparison of ML-based approaches across multiple dimensions, including artifact size, performance metrics and median processing times for each stage in the detection pipeline.

Model	Artifact Size	Accuracy	Precision / Recall	F1	Data (s)	Preproc (s)	Detect (s)	Total (s)
PhishLang	7 KB	0.94	0.93 / 0.95	0.94	0.24	0.31	0.39	0.88
PhishIntention	348 KB	0.96	0.94 / 0.96	0.95	6.52	3.27	1.08	9.93
Visual PhishNet	217 KB	0.85	0.83 / 0.86	0.85	2.51	1.38	1.14	4.51
URLNet	0.2 KB	0.73	0.72 / 0.74	0.73	0.14	0.19	0.45	0.71
StackModel	1 KB	0.83	0.85 / 0.81	0.82	0.32	0.44	0.97	1.58
GPT-3.5T (HTML)*	42 KB	0.87	0.88 / 0.86	0.87	0.37	0.79	1.42	2.30
GPT-4 (HTML)*	42 KB	0.92	0.93 / 0.91	0.92	0.55	0.85	1.81	2.85
GPT-3.5T (Parsed)*	7 KB	0.90	0.90 / 0.88	0.88	0.28	0.37	0.72	1.22
GPT-4 (Parsed)*	7 KB	0.94	0.93 / 0.93	0.93	0.32	0.58	0.51	1.26

2,147 benign. Additionally, their high computational costs, API dependencies, and commercial restrictions make them less suitable for scalable or real-time deployments without substantial adaptation. Overall, these results not only underscore the detection and computational efficiency of PhishLang, but also the need to evaluate phishing detection models holistically, not just by accuracy, but by artifact size, latency, and scalability.

III. Real-time framework

We implement our model as a framework that continuously identifies new phishing websites and reports them to various antiphishing entities, including browser protection tools, commercial blocklists, scanners, and hosting providers.

A. Identifying new URLs

We run our model on Certstream [14], a Certificate Transparency Log that streams newly registered SSL-certified URLs. This platform is extensively utilized to detect new phishing URLs [49], [72]. As of 2024, over 80% of phishing sites now use SSL certification [40]. To our knowledge, there exists no open-source real-time feed for non-SSL URLs. Moreover, since PhishLang's detection is based on the content of the website, rather than the URL, the use of SSL URLs only in the evaluation will not impact PhishLang's performance against non-SSL URLs.

If a website is flagged as benign with low confidence (<0.5), the model investigates the first five links in the website's source to detect phishing attacks that might not be apparent on the landing page, such as hidden phishing elements or image-based links. We observed an average of 27 domains per second on Certstream, and despite occasional delays in processing due to network bottlenecks, the model provided predictions within a median time of 9 minutes. From September 28, 2023, to January 11, 2024, PhishLang scanned 42.7M domains (172M websites through links), flagging 25,796 as phishing (0.057%).

To manually verify the correctness of PhishLang's detections, we randomly sampled 2,500 URLs from the set of detected websites for manual evaluation by the two coders. A second motivation for this analysis was to assess whether PhishLang could effectively identify evasive phishing websites in addition to regular ones. Evasive phishing websites have recently become an increasingly prominent threat vector, as they are not only harder for end-users to identify [56] but are also known to bypass existing anti-phishing tools with a far higher frequency than regular attacks [57]. The coders used 18 distinct technical features derived from prior literature [56], [80], [9], [46], [85] to help their annotation. These features represent concrete indicators of evasive phishing behaviors, and the coders used them to classify each phishing website into one of four evasion categories: JavaScript (JS) evasions, Clickjacking, DOM manipulations, and Text encoding attacks. Appendix A outlines the structural and behavioral indicators derived from prior literature.

TABLE III: Performance of PhishLang against evasive attacks

Attack Type	Samples	Accuracy	Precision	Recall
Regular attacks	1,623	94.2%	94.5%	95.7%
Behaviour based JS	280	92.9%	93.3%	92.6%
Clickjacking	313	91.3%	92.1%	90.5%
DOM Manipulations	94	88.4%	89.0%	87.1%
Text encoding	78	90.7%	91.2%	89.9%

For each sampled website, the coders first verified whether it impersonated any of the 409 known phishing target brands listed by OpenPhish as of August 2022 [59]. They then performed static analysis on both the website's HTML/JavaScript source code and its visual screenshot to identify matches to the 18 evasive features. The manual analysis confirmed that 2,388 of the 2,500 evaluated websites were indeed phishing, resulting in a precision of 95.5% for PhishLang. Among these true positives, the coders identified 1,623 regular phishing attacks, 280 JS evasions, 313 Clickjacking attacks, 94 DOM manipulations, and 78 Text encoding attacks. PhishLang's performance across these categories is summarized in Table III. Disagreements between coders were resolved through collaborative review of the underlying code until consensus was reached. It is worth noting that benign websites may occasionally exhibit isolated features from this list; therefore, the coders had to agree on threshold criteria, such as requiring multiple co-occurring signals or contextual indicators to avoid misclassification. Establishing these thresholds was a key part of the annotation guideline refinement and the coder discussion process. The resulting inter-rater reliability was 0.82 (Cohen's Kappa), indicating strong agreement. Due to space constraints, we dedicate Table X in Appendix A to provide detailed technical descriptions of all the features used.

To extend this categorization to the remaining 23,296 detected phishing websites, we implemented a rule-based scanner that applied the same evasive feature set on the website source code. We discuss this detector in detail in Appendix A. Overall,

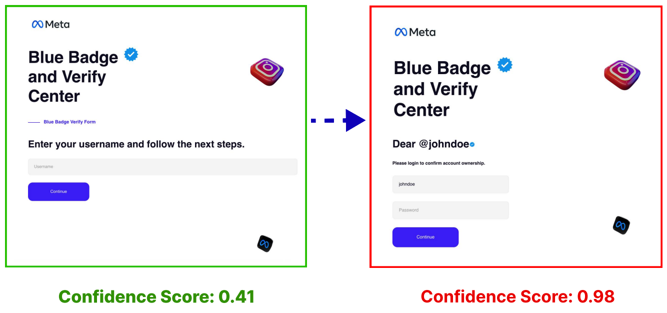

Meta
Blue Badge and Verify Center
Blue Badge Verify Form
Enter your username and follow the next steps.
Username
Continue
Meta
Blue Badge and Verify Center
Dear @johndoe
Please login to confirm account ownership.
johndoe
Password
Continue
Confidence Score: 0.41
Confidence Score: 0.98

Fig. 3: Example of an Instagram attack where the seemingly benign landing page (which was missed by PhishLang) led to a phishing page.

we found 17,396 (74.7%) regular, 3,159 JS evasion (13.6%), 3,349 Clickjacking (14.4%), 1,057 (4.5%) DOM manipulation, and 835 (3.6%) Text encoding cases.

B. *Evaluating misdetections*

Now, we evaluate the misdetections (False Positives and False Negatives) that were identified in the previous paragraph from our manual evaluation of the random sample of 2,500 URLs that were detected by PhishLang in real-time.

C. *False Positives*

Upon evaluating the 112 false positives, we identified 42 malformed websites with poorly structured text and input fields soliciting sensitive information. For example, Figure 5 shows a website with invalid characters and poorly designed input fields for personal information like name, email, and phone number, lacking clear instructions or context. These issues likely caused the site to resemble phishing samples, leading to incorrect classification by PhishLang. Additionally, 23 websites featured multiple empty links and malformed meta tags with several redirections. Figure 10 in Appendix C illustrates an example where poorly rendered input fields and empty links indicated an incomplete population of the website. Addressing these false positives without compromising PhishLang's detection features is challenging. We also discovered 14 websites with simple login fields that did not transmit data and contained no other information. These sites were classified as false positives because they neither imitated any organization nor had a deceptive motive, despite fitting traditional phishing profiles. Our crawler detects websites as soon as they appear on Certstream, so attackers could later add more details to these sites.

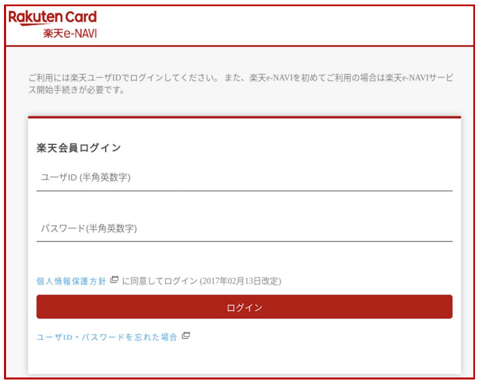
We found 21 websites with domain parking pages, likely detected due to their lack of substantial content, presence of multiple advertising links, and frequent redirections. Figure 9 in Appendix C shows one such example. To assess PhishLang's robustness against parked pages, we used keywords typical of parked pages to select 1,000 parked websites from our total sample of 42.7 million domains. PhishLang flagged only 5 of them as phishing (median confidence of 0.27).

D. *False Negatives:*

We assessed all URLs processed by PhishLang (42.7 million domains) using VirusTotal. We identified 261 websites missed by PhishLang but flagged as phishing by VirusTotal (i.e., having two or more detections, a threshold established in prior literature for identifying true positive phishing samples [61]) and confirmed them as true positives through manual inspection. Manually evaluating these websites, we found 82 had CAPTCHAs and 36 had QR codes, both common phishing evasion tactics [86], [38]. However, once the URLs were manually extracted from the QR codes and provided to PhishLang, it was able to detect all of these attempts. The same was also true for all except two CAPTCHA-protected sites. Thus, to address these issues, we have implemented the *pyzbar*[36] library to let PhishLang automatically scan QR codes. However, solving CAPTCHAs programmatically can violate many websites' terms of service [76], so no modifications were added to address this issue. We also found 25 sites that mimicked legitimate brands but initially only had input fields without malicious behavior. PhishLang's confidence was low on these landing pages, resulting in false negatives, but it detected phishing on the subsequent pages. Figure 3 shows an Instagram phishing page where PhishLang's confidence increased from 0.41 on the first page to 0.98 on the second. To address this, our framework now inputs placeholder data on benign-marked sites, submits it, and reassesses the destination, which helps detect all previously missed sites. Additionally, 118 undetected websites were in non-English languages: Japanese (57), Spanish (36), German (18), and Italian (7). Figure 4 shows an example in Japanese. Using *langdetect*, we found our training data was predominantly in English (n=18,683) with minimal coverage of these languages (Japanese=500, German=660, Spanish=403, and Italian=85). This lack of diversity likely caused misclassifications. While most phishing websites are written in English, non-English phishing scams are an emerging threat [20]. Thus, to address this issue, we added the Argos Translate library [8], a fully offline translation tool supporting English translation for 32 languages, to the framework. For each URL, PhishLang now applies Langdetect to determine if the content is non-English, and if so, the content is translated using Argos and PhishLang reattempts the detection. These additions add minimal overhead to the pipeline, with checking if the content is non-English taking a median runtime of just 0.3 seconds, and translation of non-English content using Argos taking a median time of 0.5 seconds and 270MB more memory. This enhancement led to PhishLang correctly identifying 115 out of the 118 previously missed non-English phishing websites. Figure 11 in Appendix C shows how the example previously shown in Figure 4, which initially went undetected, was translated into English using Argos, and then detected by PhishLang.

