# PhishDecloaker: Detecting CAPTCHA-cloaked Phishing Websites via Hybrid Vision-based Interactive Models


## Abstract

Phishing is a cybersecurity attack based on social engineering, incurring huge loss of finance and societal trust. While phishing defense techniques are emerging, phishing attackers never stop their trials to bypass the state-of-the-art defending solutions. Recent phishing campaigns have witnessed that CAPTCHA-based cloaking technique are adopted by emerging phishing attacks as a new round of cat-and-rat game. Our study further shows that phishing websites, *hardened* by the CAPTCHA as a cloaking technique, can compromise all known state-of-the-art industrial and academic detectors with almost *zero* cost.

In this work, we develop PhishDecloaker, an AI-powered solution to *soften* the shield of the CAPTCHA-based cloaking of the phishing websites. PhishDecloaker is designed to mimic human behaviors to solve the CAPTCHAs, allowing the modern security-crawlers to see the cloaked phishing content. Technically, PhishDecloaker orchestrates 5 types of deep computer vision models to detect the existence of CAPTCHA, analyze its taxonomy, and solve the challenge in an interactive manner. We conduct extensive experiments to evaluate PhishDecloaker in its effectiveness, efficiency, and robustness against potential adversaries. The results show that PhishDecloaker can (1) recover the phishing detection rate of many state-of-the-art phishing detectors from 0% to up to on average 74.25% on diverse CAPTCHA-cloaked phishing websites (2) generalizable to unseen CAPTCHA types (with the precision of 86% and the recall of 69%), and (3) robust against various adversaries such as FGSM, JSMA, PGD, DeepFool, and DPatch, which allows the existing phishing detectors to achieve new state-of-the-art performance on CPATCHA-cloaked phishing webpages.


## 1 Introduction

Phishing attacks cause enormous financial losses and undermined societal trust. Recent years have seen that the number of phishing attacks has grown by over 150% per year [7]. To mitigate the consequence, researchers have proposed various phishing detection solutions [2, 5, 31–34, 39] to report and explain diverse zero-day phishing websites. While those solutions can be effective against the phishing website, their effectiveness is largely based on the assumption that *a security crawler can access the phishing content of the websites*. Unfortunately, in the new round of phishing campaigns, a growing number of evidence [45,67] has shown that the assumption is less likely to be true with the emerging CAPTCHA-based cloaking techniques.

Cloaking is an increasingly adopted evasion technique by phishing attackers to display different content to security crawlers and human victims [67]. The attackers can deploy the server-side or the client-side cloaking for their phishing webpages. The server-side cloaking checks human visits by analyzing HTTP requests from the server end, such as parsing the IP address and HTTP header [28]. The client-side cloaking checks human visits by analyzing the runtime browser behaviors, such as cookie settings, browser canvas, and WebGL capabilities [3,67]. In recent years, researchers and security engineers propose remedies such as simulating a human-mimic HTTP header (to address server-side cloaking) [28] and force executing the Javascript code of cloaking (to address client-side cloaking) [67]. However, CAPTCHA-based cloaking, as an emerging novel phishing-cloaking technique, can easily nullify those anti-cloaking efforts.

CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart) was initially developed as challenge-response authentication to limit the abuse of web crawling. CAPTCHA validates a human visit with the interaction between the client and the server. On the client side, the website prompts a CAPTCHA challenge, such as picture selection and text recognition, to collect the challenge response. On the server side, the challenge response is validated against the ground-truth answer. By this means, neither the HTTP-request modification nor the Javascript force execution technique can bypass the validation. CAPTCHA can effectively serve as a cloaking technique in three folds:

Listing 1: Embeded CAPTCHA Code in HTML File.

```
<html>
...
<!-- CAPTCHA library>
<script src="https://js.hcaptcha.com/1/api.js"
        async defer></script>
...
<!-- embedded CAPTCHA div-tag>
<div id="cloaking">
    <form id="form" method="post"
    <div class="h-captcha" data-sitekey="..."
         data-callback="submitForm" />
    <input type="hidden" value="hcaptcha" name="
         captchaType" />
    </form>
</div>
...
</html>
```

- **Less Suspciousness:** CAPTCHAs are largely used in many websites, thus a phishing website with prompted CAPTCHA challenge can usually preserve its plausibility without raising the victims' alarm.

- **Low Deployment Cost:** Any website can conveniently invoke CAPTCHA service or API in a website (see Listing 1). Therefore, it is not difficult for the phishing attackers to automatically generate phishing kits equipped with CAPTCHA-cloaking. In addition, there are many free version of CAPTCHA. Thus, it incurs almost zero cost to *harden* the phishing website with a CAPTCHA.