Implementing the adversarial patches from Section IV and aforementioned modifications significantly improved performance against these false negatives. Only 2 out of 36 QR-code attacks, 5 out of 25 multi-page attacks, and 7 true-positive detections were misclassified as benign, despite a median confidence score of 0.44.

Lastly, we noted 12 websites lacking noticeable phishing



Rakuten Card
楽天e-NAVI

ご利用には楽天ユーザIDでログインしてください。 また、楽天e-NAVIを初めてご利用の場合は楽天e-NAVIサービス開始手続きが必要です。

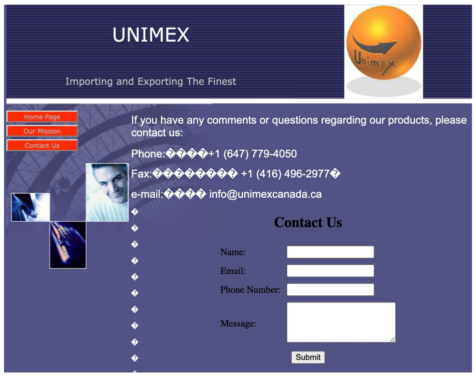
楽天会員ログイン

ユーザID (半角英数字)

パスワード(半角英数字)

個人情報保護方針 に同意してログイン (2017年02月13日改定)

Fig. 4: Example of a non-English (Japanese) false negative sample



UNIMEX

Importing and Exporting The Finest

If you have any comments or questions regarding our products, please contact us:

Phone:����+1 (647) 779-4050

Fax:�������� +1 (416) 496-2977�

e-mail:���� info@unimexcanada.ca

Contact Us

Name:

Email:

Phone Number:

Message:

Fig. 5: Example of a poorly designed (benign)website that was detected as a false positive

artifacts, with low confidence scores (Median=0.52). The small number of such samples might not significantly impact the training outcome if included in the dataset. We plan to retrain the model after gathering enough samples matching this criterion. After implementing the adversarial patches discussed in Section IV, PhishLang correctly identified 28 of the websites with poor structural design as benign. Out of the 2.5k manually observed true positives, no new samples were changed from phishing to benign, increasing the true positive rate of our model (on the initial dataset) to 96.6%.

E. Comparing detection with commercial Phishing blocklists

In this section, we evaluate the effectiveness of four popular anti-phishing blocklists against websites identified by PhishLang. These include Google Safe Browsing [1], Microsoft SmartScreen [51], PhishTank [2] and OpenPhish [58]. We also include the domain providers, i.e, the registrars or infrastructure providers (e.g., Namecheap, GoDaddy, Cloudflare) that the websites are hosted on, in this analysis, as they are often the only entities with the authority to directly take down malicious sites [57].

We evaluate the extent to which phishing websites flagged by PhishLang are covered by existing tools and blocklists, and whether our reporting leads to their subsequent detection. Upon identifying a new phishing site, PhishLang immediately queries the four blocklists in real time to determine if the URL is already flagged. If the site is not listed, it is automatically reported. We then track its detection status at 10-minute intervals for up to seven days to monitor if and when it is added to the respective blocklists. Considering ethical implications, we report all 26k identified websites immediately (if it was not already detected) to ensure user protection, despite our 97% accuracy rate suggesting a potential 3% false positives being reported. This trade-off is acceptable given the typically higher noise in blocklist data [72], [70], [57]. . To monitor domain-level takedowns, we used the Python requests library and Selenium [23] to periodically access each phishing URL every 10 minutes for up to seven days. We recorded whether the website returned an HTTP error code, failed to load, or displayed a generic parking page - any of which would indicate that the hosting provider had suspended, removed, or deactivated the site. Table IV illustrates the initial coverage of these tools and subsequent coverage after we had reported the URLs that they had failed to detect. All blocklists initially have low coverage, with a significant increase in detection rates post our reporting, especially for evasive threats. However, it is evident that the blocklists still take substantial time to identify these threats. For instance, Google Safe Browsing's detection of JS evasion attacks increased from 35% to 73%, and SmartScreen's detection of DOM-based attacks rose from 18% to 86%. Regular phishing URL detection also improved significantly. On the other hand, PhishTank and OpenPhish have different responses to reported URLs. Despite these improvements, it is important to note that these blocklists generally maintain high detection rates and low false positives [57]. A similar trend was also seen for hosting domains, whose rate of removal significantly improved after our reporting. Our model thus helps close detection gaps, particularly for new and evasive attacks.

IV. Adversarial impact

The effectiveness of Machine Learning (ML) and Deep Learning (DL) models can be significantly reduced by adversarial inputs designed to lower the model's confidence score, which can also lead to misclassification. These inputs exploit inherent vulnerabilities related to the training data, architectural design, or developmental assumptions of the model. For example, in the case of Machine-Learning phishing webpage detectors (ML-PWD), Panum et al. [60] showed how image-based phishing detection models are vulnerable to FSGM pixel-level exploitation, where adversarial websites are modified to appear like benign ones. AlEroud et al. [22] explores how models relying on URL-based features can be exploited by employing perturbations in the URI structure. Both image-based features and URL-based attributes are irrelevant to our model, which solely relies on a collection of tags extracted directly from the HTML source code. Consequently, the only avenue for an attacker to exploit our model is to target the specific HTML tags we analyze and alter their associated text.

While previous literature has investigated how attackers can modify attributes in the "Feature space" [75], [11], [19], [44] (i.e., when the website is processed into feature vectors specific to those that are used by the ML-PWD for prediction), recent studies have focused more on attackers introducing perturbations in the "problem space" (i.e., where the attacker directly modifies the webpage [46], [12], [5]). This shift is due to real-world attackers operating in the "problem space" [63], where

TABLE IV: Performance of browser protection tools, blocklists and hosting domains

Attack Type	# URLs	Safe Browsing		SmartScreen		PhishTank		OpenPhish		Hosting Domain	
		Detection (%)	Time	Detection (%)	Time	Detection (%)	Time	Detection (%)	Time	Detection (%)	Time
Regular Attack	1,595	50.92 (84.33)	3.71 hrs	34.26 (74.26)	5.85 hrs	14.8 (36.9)	7.33 hrs	22.6% (47.1)	3.81 hrs	39.2 (61.0)	4.50 hrs
JS Evasion	306	35.27 (70.36)	5.67 hrs	27.48 (75.42)	9.66 hrs	3.9 (19.4)	11.19 hrs	8.5% (27.9)	5.77 hrs	6.4% (19.4)	2.15 hrs
ReCAPTCHA/QR	91	40.97 (66.07)	3.75 hrs	15.31 (81.34)	7.54 hrs	9.9 (29.7)	8.02 hrs	16.5% (37.1)	4.05 hrs	9.9% (29.7)	10.22 hrs
Clickjacking	325	61.32 (80.28)	2.42 hrs	38.48 (77.65)	5.3 hrs	7.7 (26.8)	14.70 hrs	15.7% (36.4)	7.19 hrs	7.1% (26.8)	5.73 hrs
DOM Attacks	102	55.79 (74.39)	3.48 hrs	18.27 (85.36)	5.56 hrs	10.8 (31.0)	11.15 hrs	20.6% (40.5)	13.71 hrs	10.8% (31.0)	7.19 hrs
Text encoding	81	27.97 (72.57)	6.45 hrs	20.79 (70.36)	2.5 hrs	8.6 (28.0)	6.42 hrs	16.0% (36.7)	2.88 hrs	8.6% (28.0)	5.13 hrs

the perturbations introduced are subject to certain physical constraints. If these constraints are not met, which can happen when perturbations are added in the feature space, there is an increased risk of generating adversarial samples that, while potentially reducing the model's efficiency[82], may result in samples that are physically unrealizable.

In the context of phishing websites, this often manifests as notable artifacts in the webpage layout, as demonstrated by Yuan [92] and Draganovic et al. [25]. These artifacts make the sites suspicious to users, reducing the attacks' overall effectiveness. Moreover, to perturb the feature space, the attacker is required to have internal access to the processing pipeline of the ML-PWD - a process that can be challenging and costly [7]. The success rate of phishing, even when undetected by tools, depends on user interaction. Additionally, most adversarial implementations in the literature target specific ML-PWDs, but realistically, attackers do not target specific models. Instead, they prefer generalizable evasive attacks, that manipulate raw data in the problem space, making the attack evasive across different detection systems. Therefore, we focus on testing and improving the adversarial robustness of PhishLang by emphasizing on the problem-space attacks. These attacks are designed to introduce perturbations into the website's source code while maintaining the website's layout.

A. Evaluating Problem-space evasions

Apruzzese [90] and Yuan et al. [91] have conceptualized evasive attacks that are relevant and generalizable for creating phishing samples capable of evading the four most popular ML-PWDs. They identified 57 features that attackers commonly use to bypass ML-PWD systems. Building on this framework, Montaruli et al. [53] developed a novel set of 16 evasive attacks focusing on the problem space of adversarial attacks. These attacks involve directly modifying the HTML content using fine-grained manipulations, allowing for changes to the HTML code of phishing webpages without compromising their maliciousness or visual appearance. This approach successfully preserves both the functionality and appearance of the websites. Some examples of these attacks include *InjectIntElem*, which involves injecting a specified number of (hidden) internal HTML elements (e.g., $< a >$ tags with internal links) into the body of the webpage and *ObfuscateJS*, where the JavaScript code within $< script >$ elements of a webpage is obfuscated by encoding them into Base64 and then inserting a new script to decode and execute the original script. A full description of these attacks is provided in the author's paper. For brevity, we do not go into comprehensive details about which feature (based on Apruzzese [90] and Yuan et al.[91]) these attacks target.

Since different phishing websites can be compromised by different attacks, Montaruli et al also developed a query-efficient black-box optimization algorithm based on WAF-A-MoLE [21]. This algorithm employs an iterative methodology that involves successive rounds of mutation to modify the original malicious sample, aiming to reduce the confidence score provided by ML-PWD. Attacks are carried out as Single Round (SR) or Multi Round (MR) manipulations. SR manipulations, such as UpdateHiddenDivs, UpdateHiddenButtons, UpdateHiddenInputs, and UpdateTitle, are applied once to achieve their goal of hiding or modifying specific HTML elements and do not require further changes to evade detection. In contrast, MR manipulations, including InjectIntElem and InjectExtElem, require multiple sequential applications to progressively adjust the internal-to-external link ratios and other features to effectively evade detection. The optimizer begins by initializing the best adversarial example and score with the initial phishing webpage and its corresponding score. It then sequentially applies the SR manipulations, updating the best example and score whenever a new manipulation achieves a lower score. Afterward, the optimizer enters the MR manipulations loop, which includes a specified number of mutation rounds. In each mutation round, the algorithm generates new candidate adversarial phishing webpages from the current best example by applying one MR manipulation to each candidate. The candidate with the lowest confidence score is selected, and if this score is lower than the current best, it becomes the new best adversarial example. The number of mutation rounds, R, given the maximum query budget Q, is determined by the formula $R = \frac{Q-\#SR}{\#MR}$, where $\#SR$ and $\#MR$ represent the number of SR and MR manipulations, respectively. One attack *InjectFakeFavicon* (an SR manipulation) is not relevant to PhishLang since our parser does not look for favicons in `<head>` tags. We set a query budget of 35 (instead of 36 as in the original paper due to the omission of one SR manipulation) and injected the attacks into each sample in our test set (6,725 URLs) to choose the best adversarial sample. In addition to adversarially testing PhishLang, our goal is to introduce patches that can reduce or nullify the effectiveness of these attacks. While both SR and MR manipulations cumulatively work towards creating the best-performing adversarial sample, it is essential to identify which particular attack had the most impact. This involves determining which manipulation was instrumental in significantly reducing the confidence of PhishLang when evaluating the sample. This specific manipulation can be considered the *primary attack* that was crucial in achieving the desired adversarial effect. To identify this, after each SR and MR round, we compute the difference in the *adversarial advantage* for the sample during the current stage compared to the previous stage. The adversarial advantage, A, is computed based on the drop in the confidence score of the model towards the phishing website during subsequent optimization stages. Therefore, if C_o is the original confidence

TABLE V: Impact of different adversarial attacks. Text in parentheses indicates values post adversarial patches were applied.

Attack	Median advantage	Incorrectly predicted
InjectIntElem (A1)	0.204 (0)	7.99% (0)
InjectIntElemFoot (A2)	0.142 (0)	11.24% (0)
InjectIntLinkElem (A3)	0.231 (0)	4.43% (0)
InjectExtElem (A4)	0.285 (0)	7.02% (0)
InjectExtElemFoot (A5)	0.194 (0)	15.01% (0)
UpdateForm (A6)	0.271 (0)	4.90% (0)
ObfuscateExtLinks (A7)	0.314 (0.039)	19.01% (1.68%)
ObfuscateJS (A8)	0.348 (0.028)	20.92% (1.51%)
InjectFakeCopyright (A9)	0.201 (0)	4.10% (0)
UpdateIntAnchors (A10)	0.245 (0)	7.92% (0)
UpdateHiddenDivs (A11)	0.173 (0)	8.01% (0)
UpdateHiddenButtons (A12)	0.241 (0.025)	17.20% (1.07%)
UpdateHiddenInputs (A13)	0.403 (0.027)	13.59% (1.18%)
UpdateTitle (A14)	0.107 (0.005)	11.27% (0.63%)
UpdateIFrames (A15)	0.111 (0)	5.42% (0)

score of the model in identifying the website as phishing and C_{p_1} is the confidence score after the first perturbation round (SR as per the algorithm), then we define A as $A = C_o - C_{p_1}$.

By analyzing these differences, we can identify which manipulations had the most significant impact on reducing the model's confidence, thereby determining the *primary attack* that achieved the desired adversarial effect. If two or more attacks demonstrate similar adversarial advantages (within one standard deviation of the mean adversarial advantage across all samples), we designate these as primary attacks **[Scenario 1]**. Additionally, an attack following a primary attack may negatively impact the model's performance, but the preceding primary attack might have lessened its effectiveness. Therefore, once a primary attack is identified, we remove it from the sample and rerun the optimization algorithm. If another attack then exhibits an adversarial advantage similar to the previously recognized primary attack (within one standard deviation of the adversarial advantage across all samples), it is also designated as a primary attack **[Scenario 2]**. In our dataset, we noticed that only 11.3% of the samples met **[Scenario 1]** and 4.5% (samples met **[Scenario 2]**, indicating that for most phishing websites, only one attack had a significant impact in adversarially compromising it. Table V presents the performance of each *adversarial attack* on PhishLang, detailing the number of incorrect predictions (false negatives) and the corresponding median adversarial advantages. We observe that all problem-space-facing attacks were able to decrease the efficiency of our vanilla model initially, which was then significantly improved by applying the patches. For example, the ObfuscateJS (A8) attack had a median adversarial advantage of 0.348, which dropped to 0.028 after applying adversarial patches, and the incorrect prediction rate reduced from 20.92% to 1.51%. Similarly, the UpdateHiddenButtons (A12) attack showed a median advantage of 0.241 initially, which was reduced to 0.025 post-patch application, with incorrect predictions decreasing from 17.20% to 1.07%. We discuss the design and implementation of these adversarial patches in the next section.

B. Mitigating Adversarial Attacks

To mitigate or nullify the effectiveness of the adversarial attacks, we employ one or both of the following strategies: **a) Parser Modification -** Here, we identify that the attack can be sufficiently neutralized by modifying how our framework parses the raw HTML source code, eliminating the need for adversarial retraining, and **b) Adversarial Training -** We retrain our model using adversarial samples in cases where the respective attack is identified as the *primary attack*.

Parser modifications: Attacks InjectIntElem (A1), InjectIntElemFooter (A2), InjectIntLinkElem (A3), InjectExtElem (A4), and InjectExtElemFooter (A5) use one of four strategies to create the adversarial samples: S1, S2, S3, and S4. S1 inserts the hidden attribute into an HTML element to prevent the browser from rendering its content, adopting the hidden attribute as its default approach. S2 modifies the style attribute to "display:none" to hide elements. S3 uses the $< style >$ HTML element to achieve the same effect as S2 but through the $< style >$ tag. S4 places HTML elements to be hidden inside the $< noscript >$ tag, effective only if JavaScript is enabled in the victim's web browser. To counter these attacks, we propose two parser-based modifications. **Patch 1** ignores any tags with a hidden attribute or "display: none" in the style attribute, which reduces the effectiveness of attacks applied using S1 and S2 to 0. For tags that refer to a CSS element, the parser searches the corresponding CSS $< style >$ sheet for "display:none" attributes. If found, the initial tag is omitted from being included in the parsed representation given to the model. This constitutes **Patch 2**, which reduces the effectiveness of S3 to 0. We did not need to implement any protection for S4, as our parser does not consider $< noscript >$ elements. Patches 1 and 2 are also transferable to Attacks InjectFakeCopyright (A9), UpdateHiddenDivs (A11), and UpdateIFrames (A15), which use similar syntax to add or modify hidden elements in the source code. On the other hand, attack UpdateForm (A6) manipulates forms on a webpage by replacing the original internal link specified in the action attribute with a different internal link that does not trigger detection features, such as "#!" or "#none." To counter this attack, we propose **Patch 3**, a parsing modification that checks if the 'action' attribute directs to an internal section or is a valid external link (using Python's request library [28]) that nullified the attack effectiveness to 0. Finally, ObfuscateExtLinks (A7) obfuscates the external links in a webpage to evade multiple HTML features by substituting the external link with a random internal one that is not detected as suspicious by the HTML features (e.g., #!) and creates a new $< script >$ element that updates the value of the action attribute to the original external link when the page is loaded. While similar to A6 in terms of manipulating links to avoid detection, A7 specifically targets external links and employs JavaScript to dynamically revert the obfuscation. Regardless, Patch 3 successfully nullifies this attack and UpdateIntAnchors (A10).

Adversarial retraining: Attack ObfuscateJS (A8) obfuscates JavaScript within $< script >$ elements to evade detection features such as `HTML_popUP`, `HTML_rightClick`, and `HTML_statBar`. This technique involves extracting the original JavaScript, encoding it in Base64, and replacing the original $< script >$ content with new code that decodes and executes the obfuscated JavaScript. To counter this attack, we developed two patches: **Patch 4.1** converts Base64 (or any other non-UTF-8 encodings) to UTF-8 before parsing, and **Patch 4.2** retrains the model with adversarial samples where A7 was the primary attack. Although these patches did

TABLE VI: Impact of adversarial attacks on phishing attacks by category (Test set)

Attack Type	Before intervention		After intervention	
	Median Advantage	Incorrectly Predicted	Median Advantage	Incorrectly Predicted
Regular attacks (5,734)	0.212	648	0.030	37
Behaviour based JS (198)	0.425	73	0.246	29
Clickjacking (485)	0.230	91	0.033	14
DOM Manipulations (101)	0.380	45	0.021	3
Text encoding (207)	0.316	42	0.063	8

TABLE VII: Performance of PhishLang against evasive attacks after adversarial patching. Values in parenthesis denote % increase

Attack Type	Samples	Accuracy	Precision	Recall
Regular Attacks	1,623	97.0% (2.8%)	96.0% (1.5%)	98.3% (2.6%)
Behavior-based JS	280	94.2% (1.3%)	95.6% (2.3%)	94.1% (1.5%)
Clickjacking	313	93.5% (2.2%)	95.7% (3.6%)	94.0% (3.5%)
DOM Manipulations	94	94.1% (5.7%)	94.8% (5.8%)	93.5% (6.4%)
Text Encoding	78	95.5% (4.8%)	96.4% (5.2%)	95.1% (5.2%)

not completely nullify the attack, they significantly reduced the median adversarial advantage to 0.039 (from 0.314) and the false negative rate to 1.68% (from 19.01%). Attack UpdateHiddenButtons (A12) obfuscates all disabled button elements by removing the `disabled` attribute and inserting a new `<script>` element that re-applies this attribute during rendering using the `setAttribute()` DOM method. This approach ensures that both the rendering and original behavior of the buttons are preserved. While a potential countermeasure could involve ignoring scripts with the `disabled` attribute, this could break the user experience. For example, in a web form where the "Submit" button is initially disabled and only enabled after all required fields are filled out, ignoring such scripts could prevent proper functionality. Therefore, to address this attack, we retrained our model with adversarial samples where A12 was the primary attack (**Patch 5**). This reduced the median adversarial advantage to 0.025 (from 0.241) and the false negative rate to 1.07% (from 17.20%). Attack UpdateHiddenInputs (A13) evades detection of hidden and disabled input elements by changing the `type` attribute from "hidden" to "text" and adding the `hidden` attribute. We adversarially trained with samples that had this attack as the primary focus (**Patch 6**), which reduced the median adversarial advantage to 0.0.27 (from 0.403) and the false negative rate to 1.18% (from 13.59%). For Attack UpdateTitle (A14), we use a variant of Patch 6 that prevents JS from executing when reading the $< title >$ tag.