- **Hard to Bypass:** With CAPTCHA's server-client architecture, it is non-trivial to automatically bypass its cloaking by modern security crawlers.

Recent studies have shown that the phishing attackers are adopting CAPTCHA as a novel cloaking technique [4, 34, 38, 45, 61, 67]. The number of CAPTCHA-cloaked phishing websites has increased almost tenfold from 55,447 on January 2023 to 524,344 on June 2023. [40]. Further, our studies (see Section 2) further show that *none* of the public industrial phishing detection engines or academic state-of-the-arts can report a CAPTCHA-cloaked phishing website.

In this work, we propose PhishDecloaker, as the first step to mitigate the CAPTCHA-based cloaking problem. PhishDecloaker is designed to mimic human behaviors to solve the CAPTCHA in an interactive manner. Technically, PhishDecloaker orchestrates 5 types of deep computer vision models, consisting of three stages, i.e., detection, recognition, and solving. Specifically, PhishDecloaker first detects the existence of a CAPTCHA, by formulating it as an object detection problem [32, 33, 61] on a webpage screenshot. Then, PhishDecloaker recognizes what types of CAPTCHA (as a metric learning problem) so that it can schedule a follow-up challenge-solving plan. Such a three-stage design allows us to flexibly extend PhishDecloaker to solve new types of CAPTCHA and preserve its performance even on out-of-distribution CAPTCHAs. Our implementation of PhishDecloaker supports CAPTCHA types such as re-CAPTCHA, hCAPTCHA, slider, and rotation, taking 98.9% of the CAPTCHA market share [11].

We conduct extensive experiments to evaluate PhishDecloaker in its effectiveness, efficiency, and robustness against potential adversaries. The results show that PhishDecloaker can (1) effectively recover the phishing detection rate of many state-of-the-art phishing detectors from 0% to an average of 74.25% on diverse CAPTCHA-cloaked phishing websites and (2) generalizable to unseen CAPTCHA (with average precision and recall of 86% and 69%), and (2) robust against various evasion attacks such as FGSM, JSMA, PGD, DeepFool, and DPatch.

In summary, this work makes the following contributions:

- We develop PhishDecloaker, a hybrid deep-vision system to detect, recognize, and solve diverse CAPTCHAs. The system is accurate to support the mainstream CAPTCHAs and is extensible for new types of CAPTCHAs. To the best of our knowledge, our work is the first to address the CAPTCHA-based cloaking problem for phishing detection.

- We deliver a CAPTCHA-based hardening framework, which allows us to experimentally translate a phishing kit to its CAPTCHA-based cloaking version. We implement the technique on top of the published DynaPD dataset[1] [34] as its expansion.

- We deliver PhishDecloaker as a tool, integrated with existing state-of-the-art phishing detectors, significantly enhancing their capabilities of detecting CAPTCHA-cloaked phishing websites.

- We conduct extensive experiments to evaluate our Phishdecloaker. Our results show that PhishDecloaker can effectively solve the CAPTCHA challenge. Further, PhishDecloaker is robust against out-of-distribution CAPTCHAs and potential adversarial attacks.

Given the space limit, more details of PhishDecloaker are available at [49].

## 2 An Empirical Study of Anti-Phishing Entities against CAPTCHA-Cloaking

In this section, we introduce our empirical study to answer the research questions as *what is the performance of the state-of-the-art anti-phishing solutions on accessing CAPTCHA-cloaked websites?*.

To answer the question, we design a website-hardening framework, Cloaken, to automatically equip a website with

[1]The DynaPD dataset provides thousands of alive phishing websites to interact with security researchers.

Table 1: Results of empirical study on evaluating the capability of the state-of-the-art phishing detectors to solve CAPTCHAs.

| Phishing Detectors | URLs Submitted / CAPTCHA Solved | | | |
|---|---|---|---|---|
| | reCAPTCHA v2 | hCaptcha | GeeTest Slide | Rotation |
| Phishpedia | 30 / 0 (0%) | 30 / 0 (0%) | 30 / 0 (0%) | 30 / 0 (0%) |
| Phishintention | 30 / 0 (0%) | 30 / 0 (0%) | 30 / 0 (0%) | 30 / 0 (0%) |
| VirusTotal (including 92 phishing detectors) | 30 / 0 (0%) | 30 / 0 (0%) | 30 / 0 (0%) | 30 / 0 (0%) |

its CAPTCHA-cloaking functionality. Specifically, Cloaken prepares a set of CAPTCHA service implementation such as reCAPTCHA and hCATPCHA. Given the source code of a website, Cloaken is designed to instrument the webpage with the code to generate a CAPTCHA challenge. Cloaken can generate either a user-specified CAPTCHA type or a random CAPTCHA by default.