C. Advesarial impact on phishing categories

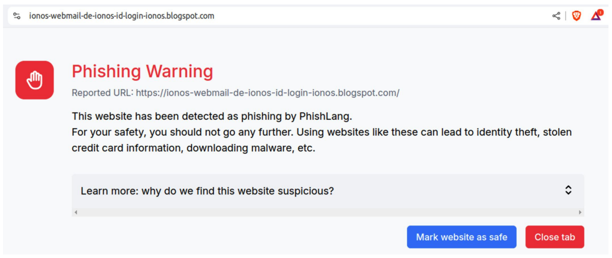
We also examined the impact of adversarial attacks on our test set samples based on different attack categories. Table VI shows the median adversarial impact and the number of incorrectly predicted samples for the five phishing attack categories (Regular, BJE, Clickjacking, DOM Manipulations, and Text Encoding) both before and after implementing interventions (such as parser modifications or adversarial training). We found that after applying adversarial patches, the True Positive Rate (TPR) for all attack categories, except BJE, improved beyond their initial training accuracies. For instance, regular phishing attacks achieved a TPR of 99.4%, while DOM manipulations reached 98%.

Despite the application of adversarial patches, websites with Behavioral JavaScript Evasion (BJE) continued to have a comparatively lower TPR (85.9%) than other attack categories. Upon closer inspection, we discovered that a significant portion of these samples were affected by the ObfuscateJS (A8) attack. Given the low distribution of BJE samples in our ground truth dataset (n=715), we decided to collect more samples for this attack category. From July 11th to 29th, 2024, we gathered the source code for 17,305 new websites from PhishTank which had been verified as phishing, and then identified 411 new BJE samples. These samples were then adversarially augmented using the optimization algorithm detailed in Section IV. Both the original and adversarial samples were added to the ground truth and used for retraining. Following the retraining, we observed a significant reduction in the adversarial advantage of the BJE samples, both with and without the patch. Without the patch, the median adversarial advantage decreased to 0.084, and after applying Patch 5, it further reduced to 0.039, achieving a TPR rate of 97.2%. In the next section, we identify whether the performance boost due to adversarial training transfers to real-world websites outside the ground-truth dataset.

We also re-evaluated our adversarially patched model on the metadata of the 42.7M domains from Certstream that were collected in Section III-A. We find that 1,778 more websites were flagged as phishing, a noticeable increase of 6.88% compared to when the initial model was run (25,796 URLs to 27,574 URLs). Breaking down these numbers per category: The model detected 18,549 regular (6.6% increase), 3,391 JavaScript evasion (7.3% increase), 3,583 Clickjacking (7.0% increase), 1,147 DOM (8.5% increase), and 904 Text encoding attacks (8.3% increase). However, it is more important to evaluate the false-positive and false-negative rates of these samples. We pseudo-randomly selected the same number of URLs (2.5k) from the detected set, with the same per category distribution, with 1,632 regular attacks, 280 BJEs, 313 clickjacking, 94 DOM Manipulations, and 78 text encoding. Similar to the evaluation of the initial model, we also randomly selected 2.5k benign websites. Table VII illustrates the performance of the model against different categories of attacks. We see a noticeable boost across all attack categories, for example, for Behavior-based JS evasion attacks, the model now achieves an accuracy of 94.8% (an increase of 1.9%), whereas for Text Encoding attacks, the accuracy increased to 95.7% (an increase of 3.2%). This indicates that the boost in detection for attack categories transferred from the test set (as seen in the previous section) to real-world websites.

V. Client-side application

Building on the PhishLang framework detailed throughout this paper, we now discuss the design, functionality, and evaluation of our client-side application. This application integrates a Chromium-based browser extension with a background process to block phishing websites without relying on online blocklists. Once installed, the application automatically deploys our trained MobileBERT-based model as a local server, which the browser extension utilizes to analyze all visited URLs. This setup is compatible with Microsoft Windows 10 and later, as well as Debian-based Linux distributions like Ubuntu, with MacOS support planned for future releases. The


ionos-webmail-de-ionos-id-login-ionos.blogspot.com

Phishing Warning

Reported URL: https://ionos-webmail-de-ionos-id-login-ionos.blogspot.com/

This website has been detected as phishing by PhishLang.
For your safety, you should not go any further. Using websites like these can lead to identity theft, stolen credit card information, downloading malware, etc.

Learn more: why do we find this website suspicious?

Fig. 6: PhishLang client-side extension running in Brave Browser.

installation process seamlessly configures the model, initializes the local service, and installs the extension for supported Chromium-based browsers. Notably, the detection system operates entirely on the local machine, utilizing CPU DRAM to ensure privacy and independence from external servers. The deployed model in the client-side extension includes all adversarial patches and parser improvements introduced in Section IV, providing hardened, real-time protection against evasive phishing threats. Future adversarially retrained models are automatically pushed to the open-source release from our end, ensuring that users always benefit from the latest robustness enhancements without requiring manual updates.

A. *Evaluating the performance of our client-side extension*

We evaluate 2,000 verified phishing websites from PhishTank, comparing the performance of PhishLang on four distinct system configurations. The first configuration is our experimental setup, featuring an Intel Xeon W Processor with 184GB of RAM and 4x NVIDIA A5000 GPUs, running Ubuntu 22.04 LTS, with the model running natively (i.e., without the browser extension). The second configuration is a mid-tier setup, consisting of an Intel Core i5 13th Gen processor, 8GB of RAM, no GPU, and running Windows 11 Build 22631. The third configuration is a low-tier setup, using an Intel Celeron N4500 processor, 2GB of RAM, no GPU, and running Ubuntu 24.04 LTS. The final configuration involves a virtual machine using VMware Player emulated with 2 core, 4GB of RAM, no GPU acceleration, and running Windows 11 Build 22631. These various configurations were chosen to assess PhishLang's performance across a spectrum of hardware configurations that reflect typical end-user environments. Table VIII presents the performance of PhishLang in terms of efficiency, memory usage, and inference time. Our findings show that the client-side application running on both Ubuntu and Windows 11 achieves the same efficiency as the model running on our experimental server, indicating no performance loss when PhishLang operates on the client-side. Additionally, all configurations exhibit similar memory usage. Memory usage was calculated by considering the system service (local server) and the memory consumed by the browser extension. It is noteworthy that the experimental setup has lower RAM usage since it runs the base model without any browser extension overhead. Regarding inference time, the low-tier and virtual machine configurations, despite having weaker processors, demonstrate only a slight decrease in speed compared to the experimental and mid-tier setups. All configurations maintain a median inference time of under one second. This suggests that

TABLE VIII: Performance of the PhishLang app for various system configurations

Configuration	Precision/Recall	Memory usage			Inference time (in secs)		
		Min	Max	Median	Min	Max	Median
Experimental	0.95/ 0.96	28MB	215MB	76MB	0.08	1.09	0.44
Medium-end	0.95/ 0.96	51MB	308MB	93MB	0.13	1.47	0.59
Low-end	0.95/ 0.96	54MB	247MB	89MB	0.18	1.95	0.73
Virtual Machine	0.95/ 0.96	68MB	231MB	102MB	0.11	2.43	0.81

TABLE IX: Comparison of Zero-day protection by anti-phishing tools

Tool	**Regular**	**JS-Evasions**	**Click-jacking**	**DOM Manipul.**	**Text Encoding**	**Overall**
PhishLang	**94**	**97**	**94**	**87**	**85**	**457 (91.4%)**
GSB (Enhanced)	56	53	39	44	29	221 (44.2%)
GSB (Client-only)	38	5	11	9	9	77 (15.4%)
McAfee	63	69	46	31	18	227 (45.4%)
BitDefender	45	48	36	39	7	175 (35%)
Avast	84	52	29	9	16	192 (38.4%)
Trend-Micro	52	18	43	12	5	130 (26%)

our application offers rapid inference regardless of the system configuration. It is important to note that the reported inference time excludes the time required to fetch the website, as server congestion or intentional delays by attackers can significantly affect website loading times.

B. *Comparison with Commercial Anti-phishing tools*

We also compare the performance of PhishLang with five widely used commercial anti-phishing tools. Like PhishLang, these tools operate within the user's web browser, scanning links of visited websites and providing warnings if a website is identified as phishing. Among these, Google Safe Browsing serves as the default protection mechanism in Google Chrome, Mozilla Firefox, and Safari, and several other browsers, collectively covering nearly 96% of all internet users [78]. Beyond browser-integrated protections, many users choose specialized antivirus solutions that incorporate anti-phishing functionalities. For our evaluation, we selected four popular anti-phishing tools that have been certified by AV-Comparatives [10], an independent organization that tests and certifies security software. These tools include Avast Online Security, BitDefender TrafficLight, McAfee WebAdvisor, and TrendMicro Toolbar.

To ensure unbiased evaluation, we tested all tools against 500 previously unseen phishing domains that were created and hosted by us. We designed the layout of these websites based on the top 50 most frequently targeted brands, as identified in the January 2024 OpenPhish rankings [59], since these brands are more attractive targets for anti-phishing detectors aiming to maximize real-world impact. We then systematically augmented these base designs with the features for regular credential phishing, as well as the four evasive categories (See Table X in Appendix A), with each category consisting of 100 samples. To maintain ethical standards, all phishing functionalities were disarmed, ensuring any information entered on these sites was immediately discarded. The websites were also taken down immediately after completing our experiments. This approach aligns with methodologies used in prior research for simulating real-world phishing scenarios [56]. This methodology aligns with established best practices for safely simulating phishing scenarios in prior academic research [56].

We then deployed these domains using GoPhish [88], an open-source phishing framework that enabled secure hosting and fine-grained control over deployment, ensuring consistency across experiments.

C. Evaluation strategies

Approaches: We evaluate the performance of the tools using two evaluation strategies: *zero-day protection* and *over-time protection*. Zero-day protection measures the detection rate of each tool immediately after a phishing website is launched. This is crucial because anti-phishing tools often rely on both local and online models: the local model provides immediate detection, while the online model, typically more sophisticated and resource-intensive, queues the URL for analysis and subsequently pushes updates to all users once a threat is confirmed [57]. PhishLang, being primarily client-side, is *designed* to deliver zero-day protection. Since the risk of phishing attacks escalates rapidly with time [56], zero-day detection represents the ideal outcome for any anti-phishing tool. The second evaluation strategy, over-time protection, examines what percentage of websites are detected over a seven-day period. Anti-phishing tools often improve detection rates post-launch by leveraging larger models, intelligence from partner organizations, manual reviews, or more advanced analyses that take additional time [55], [62]. However, PhishLang is at a disadvantage in this scenario as it relies exclusively on client-side protection.