**Phishing Detectors.** We select all the 92 industrial phishing detection engines included by VirusTotal [53] and 2 academic phishing detectors as Phishpedia [32] and PhishIntention [33]. The industrial phishing detectors are selected for the popularity and effectiveness of VirusTotal [3,4,38,45]. The academic detectors are selected for their state-of-the-art performance of reporting zero-day phishing websites in the wild [32, 33].

**CAPTCHA Types.** We select four types of CAPTCHAs, i.e., reCAPTCHA v2, hCAPTCHA, GeeTest Slide, and Rotation in this study, based on their popularity in the shared market [55].

**Evaluation Setup.** In the study, we call a pair of detector and CAPTCHA-type, $(d, t_c)$, as a configuration where $d$ represents a phishing detector and $t_c$ represents a type of CAPTCHA. For each configuration, we generate $k$ webpages with unique URLs, equip it with a random CAPTCHA of a type $t_c$, and submit it to $d$ for analysis. Once an URL is submitted, a phishing detector will activate a web crawler to visit its webpage. For each generated webpage, we leave a script to record the visiting event if its protecting CAPTCHA is solved. Note that VirusTotal includes 92 phishing detectors, thus we consider that VirusTotal can solve the CAPTCHA of a configuration if any of its phishing detectors can solve the CAPTCHA. We deployed the system on a singular EC2 instance within Amazon Web Services (AWS). In this study, we let $k$ be 30, i.e., 30 submitted URLs for each configuration. These submissions were distributed at random intervals throughout the study. More details are available at [49].

Note that we do not generate phishing websites in this study as we just need to evaluate the capability of different phishing detectors to parse CAPTCHA. Moreover, submitting fake phishing reports to anti-phishing entities raises ethical concerns, we discuss them in detail in Section 6.

**Results.** Table 1 shows the results of 92 phishing detectors in the study. Overall, none of the selected phishing detectors can access the CAPTCHA-cloaked websites. This findings indicates a need for the modern security crawlers to solve CAPTCHAs, further motivating us to design PhishDecloaker, a solution towards CAPTCHA-cloaked phishing websites.

## 3 Threat Model

Assume that there are a set of phishing detectors $\mathcal{D} = \{d_1, d_2, ..., d_n\}$, where each detector $d_i$ $(i = 1, 2, ..., n)$ needs to access the webpage content to report phishing alarm. Further, each detector $d_i$ can be modelled as a function $d_i(.) : \mathcal{W} \rightarrow \{0, 1\}$ where $\mathcal{W}$ is the set of webpages. Specifically, a detector $d_i(.)$ maps a webpage $w \in \mathcal{W}$ to a boolean value where 0 indicates that $w$ is benign and 1 indicates that $w$ is phishing.

In our threat model, an attacker can equip his or her phishing website $w_p$ with a CAPTCHA instance $c \in \mathcal{C}$ where $\mathcal{C}$ is the set of CAPTCHA instances under pre-defined CAPTCHA types (e.g., reCAPTCHA, hCaptcha, and GeeTest). Equipped with a human-authentication challenge by $c$, the attacker can render a new webpage $w'_p \leftarrow c \oplus w_p$ so that $\forall d_i \in \mathcal{D}, d_i(w'_p) = 0$, where $\oplus$ is an operation to render the CAPTCHA on top of the webpage $w_p$. The prepared CAPTCHA sets share the following features:

- **Diverse Types of CAPTCHAs**. The attacker can adopt diverse types of CAPTCHA including commercial versions (e.g., reCAPTCHA and hCAPTCHA) and open-source versions. Also, we assume that the attacker customize its own implementation of well-known CAPTCHA challenge, e.g., with similar appearance to reCAPTCHA and hCAPTCHA.
- **Code Obfuscation**. The CAPTCHA can have its partial execution in the client side, for example implemented by Javascript code. We assume that the attacker can adopt code obfuscation techniques [1, 59] to modify the underlying code or structure of CAPTCHAs while preserving the same appearance and functionality inside a browser.
- **Adversarial Images**. An attacker may introduce noise and distortion into CAPTCHA images [26, 51, 57]. This can make it difficult for deep learning models to extract the relevant features and classify the CAPTCHA accurately, as the modified CAPTCHA may no longer conform to the expected patterns or features used for classification.