Tool specifications and setup: With PhishLang being a client-side-only tool, we aimed to test the client-side components of other tools as well. However, most tools integrate both client-side and online modules, with the latter being inseparable in practice, except for Google Safe Browsing (GSB). GSB's default Standard protection appears to rely solely on online blocklists for detection without incorporating a real-time component . It only sends a random sample of the user's browser history for analysis. In contrast, its Enhanced protection setting invokes the real-time module. To ensure we utilized only the client-side component, we implemented the strategy described by Pourmohamad et al. [64], isolating the client-side component for evaluation purposes. We included the GSB Enhanced mode as part of our testing to assess its real-time capabilities. Each tool was installed in Google Chrome within a separate VM running Windows 11 with 4GB of RAM. Using Selenium, we automated browser visits to each of the 500 websites recursively until detected by the respective tool. The first iteration, conducted immediately after the websites were launched, was treated as *the zero-day metric*. Subsequent iterations continuously monitored over-time protection.

D. Results

Table IX compares the zero-day protection capabilities of the anti-phishing tools. PhishLang demonstrates a substantial advantage, achieving a detection rate of 91.4% for *zero-day* phishing websites. In contrast, GSB-Enhanced detects slightly over 44% of such websites, while its client-only variant performs considerably worse, identifying only 15.4%. This aligns with the findings of Pourmohamad et al. [64], which report significantly low detection rates for the GSB client-only model. Among the other commercial tools, performance is relatively similar, with Trend Micro showing the lowest detection rate

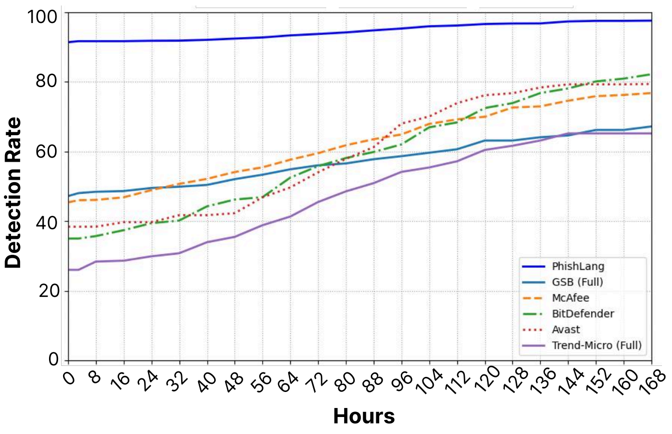


Fig. 7: Comparison of anti-phishing tools overtime

at just 26%. Notably, all tools, except for PhishLang, provide stronger protection against regular phishing attacks compared to evasive ones. This highlights a significant vulnerability for users relying on these tools against phishing threats at zero day. Figure 7 illustrates the performance of the tools over seven days, measured at 8-hour intervals. The results show a steady improvement in detection rates across all tools, starting from their Day Zero findings and gradually increasing over time. Avast, BitDefender, and McAfee demonstrated notable gains, achieving detection rates of 79.4%, 82.2%, and 76.8%, respectively. Similarly, GSB and Trend Micro followed a comparable trend, reaching 67.2% and 65.2%, with most of their improvements concentrated in the Regular and Clickjacking categories. However, these tools showed limited progress in detecting other evasive threats. Notably, their final detection rates remained *lower than PhishLang's Day Zero performance.* It is important to highlight that anti-phishing tools are specifically designed to maintain very low false positive rates to avoid blocking legitimate websites [62], [57]. In Section III-B, we established PhishLang's similarly low false positive rate on a live stream of Certstream URLs. While these experiments demonstrate PhishLang's performs better than other anti-phishing tools, some limitations of the study must be acknowledged. In real-world scenarios, phishing URLs are often distributed multiple times across emails, social media platforms, and other channels, significantly increasing their exposure to anti-phishing systems. This heightened visibility enhances their likelihood of detection - a condition our URLs were not subjected to. In our setup, the only visibility these URLs received was through repeated queries from our individual browsers during the longitudinal experiment, however, they still appeared to originate from the same subnet, potentially resembling traffic from a single isolated location. However, this does not undermine the fact that phishing attacks should be detected *as soon as they are discovered* to minimize damage, a task at which PhishLang excelled throughout the experiment, especially against evasive threats as soon they appeared.

VI. Conclusion

In this work, we introduced PhishLang, a lightweight, fully client-side phishing detection framework powered by MobileBERT. By focusing on actionable elements within a website's source code, PhishLang moves beyond traditional heuristic and static features to effectively detect evasive, zero-day phishing threats. When deployed in real time, PhishLang identified thousands of phishing attacks, including many that were missed by popular anti-phishing blocklists. The model also demonstrates strong adversarial resilience, successfully defending against a wide range of problem-space evasion attacks and was further improved through targeted adversarial patches. As a browser extension, PhishLang runs efficiently even on low-end consumer systems, while outperforming several commercial anti-phishing tools in zero-day protection. To support broader adoption and community-driven improvements, we open-source both the PhishLang detection framework and the client-side browser extension.

Ethical Considerations

This research does not involve human subjects, interaction with users, or the collection of any personally identifiable information (PII). All data used for training and evaluation was sourced from publicly available phishing and benign websites. The PhishLang framework operates entirely on publicly accessible metadata and HTML content, and does not engage with backend infrastructure or user data. Accordingly, this study did not require IRB approval.

Risk Mitigation: In accordance with the Menlo Report's [13] principle of Beneficence, we carefully evaluated potential risks associated with the development and deployment of PhishLang, aiming to maximize societal benefit while minimizing harm. Although detailing detection mechanisms could potentially inform adversarial adaptation, we proactively evaluated our framework against 16 types of realistic problem-space adversarial attacks and applied targeted mitigations that render most evasion attempts ineffective or impractical. These improvements significantly enhanced the model's robustness in real-world conditions. PhishLang operates entirely on the client-side, without transmitting any user data externally, thereby eliminating privacy risks during deployment. No remote logging, telemetry, or external dependencies are used.

Responsible Disclosure: All phishing URLs identified by PhishLang during live deployment were reported within 3 hours to relevant hosting providers, blocklist maintainers, and affected organizations. This responsible disclosure process directly contributed to the takedown of threats that had evaded detection by popular antiphishing blocklists and tools.

Legal and Platform Compliance: We ensured compliance with all applicable laws and platform terms of service. No authentication systems were probed, and we did not submit information to live phishing forms. All evaluations were based on passive analysis of publicly accessible HTML content and metadata. Training data was sourced from an open repository (PhishPedia).

Open Source and Data Sharing: We have open-sourced the PhishLang framework and Chromium browser extension to support transparency and reproducibility. However, given the sensitive nature of live phishing websites, we do not release the URLs publicly. Access may be granted upon request under a legal data-sharing agreement, subject to institutional approval.

Acknowledgmesnts

This work was supported by the National Science Foundation under Award No. 2239646 and the Comcast Innovation Grant. We also thank Poojitha Thota, Ryan Lahlou, and Solomon Dandekar for their help in evaluating the functionality of the PhishLang extension across its various iterations.

References

[1] "Google Safebrowsing," https://safebrowsing.google.com/, 2020.

[2] "PhishTank," https://www.phishtank.com/faq.php, 2020.

[3] "The top 500 sites on the web," https://www.alexa.com/topsites, 2020.

[4] "2021 Cybersecurity threat trends: phishing, crypto top the list," https://tinyurl.com/tmztcxsn, 2021.

[5] S. Abdelnabi, K. Krombholz, and M. Fritz, "Visualphishnet: Zero-day phishing website detection by visual similarity," in *Proceedings of the 2020 ACM SIGSAC conference on computer and communications security*, 2020, pp. 1681–1698.

[6] A. Aleroud and L. Zhou, "Phishing environments, techniques, and countermeasures: A survey," *Computers & Security*, vol. 68, pp. 160–196, 2017.

[7] G. Apruzzese, M. Andreolini, L. Ferretti, M. Marchetti, and M. Colajanni, "Modeling realistic adversarial attacks against network intrusion detection systems," *Digital Threats: Research and Practice (DTRAP)*, vol. 3, no. 3, pp. 1–19, 2022.

[8] Argos Open Technologies, LLC, "Argos Translate: Open-source offline translation library," GitHub repository, Jul. 2025, https://github.com/argosopentech/argos-translate.

[9] Auth0. (2021, June) Preventing clickjacking attacks. [Online]. Available: https://auth0.com/blog/preventing-clickjacking-attacks/

[10] AVComparatives, https://www.av-comparatives.org/news/anti-phishing-certification-test-2019/.

[11] T. N. Bac, P. T. Duy, and V.-H. Pham, "Pwdgan: Generating adversarial malicious url examples for deceiving black-box phishing website detector using gans," in *2021 IEEE International Conference on Machine Learning and Applied Network Technologies (ICMLANT)*. IEEE, 2021, pp. 1–4.

[12] A. C. Bahnsen, I. Torroledo, L. D. Camacho, and S. Villegas, "Deep-phish: simulating malicious ai," in *2018 APWG symposium on electronic crime research (eCrime)*, 2018, pp. 1–8.

[13] M. Bailey, D. Dittrich, E. Kenneally, and D. Maughan, "The menlo report," *IEEE Security & Privacy*, vol. 10, no. 2, pp. 71–75, 2012.

[14] Certstream, "Certstream," https://certstream.calidog.io.

[15] R. Chataut, P. K. Gyawali, and Y. Usman, "Can ai keep you safe? a study of large language models for phishing detection," in *2024 IEEE 14th Annual Computing and Communication Workshop and Conference (CCWC)*. IEEE, 2024, pp. 0548–0554.

[16] Y. Chen, F. Zahedi, and A. Abbasi, "Interface design elements for anti-phishing systems," in *Service-Oriented Perspectives in Design Science Research: 6th International Conference, DESRIST 2011, Milwaukee, WI, USA, May 5-6, 2011. Proceedings 6*. Springer, 2011, pp. 253–265.

[17] K. L. Chiew, K. S. C. Yong, and C. L. Tan, "A survey of phishing attacks: their types, vectors and technical approaches," *Expert Systems with Applications*, vol. 106, pp. 1–20, 2018.

[18] M. Ciniselli, N. Cooper, L. Pascarella, D. Poshyvanyk, M. Di Penta, and G. Bavota, "An empirical study on the usage of bert models for code completion," in *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*. IEEE, 2021, pp. 108–119.

[19] I. Corona, B. Biggio, M. Contini, L. Piras, R. Corda, M. Mereu, G. Mureddu, D. Ariu, and F. Roli, "Deltaphish: Detecting phishing webpages in compromised websites," in *Computer Security–ESORICS 2017: 22nd European Symposium on Research in Computer Security, Oslo, Norway, September 11-15, 2017, Proceedings, Part I 22*. Springer, 2017, pp. 370–388.

[20] Darktrace, "Lost in translation: Darktrace blocks non-english phishing campaign concealing hidden payloads," 2024, accessed: 2024-08-08. [Online]. Available: https://darktrace.com/blog/lost-in-translation-darktrace-blocks-non-english-phishing-campaign-concealing-hidden-payloads

[21] L. Demetrio, A. Valenza, G. Costa, and G. Lagorio, "Waf-a-mole: evading web application firewalls through adversarial machine learning," in *Proceedings of the 35th Annual ACM Symposium on Applied Computing*, 2020, pp. 1745–1752.

[22] G. Desolda, J. Aneke, C. Ardito, R. Lanzilotti, and M. F. Costabile, "Explanations in warning dialogs to help users defend against phishing attacks," *International Journal of Human-Computer Studies*, vol. 176, p. 103056, 2023.

[23] S. P. B. . documentation, "Selenium with python — selenium python bindings 2 documentation." [Online]. Available: https://selenium-python.readthedocs.io/

[24] J. S. Downs, M. Holbrook, and L. F. Cranor, "Behavioral response to phishing risk," in *Proceedings of the anti-phishing working groups 2nd annual eCrime researchers summit*, 2007, pp. 37–44.

[25] A. Draganovic, S. Dambra, J. A. Iuit, K. Roundy, and G. Apruzzese, ""do users fall for real adversarial phishing?" investigating the human response to evasive webpages," in *2023 APWG Symposium on Electronic Crime Research (eCrime)*. IEEE, 2023, pp. 1–14.

[26] P. T. Duy, V. Q. Minh, B. T. H. Dang, N. D. H. Son, N. H. Quyen, and V.-H. Pham, "A study on adversarial sample resistance and defense mechanism for multimodal learning-based phishing website detection," *IEEE Access*, 2024.

[27] M. Elsadig, A. O. Ibrahim, S. Basheer, M. A. Alohali, S. Alshunaifi, H. Alqahtani, N. Alharbi, and W. Nagmeldin, "Intelligent deep machine learning cyber phishing url detection based on bert features extraction," *Electronics*, vol. 11, no. 22, p. 3647, 2022.

[28] K. R. et al., "Python requests," 2024, accessed: 2024-08-07. [Online]. Available: https://pypi.org/project/requests/

[29] E. Grave, P. Bojanowski, P. Gupta, A. Joulin, and T. Mikolov, "Learning word vectors for 157 languages," *arXiv preprint arXiv:1802.06893*, 2018.

[30] B. Guo, Y. Zhang, C. Xu, F. Shi, Y. Li, and M. Zhang, "Hinphish: An effective phishing detection approach based on heterogeneous information networks," *Applied Sciences*, vol. 11, no. 20, p. 9733, 2021.

[31] B. B. Gupta, N. A. Arachchilage, and K. E. Psannis, "Defending against phishing attacks: taxonomy of methods, current issues and future directions," *Telecommunication Systems*, vol. 67, no. 2, pp. 247–267, 2018.

[32] D. He, X. Lv, S. Zhu, S. Chan, and K.-K. R. Choo, "A method for detecting phishing websites based on tiny-bert stacking," *IEEE Internet of Things Journal*, 2023.

[33] P. He, X. Liu, J. Gao, and W. Chen, "Deberta: Decoding-enhanced bert with disentangled attention," *arXiv preprint arXiv:2006.03654*, 2020.

[34] F. Heiding, B. Schneier, A. Vishwanath, and J. Bernstein, "Devising and detecting phishing: large language models vs. smaller human models," *arXiv preprint arXiv:2308.12287*, 2023.

[35] Z. Huang and M. Benyoucef, "From e-commerce to social commerce: A close look at design features," *Electronic Commerce Research and Applications*, vol. 12, no. 4, pp. 246–259, 2013.

[36] L. Hudson, "pyzbar · pypi." [Online]. Available: https://pypi.org/project/pyzbar/

[37] X. Jiao, Y. Yin, L. Shang, X. Jiang, X. Chen, L. Li, F. Wang, and Q. Liu, "Tinybert: Distilling bert for natural language understanding," *arXiv preprint arXiv:1909.10351*, 2019.

[38] L. Kang and J. Xiang, "Captcha phishing: a practical attack on human interaction proofing," in *Information Security and Cryptology: 5th International Conference, Inscrypt 2009, Beijing, China, December 12-15, 2009. Revised Selected Papers 5*. Springer, 2010, pp. 411–425.

[39] T. Koide, N. Fukushi, H. Nakano, and D. Chiba, "Detecting phishing sites using chatgpt," *arXiv preprint arXiv:2306.05816*, 2023.

[40] KrebsonSecurity, https://tinyurl.com/4736w7d5.

[41] S. Kuikel, A. Piplai, and P. Aggarwal, "Evaluating large language models for phishing detection, self-consistency, faithfulness, and explainability," *arXiv preprint arXiv:2506.13746*, 2025.

[42] H. Le, Q. Pham, D. Sahoo, and S. C. Hoi, "Urlnet: Learning a url representation with deep learning for malicious url detection," *arXiv preprint arXiv:1802.03162*, 2018.

[43] T. Le Scao, A. Fan, C. Akiki, E. Pavlick, S. Ilić, D. Hesslow, R. Castagné, A. S. Luccioni, F. Yvon, M. Gallé *et al.*, "Bloom: A 176b-parameter open-access multilingual language model," 2023.

[44] J. Lee, P. Ye, R. Liu, D. M. Divakaran, and M. C. Chan, "Building robust phishing detection system: an empirical analysis," *NDSS MADWeb*, 2020.

[45] Y. Li, Z. Yang, X. Chen, H. Yuan, and W. Liu, "A stacking model using url and html features for phishing webpage detection," *Future Generation Computer Systems*, vol. 94, pp. 27–39, 2019.

[46] B. Liang, M. Su, W. You, W. Shi, and G. Yang, "Cracking classifiers for evasion: A case study on the google's phishing pages filter," in *Proceedings of the 25th International Conference on World Wide Web*, 2016, pp. 345–356.

[47] Y. Lin, R. Liu, D. M. Divakaran, J. Y. Ng, Q. Z. Chan, Y. Lu, Y. Si, F. Zhang, and J. S. Dong, "Phishpedia: A hybrid deep learning based approach to visually identify phishing webpages." in *USENIX Security Symposium*, 2021, pp. 3793–3810.

[48] R. Liu, Y. Lin, X. Teoh, G. Liu, Z. Huang, and J. S. Dong, "Less defined knowledge and more true alarms: Reference-based phishing detection without a pre-defined reference list," in *33rd USENIX Security Symposium (USENIX Security 24)*, 2024, pp. 523–540.

[49] R. Liu, Y. Lin, X. Yang, S. H. Ng, D. M. Divakaran, and J. S. Dong, "Inferring phishing intention via webpage appearance and dynamics: A deep vision based approach," in *30th {USENIX} Security Symposium ({USENIX} Security 21)*, 2022.

[50] J. Mao, W. Tian, P. Li, T. Wei, and Z. Liang, "Phishing-alarm: Robust and efficient phishing detection via page component similarity," *IEEE Access*, vol. 5, pp. 17 020–17 030, 2017.

[51] Microsoft, "Microsoft defender smartscreen overview - windows security." [Online]. Available: https://learn.microsoft.com/en-us/windows/security/operating-system-security/virus-and-threat-protection/microsoft-defender-smartscreen/

[52] A. Mishra and B. B. Gupta, "Intelligent phishing detection system using similarity matching algorithms," *International Journal of Information and Communication Technology*, vol. 12, no. 1-2, pp. 51–73, 2018.

[53] B. Montaruli, L. Demetrio, M. Pintor, L. Compagna, D. Balzarotti, and B. Biggio, "Raze to the ground: Query-efficient adversarial html attacks on machine-learning phishing webpage detectors," in *Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security*, 2023, pp. 233–244.

[54] L. A. T. Nguyen, B. L. To, H. K. Nguyen, and M. H. Nguyen, "Detecting phishing web sites: A heuristic url-based approach," in *2013 International Conference on Advanced Technologies for Communications (ATC 2013)*. IEEE, 2013, pp. 597–602.

[55] A. Oest, Y. Safaei, A. Doupé, G.-J. Ahn, B. Wardman, and K. Tyers, "Phishfarm: A scalable framework for measuring the effectiveness of evasion techniques against browser phishing blacklists," in *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2019, pp. 1344–1361.

[56] A. Oest, Y. Safaei, P. Zhang, B. Wardman, K. Tyers, Y. Shoshitaishvili, and A. Doupé, "Phishtime: Continuous longitudinal measurement of the effectiveness of anti-phishing blacklists," in *29th {USENIX} Security Symposium ({USENIX} Security 20)*, 2020, pp. 379–396.

[57] A. Oest, P. Zhang, B. Wardman, E. Nunes, J. Burgis, A. Zand, K. Thomas, A. Doupé, and G.-J. Ahn, "Sunrise to sunset: Analyzing the end-to-end life cycle and effectiveness of phishing attacks at scale," in *29th USENIX Security Symposium (USENIX Security 20)*, 2020.

[58] Openphish, "Phishing feed," "https://openphish.com/faq.html".

[59] OpenPhish, "Openphish monthly brands list," https://openphish.com/monthly_brands_list.csv, 2022.

[60] T. K. Panum, K. Hageman, R. R. Hansen, and J. M. Pedersen, "Towards adversarial phishing detection," in *13th USENIX Workshop on Cyber Security Experimentation and Test (CSET 20)*, 2020.

[61] P. Peng, L. Yang, L. Song, and G. Wang, "Opening the blackbox of virustotal: Analyzing online phishing scan engines," in *Proceedings of the Internet Measurement Conference*, 2019, pp. 478–485.

[62] ——, "Opening the blackbox of virustotal: Analyzing online phishing scan engines," in *Proceedings of the Internet Measurement Conference*, 2019, pp. 478–485.

[63] F. Pierazzi, F. Pendlebury, J. Cortellazzi, and L. Cavallaro, "Intriguing properties of adversarial ml attacks in the problem space," in *2020 IEEE symposium on security and privacy (SP)*. IEEE, 2020, pp. 1332–1349.