Given the above threat model, we adopt a vision-based solution (in comparison to program analysis solution) to detect and classify CAPTCHAs. In other words, we require that our solution (1) does not reply on code analysis and (2) must be robust against out-of-distribution CAPTCHAs and adversarial attacks such as noise and distortion.

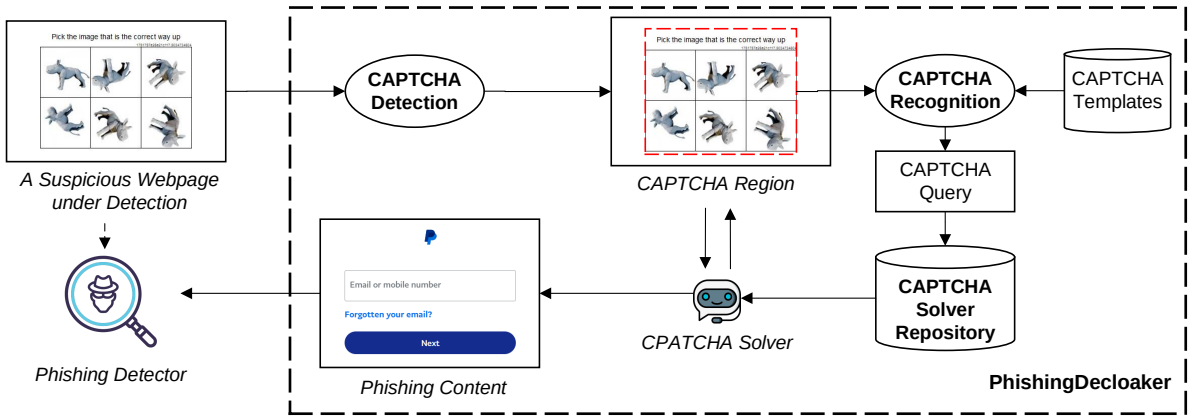



Figure 1: System Design of PhishDecloaker. PhishDecloaker is designed to remove the "cloak" of a CAPTCHA-cloaked phishing webpage, by detecting, recognizing, and solving the CPATCHA challenges.

## 4 Approach

**Overview.** Figure 1 provides an overview of PhishDecloaker. Given a suspicious webpage $w_p$ with cloaking potential, contrary to feed $w_p$ into a phishing detector (Google Safe Browsing, Phishpedia, etc.), PhishDecloaker tries to remove its "cloak" by looking and interacting with $w_p$. Technically, PhishDecloaker operates on the screenshot of $w_p$ to avoid potential code obfuscation, which consists of three steps:

**Step 1. CAPTCHA Detection** (Section 4.1). We identify the CAPTCHA instance on the webpage by formulating it as an object detection problem in computer vision. We denote the detected CAPTCHA as $c$.

**Step 2. CAPTCHA Recognition** (Section 4.2). With our prepared database of CAPTCHA templates, we determine the type of the CAPTCHA $c$, $t_c$, by matching the CAPTCHA instance $c$ with its best fit in the template database through a learned OCR-aided Metric Learning network. The solution is designed in a similar way as a face recognition problem in computer vision. By this means, PhishDecloaker provides an extensible CAPTCHA recognition framework to flexibly including new CAPTCHA types.

**Step 3. CAPTCHA Solving** (Section 4.3). PhishDecloaker is further equipped with an arsenal of CAPTCHA solvers, as the CAPTCHA solver repository. Given the CAPTCHA type $t_c$, we formulate it as a query to find the most appropriate CAPTCHA solver. The found solver interacts with the webpage $w_p$ to solve the challenge. This interaction can be repeated multiple times to increase the success rate of anti-phishing crawlers and phishing detectors. PhishDecloaker provides such an extensible design to integrate new CAPTCHA solvers in the CAPTCHA solver repository.