[64] R. Pourmohamad, S. Wirsz, A. Oest, T. Bao, Y. Shoshitaishvili, R. Wang, A. Doupé, and R. A. Bazzi, "Deep dive into client-side anti-phishing: A longitudinal study bridging academia and industry," in *Proceedings of the 19th ACM Asia Conference on Computer and Communications Security*, 2024, pp. 638–653.

[65] R. W. Purwanto, A. Pal, A. Blair, and S. Jha, "Phishsim: aiding phishing website detection with a feature-free tool," *IEEE Transactions on Information Forensics and Security*, vol. 17, pp. 1497–1512, 2022.

[66] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever *et al.*, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[67] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the limits of transfer learning with a unified text-to-text transformer," *Journal of machine learning research*, vol. 21, no. 140, pp. 1–67, 2020.

[68] A. Rahali and M. A. Akhloufi, "Malbertv2: Code aware bert-based model for malware identification," *Big Data and Cognitive Computing*, vol. 7, no. 2, p. 60, 2023.

[69] S. Roopak and T. Thomas, "A novel phishing page detection mechanism using html source code comparison and cosine similarity," in *2014 Fourth International Conference on Advances in Computing and Communications*. IEEE, 2014, pp. 167–170.

[70] S. S. Roy, U. Karanjit, and S. Nilizadeh, "Evaluating the effectiveness of phishing reports on twitter," in *2021 APWG Symposium on Electronic Crime Research (eCrime)*. IEEE, 2021, pp. 1–13.

[71] ——, "A large-scale analysis of phishing websites hosted on free web hosting domains," *arXiv preprint arXiv:2212.02563*, 2022.

[72] S. Saha Roy, U. Karanjit, and S. Nilizadeh, "Phishing in the free waters: A study of phishing attacks created using free website building services," in *Proceedings of the 2023 ACM on Internet Measurement Conference*, 2023, pp. 268–281.

[73] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter," *arXiv preprint arXiv:1910.01108*, 2019.

[74] G. Sarti and M. Nissim, "It5: Large-scale text-to-text pretraining for italian language understanding and generation," *arXiv preprint arXiv:2203.03759*, 2022.

[75] H. Shirazi, B. Bezawada, I. Ray, and C. Anderson, "Adversarial sampling attacks against phishing detection," in *Data and Applications Security and Privacy XXXIII: 33rd Annual IFIP WG 11.3 Conference, DBSec 2019, Charleston, SC, USA, July 15–17, 2019, Proceedings 33*. Springer, 2019, pp. 83–101.

[76] S. Sivakorn, J. Polakis, and A. D. Keromytis, "I'm not a human: Breaking the google recaptcha," *Black Hat*, vol. 14, pp. 1–12, 2016.

[77] J. Sreedharan and R. Mohandas, "Systems and methods for risk rating and pro-actively detecting malicious online ads," apr 5 2016, uS Patent 9,306,968.

[78] S. G. Stats, "Browser market share worldwide." [Online]. Available: https://gs.statcounter.com/browser-market-share

[79] Z. Sun, H. Yu, X. Song, R. Liu, Y. Yang, and D. Zhou, "Mobilebert: a compact task-agnostic bert for resource-limited devices," *arXiv preprint arXiv:2004.02984*, 2020.

[80] S. Team, "iframe injection attacks and mitigation," *SecNHack*, February 2022, [Online; accessed 9-March-2023]. [Online]. Available: https://secnhack.in/iframe-injection-attacks-and-mitigation/

[81] C. Tom Huddleston Jr., "How this scammer used phishing emails to steal over $100 million from google and facebook," bit.ly/45qZXTM.

[82] L. Tong, B. Li, C. Hajaj, C. Xiao, N. Zhang, and Y. Vorobeychik, "Improving robustness of {ML} classifiers against realizable evasion attacks using conserved features," in *28th USENIX Security Symposium (USENIX Security 19)*, 2019, pp. 285–302.

[83] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar *et al.*, "Llama: Open and efficient foundation language models," *arXiv preprint arXiv:2302.13971*, 2023.

[84] F. Trad and A. Chehab, "Prompt engineering or fine-tuning? a case study on phishing detection with large language models," *Machine Learning and Knowledge Extraction*, vol. 6, no. 1, pp. 367–384, 2024.

[85] C. Ventures, "Beware of lookalike domains in punycode phishing attacks," *Cybersecurity Ventures*, 2019. [Online]. Available: https://cybersecurityventures.com/beware-of-lookalike-domains-in-punycode-phishing-attacks/

[86] T. Vidas, E. Owusu, S. Wang, C. Zeng, L. F. Cranor, and N. Christin, "Qrishing: The susceptibility of smartphone users to qr code phishing attacks," in *Financial Cryptography and Data Security: FC 2013 Workshops, USEC and WAHC 2013, Okinawa, Japan, April 1, 2013, Revised Selected Papers 17*. Springer, 2013, pp. 52–69.

[87] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. L. Scao, S. Gugger, M. Drame, Q. Lhoest, and A. M. Rush, "Transformers: State-of-the-art natural language processing," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Online: Association for Computational Linguistics, Oct. 2020, pp. 38–45. [Online]. Available: https://www.aclweb.org/anthology/2020.emnlp-demos.6

[88] J. Wright and contributors, "Gophish: Open-source phishing toolkit," https://github.com/gophish/gophish, 2022.

[89] S. Yesir and İ. Soğukpinar, "Malware detection and classification using fasttext and bert," in *2021 9th International Symposium on Digital Forensics and Security (ISDFS)*. IEEE, 2021, pp. 1–6.

[90] Y. Yuan, G. Apruzzese, and M. Conti, "Multi-spacephish: Extending the evasion-space of adversarial attacks against phishing website detectors using machine learning," *Digital Threats: Research and Practice*, 2023.

[91] ——, "Multi-spacephish: Extending the evasion-space of adversarial attacks against phishing website detectors using machine learning," *Digital Threats: Research and Practice*, vol. 5, no. 2, pp. 1–51, 2024.

[92] Y. Yuan, Q. Hao, G. Apruzzese, M. Conti, and G. Wang, "" are adversarial phishing webpages a threat in reality?" understanding the users' perception of adversarial webpages," in *Proceedings of the ACM on Web Conference 2024*, 2024, pp. 1712–1723.

[93] W. E. Zhang, Q. Z. Sheng, A. Alhazmi, and C. Li, "Adversarial attacks on deep-learning models in natural language processing: A survey," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 11, no. 3, pp. 1–41, 2020.

[94] Z. Zhang, Y. Wu, H. Zhao, Z. Li, S. Zhang, X. Zhou, and X. Zhou, "Semantics-aware bert for language understanding," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 05, 2020, pp. 9628–9635.

[95] Y. Zhou, Y. Zhang, J. Xiao, Y. Wang, and W. Lin, "Visual similarity based anti-phishing with the combination of local and global features," in *2014 IEEE 13th International Conference on Trust, Security and Privacy in Computing and Communications*. IEEE, 2014, pp. 189–196.

Appendix A
Taxonomy and classifier for categorizing evasive phishing pages

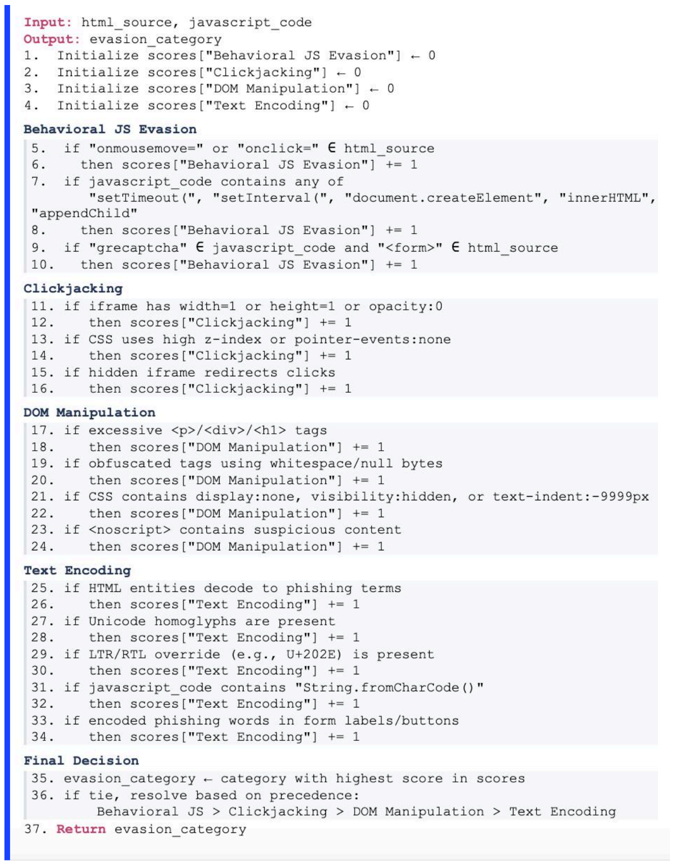
In Section III-A, our coders manually looked through 2,500 websites that were identified by PhishLang in real-time, and validated 2,388 as phishing. They further used a series of heuristics to categorize these websites into regular phishing and four evasive categories. Table X outlines the structural and behavioral indicators derived from prior literature [56], [80], [9], [46], [85] that were used by them to categorize the websites manually. These features were chosen for their reproducibility and ability to capture evasive techniques that manipulate rendering behavior, obscure intent, or evade automated scanners. The manual annotation process involved the coders inspecting each website's source code to detect the presence of these evasive features. For example, to identify Behavioral JavaScript Evasions (as described in Liang et al. [46]), coders flagged websites containing constructs such as `setTimeout()` or `setInterval()` to delay the appearance of key elements like login forms, or those using `document.write()` and `innerHTML` to dynamically inject phishing payloads post-render. Both coders jointly reviewed and labeled the complete set of 2,500 websites, and disagreements were resolved through three adjudication sessions where both coders reviewed the raw HTML and JavaScript code together to assess how specific constructs aligned with the evasive feature definitions. The resulting inter-coder agreement was 0.91, reflecting strong reliability and consistency in the manual classification.