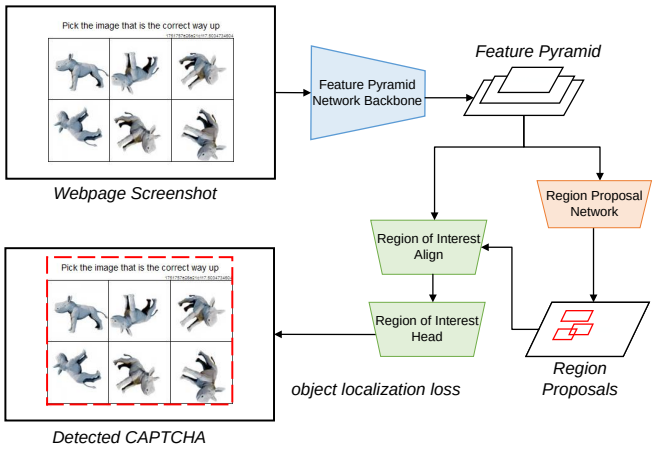



Figure 2: Model architecture of CAPTCHA detection model, consisting of multi-stage processes

### 4.1 CAPTCHA Detection

Given a webpage screenshot, denoted as $\mathcal{S}$, as input, the our CAPTCHA detection model generates object proposals $\mathcal{C}(\mathcal{S}) = \{t | t = \langle x, y, w, h \rangle\}$. As showed in Figure 2, these proposals (in red dashed rectangle) consist of bounding boxes that contains the CAPTCHA region.

Figure 2 illustrates the structure of our detection model. We employ an Object Localization Network (OLN) [30], which is a two-stage network comprising a Region Proposal Network (RPN) stage and a Region of Interest (RoI) stage. Given a webpage screenshot, the backbone network (Feature Pyramid Network) transforms the webpage into a feature pyramid $\mathcal{F} = \{f | f = k \times k, f \in R^2\}$ where $k = \frac{W}{4}, \frac{W}{8}, \frac{W}{16}$.... Specifically, each element in $\mathcal{F}$ is a feature map in the form of a $k \times k$ matrix. Each feature map captures the spatial features of the webpage screenshot in different granularity. Then, the

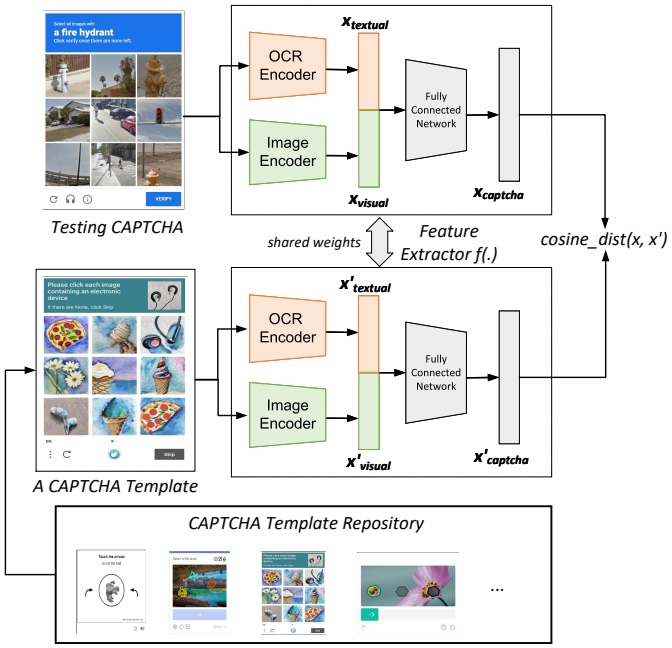



Figure 3: OCR-aided Siamese Model. The CAPTCHA recognition system comprises a feature extractor $f(.)$ and a recognition head. The extractor maps a CAPTCHA image to a low-dimensional embedding $\mathbf{x}_{captcha}$. The recognition head matches the testing CAPTCHA with CAPTCHA templates using cosine distance.

feature pyramid is fed into the Region Proposal Network, generating initial location proposals for the foreground object (i.e., CAPTCHA). These proposals undergo further refinement in the Region of Interest (RoI) network to yield the final bounding boxes.

Different from a conventional object detection model [54] where objects are both detected and classified, we customize our CAPTCHA detector to have only detection functionality. Technically, in comparison to the loss function of (1) object location on $x, y, w$, and $h$ and (2) object classification, we train our CAPTCHA detector with only focus of the object location. The customization allows our model training process to focus all the computation resource on a single optimization objective. This is useful considering that the CAPTCHA containers can be very diverse, which can include task instructions that guide the user in solving the CAPTCHA, the challenge body with visual elements, and user interaction buttons for controlling and engaging with the CAPTCHA. They can take various forms and styles, which may encompass distorted text, images, specific object clicks, or even behavioral cues such as slider dragging.

## 4.2 CAPTCHA Recognition

The CAPTCHA recognition model is designed to map a testing CAPTCHA instance $c$ to its best fit in a prepared CAPTCHA templates $\mathcal{C}_t$ where each element in $\mathcal{C}_t$ is a representative CAPTCHA instance of a CAPTCHA type. Our CAPTCHA recognition model consists of a feature extractor mapping an object proposal (i.e., a CAPTCHA instance) to a feature vector, i.e. $f_\theta(.) : \mathcal{C} \rightarrow \mathbb{R}^n$. We denote the type function $type : \mathcal{C} \rightarrow \mathcal{T}$ which returns the type of a CAPTCHA instance, where $\mathcal{T}$ is the set of CAPTCHA types in $\mathcal{C}_t$. Then, we can select the best fit $c^*$ of a test CAPTCHA instance $c$ by

$$c^* = \arg\max_{c_t \in \mathcal{C}_t} cos(f_\theta(c), f_\theta(c_t))$$

Given a threshold $th$, we can decide the type of $c$ by $type(c^*)$ if $cos(f_\theta(c), f_\theta(c^*)) > th$.

To learn $f_\theta$, we shall address the following challenges:

- **Multi-modal representation learning:** A CAPTCHA challenge contains multi-modal information, including the challenge description in plain text and the challenge body in images.
- **Intra-type diversity:** Challenges within the same CAPTCHA type can differ significantly due to various service vendors or updates to the CAPTCHA pool.
- **Inter-type generalization:** As CAPTCHA technology evolves, new types emerge. The model must be easily adaptable to new types of CAPTCHAs.

To incorporate both textual and visual features into the representation, we introduce a dual-branch architecture for our feature extractor $f_\theta(.)$ (See Figure 3). The architecture consists of: (1) a text encoder, pre-trained on an Optical Character Recognition (OCR) task, and (2) an image encoder, pre-trained on an image classification task. Both encoders takes the CAPTCHA image as input, and produces the respective embeddings $\mathbf{x}_{visual}$ and $\mathbf{x}_{textual}$. These two branches capture distinct yet complementary information. The OCR-based encoder focuses on character-indicative features essential for understanding task instructions. In contrast, the image encoder identifies salient visual patterns, capturing the CAPTCHA's layout and design. A fully-connected projection layer is added to fuse the two modalities with additional non-linearity: $\mathbf{x}_{captcha} = \sigma(W^T[\mathbf{x}_{visual} \oplus \mathbf{x}_{textual}] + \mathbf{b})$.

With the feature extractor in place, the next step is to design the recognition head. A straightforward approach involves adding a Softmax activation and using conventional Cross Entropy loss for model fitting. However, this approach tends to overfit to observed samples, especially when the training set is small, leading to inaccurate predictions for new variants within known categories. Additionally, this approach lacks the flexibility to accommodate new CAPTCHA types during

runtime, as the number of classes must be predefined before inference.

In this work, we employ a Deep Siamese model to address the challenges outlined earlier. The Siamese model aims to learn an embedding space that accurately captures semantic similarities between images. It is trained on pairs of samples, denoted as "positive pair" if they belong to the same class and "negative pairs" if they come from different classes. A loss function ensures that positive pairs are closer in the embedding space than negative pairs. Specifically for CAPTCHAs, our goal is to cluster those of the same type together. During inference, we input the test CAPTCHA along with a set of CAPTCHA *templates* from different classes of CAPTCHAs. We then rank the distances to identify the closest class as the final prediction.

In this manner, we effectively tackle the aforementioned challenges: For the *intra-type diversity* issue, the pairwise training paradigm enables the model to be more sensitive in distinguishing "variations within the class" from "variations relative to other classes". Given an unseen sample from a known class, the model is more inclined to treat it as a variant of a known class rather than a novel class of CAPTCHA. For the *inter-type generalization* issue, accommodating new CAPTCHA types is straightforward: new CAPTCHAs can be easily added to the template database, serving as reference points for future queries.

Technically, during training, we freeze the textual branch and fine-tune all other remaining modules using Sub-center ArcFace loss [14]:

$$L = -\frac{1}{N}\sum_{i=1}^{N}\log\left(\frac{e^{s\cdot(\cos(\theta_{y_i}+m)-1)}}{e^{s\cdot(\cos(\theta_{y_i}+m)-1)}+\sum_{j=1,j\neq y_i}^{C}e^{s\cdot\cos(\theta_j)}}\right) \tag{1}$$

In this setup, we learn a set of parameters representing the embedding centers for each class. The embedding feature $i$ and the center for its ground truth class $y_i$ are considered a "positive pair", while it and the center for another class $j$ form a "negative pair". $\theta_{y_i}$ is the angle between the positive pair, and $\theta_j$ is the angle between the negative pair. This loss function encourages CAPTCHA embeddings to be close to their respective class centers and distant from irrelevant ones. Additionally, to address class imbalance, as some CAPTCHA types are more common than others, we assign class weights to the loss, calculated as $\frac{1}{\log(n_c)}$, where $n_c$ is each class's frequency in the training set.