Automate Detection of Attack Categories: To automate this classification of evasive behavior, we encoded these same heuristics into a lightweight, rule-based detection algorithm. Unlike the evaluation by the human coders, qualitative judgment is not feasible in automated settings; thus, we introduced a quantitative scoring rule. This algorithm takes the HTML source and JavaScript content (when present) of a page as input, scans for the known evasive features, and assigns a single evasive category based on a scoring strategy. The scoring system reflects the number of matched heuristics per category. A fallback category of Regular Phishing is used when no evasive indicators are detected. The full pseudocode of this detection logic is provided in Figure 8

To evaluate the effectiveness of this rule-based approach, we applied it to the same 2,388 websites that were previously categorized manually by the coders. The rule-based algorithm correctly matched the coders' labels in 2,270 cases (95.1%). It achieved high agreement across all categories: 1,546 out of 1,623 Regular Phishing (95.2%), 266 out of 280 Behavioral JavaScript Evasion (95.0%), 295 out of 313 Clickjacking (94.2%), 87 out of 94 DOM Manipulation (92.6%), and 76 out of 78 Text Encoding (97.4%). These results suggest that the rule-based classifier provides a reliable and scalable alternative to manual annotation. However, these feature lists are not exhaustive, and it is possible that during both manual inspection and automated detection, some evasive websites may have been mislabeled as regular. While the chosen categories reflect several widely studied evasion strategies, they do not encompass the full spectrum of evasive techniques seen in the wild.

TABLE X: Technical Features Used for Evasion Labeling

Attack Category	Technical Features Used for Labeling
Behavioral JS Evasion [59]	JS-triggered content on events: `onmousemove`, `onclick` `setTimeout()` or `setInterval()` delays before UI shown DOM created via `document.createElement` Content injected using `innerHTML`, `appendChild` Phishing form rendered only after CAPTCHA interaction
Clickjacking [95, 16]	`<iframe>` with `width=1`, `height=1`, or `opacity: 0` `z-index` stacking to overlay invisible login forms Multiple iframes with `pointer-events: none` Hidden iframes redirecting click actions
DOM Manipulation [59]	Benign-looking tag insertion: `<p>`, `<div>`, `<h1>` Obfuscation via whitespace or null bytes in tags CSS: `display: none`, `visibility: hidden`, `text-indent: -9999px` Misleading content inside `<noscript>` blocks
Text Encoding [100, 59]	HTML entities like `&#x6C;&#x6F;&x67;&#x69;&#x6E;` Unicode homoglyphs (e.g., Cyrillic ‘’ for Latin ‘a’) LTR/RTL override characters (e.g., `&#x202E;`) JavaScript obfuscation via `String.fromCharCode()` Encoded form labels/button text to evade filters



```
Input: html_source, javascript_code
Output: evasion_category
1.  Initialize scores["Behavioral JS Evasion"] ← 0
2.  Initialize scores["Clickjacking"] ← 0
3.  Initialize scores["DOM Manipulation"] ← 0
4.  Initialize scores["Text Encoding"] ← 0
Behavioral JS Evasion
 5.  if "onmousemove=" or "onclick=" ∈ html_source
 6.    then scores["Behavioral JS Evasion"] += 1
 7.  if javascript_code contains any of
        "setTimeout(", "setInterval(", "document.createElement", "innerHTML",
 "appendChild"
 8.    then scores["Behavioral JS Evasion"] += 1
 9.  if "grecaptcha" ∈ javascript_code and "<form>" ∈ html_source
 10.   then scores["Behavioral JS Evasion"] += 1
Clickjacking
 11. if iframe has width=1 or height=1 or opacity:0
 12.    then scores["Clickjacking"] += 1
 13. if CSS uses high z-index or pointer-events:none
 14.    then scores["Clickjacking"] += 1
 15. if hidden iframe redirects clicks
 16.    then scores["Clickjacking"] += 1
DOM Manipulation
 17. if excessive <p>/<div>/<h1> tags
 18.    then scores["DOM Manipulation"] += 1
 19. if obfuscated tags using whitespace/null bytes
 20.    then scores["DOM Manipulation"] += 1
 21. if CSS contains display:none, visibility:hidden, or text-indent:-9999px
 22.    then scores["DOM Manipulation"] += 1
 23. if <noscript> contains suspicious content
 24.    then scores["DOM Manipulation"] += 1
Text Encoding
 25. if HTML entities decode to phishing terms
 26.    then scores["Text Encoding"] += 1
 27. if Unicode homoglyphs are present
 28.    then scores["Text Encoding"] += 1
 29. if LTR/RTL override (e.g., U+202E) is present
 30.    then scores["Text Encoding"] += 1
 31. if javascript_code contains "String.fromCharCode()"
 32.    then scores["Text Encoding"] += 1
 33. if encoded phishing words in form labels/buttons
 34.    then scores["Text Encoding"] += 1
Final Decision
 35. evasion_category ← category with highest score in scores
 36. if tie, resolve based on precedence:
        Behavioral JS > Clickjacking > DOM Manipulation > Text Encoding
37. Return evasion_category
```

Fig. 8: Pseudocode for automated scoring-based categorization of evasive phishing pages

Appendix B
GPT Prompts

To further validate the coders' selection of the tags, we queried GPT-4 with 3,000 confirmed phishing websites from the PhishPedia dataset, asking it to identify HTML tags most relevant to phishing detection. Prior work has shown GPT-4 to be proficient at identifying phishing patterns [15], [39], making it a credible secondary reference. The model confirmed our tag set and additionally recommended `<applet>` and `<img>`. However, `<applet>` was excluded due to its deprecation in HTML5 and extremely low prevalence in the dataset. `<img>` was excluded because it typically encodes filenames or links rather than semantic content, which a language model cannot interpret. The GPT prompt used for this task is provided in the next page.

We also used a GPT prompt later in Section II-C to compare their performance in phishing website detection versus PhishLang. That prompt is also provided in the next page.

Appendix C
Examples of Misdetections and Improvements

We present examples of the false positives and false negatives discussed in Section III-B, which were initially misclassified by PhishLang but later resolved through parser modifications and adversarial patches.

Prompt used for validating Malicious Tags

You are an expert in phishing detection. I will provide the HTML source code of a known phishing website. Based on its content and structure, identify which HTML tags in this specific webpage are most relevant for detecting phishing behavior. Focus on actionable or deceptive elements and ignore aesthetic tags unless they contribute to phishing functionality.
Please return a list of relevant tags along with a brief explanation for why each tag may indicate phishing activity in this page.
The source code is: {`source-code`}
ONLY output your response in the following format: {`Tag1: Explanation1; Tag2: Explanation2; ...`}
Do not include any other commentary.

Prompt for detecting phishing websites

You are a cybersecurity expert specializing in phishing detection. Below is the HTML source code of a website. Based on its content and structure, determine whether this website is a **phishing website** or a **benign website**.
Consider whether the website:
Attempts to impersonate a legitimate organization
Collects sensitive information such as usernames or passwords
Uses deceptive elements such as misleading form actions, fake login interfaces, or suspicious links

Please respond with one of the following two options only: `Phishing` or `Benign`
The source code is: {`source-code`}
ONLY output one word: `Phishing` or `Benign`
Do not include any explanation or additional commentary.

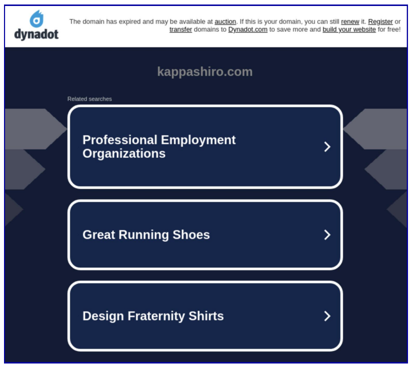
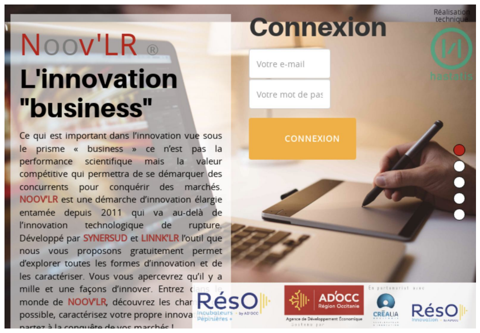

The domain has expired and may be available at auction. If this is your domain, you can still renew it. Register or transfer domains to Dynadot.com to save more and build your website for free!

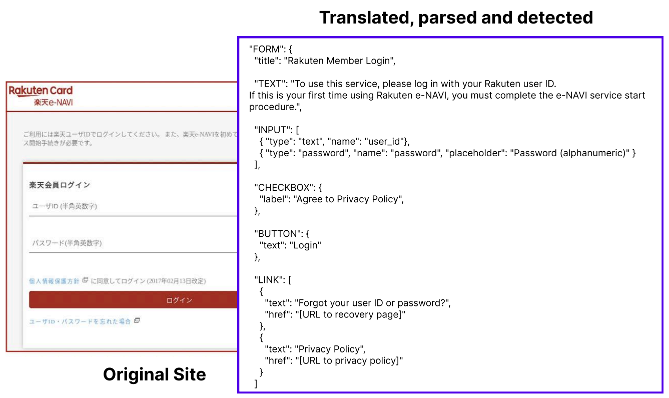
Fig. 9: An example of a parked domain page that was flagged as a false positive


Noov'LR ®

L'innovation "business"

Ce qui est important dans l'innovation vue sous le prisme « business » ce n'est pas la performance scientifique mais la valeur compétitive qui permettra de se démarquer des concurrents pour conquérir des marchés. NOOV'LR est une démarche d'innovation élargie entamée depuis 2011 qui va au-delà de l'innovation technologique de rupture. Développé par SYNERSUD et LINNK'LR l'outil que nous vous proposons gratuitement permet d'explorer toutes les formes d'innovation et de les caractériser. Vous vous apercevrez qu'il y a mille et une façons d'innover. Entrez d monde de NOOV'LR, découvrez les char possible, caractérisez votre propre innova

Fig. 10: An example of a false positive with poor layout


Translated, parsed and detected
Rakuten Card
楽天e-NAVI
ご利用には楽天ユーザIDでログインしてください。 また、楽天e-NAVIを初めて
ス開始手続きが必要です。
楽天会員ログイン
ユーザID (半角英数字)
パスワード(半角英数字)
個人情報保護方針 に同意してログイン (2017年02月13日改定)
ログイン
ユーザID・パスワードを忘れた場合
Original Site

```
"FORM": {
  "title": "Rakuten Member Login",

  "TEXT": "To use this service, please log in with your Rakuten user ID.
If this is your first time using Rakuten e-NAVI, you must complete the e-NAVI service start
procedure.",

  "INPUT": [
   { "type": "text", "name": "user_id"},
   { "type": "password", "name": "password", "placeholder": "Password (alphanumeric)" }
  ],

  "CHECKBOX": {
   "label": "Agree to Privacy Policy",
  },

  "BUTTON": {
   "text": "Login"
  },

  "LINK": [
   {
    "text": "Forgot your user ID or password?",
    "href": "[URL to recovery page]"
   },
   {
    "text": "Privacy Policy",
    "href": "[URL to privacy policy]"
   }
  ]
```

Fig. 11: Misdetection in Figure 4 that was translated by Argos library and then detected by PhishLang.