## 4.3 CAPTCHA Solvers

The types of challenges presented by CAPTCHAs are diverse, as shown in Figure 4. Each type may require a unique skill set, such as object recognition, visual question answering, pattern matching, or orientation identification. To address the problem, we develop an arsenal of CAPTCHA solvers for each supported CAPTCHA type. For some CAPTCHA types (e.g., reCAPTCHA), we adopt the state-of-the-art solvers; while for other important CAPTCHA types (e.g., slider) where no solver is available, we develop our own AI-powered solving solutions. Despite that we do not claim contributions in solving a particular CAPTCHA type, our design is extensible to new CAPTCHA type. Further, our support of four types of CAPTCHA has covered the most common CAPTCHAs.

**reCAPTCHA v2 Solver** Google reCAPTCHA v2 is the most prevalent type of CAPTCHA among the Top 1 Million Sites [11]. We solve the reCAPTCHA v2 challenges by employing an object detection model similar to that of Hossen et al. [27].

**hCaptcha Solver** hCaptcha is an image-based CAPTCHA service similar to reCAPTCHA. However, hCaptcha presents users with more realistic and even AI-generated images. In this work, we address two commonly occurring versions of hCaptcha:

- **Version 1 (Object-Identifying hCaptcha).** Object-Identifying hCaptcha is similar to reCAPTCHA v2 (See Figure 4b). But it offers more complex challenge descriptions, enriched with extra context on styles or relations to other objects (e.g., someone playing football). Hence, we approach the problem as an open-set visual question answering (VQA) model. In a typical VQA task, a model is presented with an image alongside a text-based question about the visual content. The model then generates an answer, which can be a simple "yes" or "no" or a more complex textual answer. We use the OFA architecture [64], which has proven generalizability to out-of-domain images (e.g., anime or synthetic pictures). Given a CAPTCHA challenge specifying the object description as $x$, our OFA model takes the question "Is this a/an $x$?" and each candidate grid, then outputs an answer of either "yes" or "no". Based on the answers, the solver selects all the grids with "yes".
- **Version 2 (Detail-Focused hCaptcha).** Version 2 of hCaptcha is emerging as a new type of challenge (See Figure 4c). This version ask the user to point out specific details within the images. To solve these more advanced challenges, we employ the off-the-shelf tool *hCaptcha Challenger* [25].

**Slider CAPTCHA Solver** Slider-based CAPTCHAs require users to slide a puzzle piece into an empty spot on a background image [55, 70]. In addition to the accuracy of the placement, the CAPTCHAs also analyze the sliding trajectory to detect automated behavior. For example, human users are unlikely to maintain a constant speed throughout the slide.

We design our slider solver with traditional computer vision techniques. First, we identify the the background image and puzzle piece elements from the webpage source code. We then apply pre-processing techniques like Gaussian blurring


(a) reCAPTCHA v2 (b) hCaptcha (Version 1) (c) hCaptcha (Version 2) (d) GeeTest Slider (e) Baidu Rotation

Figure 4: Examples of CAPTCHA challenges of different types.

for denoising and grayscale conversion followed by Sobel edge detection [60] for edge sharpening. Next, we use the puzzle piece as a template for template matching [50], locating a similar region within the background image. Finally, the solver uses easing functions to simulate a human-like dragging trajectory.

**Rotation CAPTCHA Solver** Rotation CAPTCHAs require users to adjust randomly rotated images to their upright orientation [24]. These challenge images usually feature natural and man-made landscapes.

In this work, we treat the image rotation problem as a regression task to predict the current degree of rotation for the challenge image. Once the rotation angle is determined, the solver can interact with the CAPTCHA to correct the orientation. To construct our model, we adopt EfficientNet [62], pretrained on ImageNet. We further fine-tune the model using randomly rotated samples from the Landscape Dataset [21], a community-contributed collection of 7,268 images that depict natural and man-made landscapes. Cosine distance to the ground-truth angle serves as the training loss.

## 4.4 Adversarial Countermeasure

Since PhishDecloaker orchestrates several deep-learning models, it may be vulnerable to adversarial attacks at runtime. We identify two plausible attack scenarios, i.e., system-level attack and model-dependant attack. A system-level attack that introduces blurring, noise, or other obfuscations to CAPTCHA challenges, hindering the models' ability to identify content. This commonly occurs when CAPTCHAs detect suspicious activities from an IP address and present more challenging images. This attack is model-agnostic and does not target any specific model. A model-dependent attack exploits existing gradient-based methods [22, 35, 37, 42, 52] to deliberately perturb inputs and induce incorrect predictions. Both our CAPTCHA detector and recognition models could be susceptible to this type of attack.

To counter the former, we use adversarial training for all our deep-learning models. This approach aims to enhance model robustness by introducing adversarial examples during training. These examples are created by applying random augmentations to the original input data, potentially leading the model to make incorrect predictions. In our work, we consider the following types of augmentations such as Random Mask, Gaussian Noise, and Gaussian Blur. We mix adversarial and clean samples at a ratio of 6:4. To counter the latter, we implement PhishDecloaker with the gradient masking technique as proposed in [32]. Specifically, we replace the ReLU activation function with a step ReLU function defined as $f(x) = \max(0, \alpha \cdot \lceil \frac{x}{\alpha} \rceil)$, where $\alpha$ is the discretization parameter. This renders the activations non-differentiable, effectively zeroing out the gradients.

# 5 Experiments

In this section, we evaluate PhishDecloaker with the following research questions:

- **RQ1**: What is the overall performance of PhishDecloaker in solving CAPTCHAs?
- **RQ2**: What is the performance of the CAPTCHA detection component of PhishDecloaker?
- **RQ3**: What is the performance of the CAPTCHA recognition component of PhishDecloaker (on both seen and unseen CAPTCHA types)?
- **RQ4**: What is the performance of the CAPTCHA solving component of PhishDecloaker?
- **RQ5**: Is PhishDecloaker robust against adversarial attacks on its deep learning models?

In the following, we first introduce the settings for our model training, followed by the experiment settings to evaluate each research question. Due to space limitations, further experimental details are available at [49].

## 5.1 RQ1: Overall Performance of Phishing Detection

### 5.1.1 Experiment Setup

**Subject Phishing Detectors** We select Phishpedia [32] and PhishIntention [33] for their state-of-the-art performance on

Table 2: Phishing detection rate on DynaPD. The percentages are calculated as changes relative to the baseline (No Cloaking). The average runtime overhead is computed for each module (CAPTCHA detection, CAPTCHA recognition and CAPTCHA solver) and concatenated by "+".

| Group | No Cloaking | After Cloaking | | | |
|---|---|---|---|---|---|
| | | reCAPTCHA v2 | hCaptcha | GeeTest Slide | Rotation |
| Phishpedia | 0.73 | 0.00 (↓100%) | 0.00 (↓100%) | 0.00 (↓100%) | 0.00 (↓100%) |
| PhishIntention | 0.53 | 0.00 (↓100%) | 0.00 (↓100%) | 0.00 (↓100%) | 0.00 (↓100%) |
| Phishpedia + PhishDecloaker | 0.73 | 0.57 (↓22.2%) | 0.29 (↓59.8%) | 0.69 (↓5.1%) | 0.61 (↓16.1%) |
| PhishIntention + PhishDecloaker | 0.53 | 0.41 (↓22.3%) | 0.21 (↓59.8%) | 0.50 (↓5.1%) | 0.45 (↓16.0%) |
| Runtime Overhead | - | 0.13 + 0.05 + 44.17 | 0.12 + 0.05 + 8.81 | 0.13 + 0.06 + 5.12 | 0.13 + 0.08 + 5.01 |

detecting zero-day phishing websites. Following the instructions of both detectors, we use a reference list of 277 phishing targets (i.e., Facebook, Bank of America, etc).

**Phishing Dataset** We apply our hardening framework, Cloaken, to the DynaPD dataset [34]. The DynaPD dataset comprises approximately 6K deployable, interactable phishing kits, providing a replicable environment to study CAPTCHA cloaking on phishing kits. Due to limitations in the reference lists of phishing detectors [32, 33], we filter out phishing kits targeting sites not included in the reference list. This filtering results in a dataset of 2,960 phishing kits for our study. Cloaken cloaks each phishing kit with 4 CAPTCHA instances under the category of reCAPTCHA, hCAPTCHA, slider, and rotation, none of these CAPTCHA instances are used for training the models.

**Measurement** We evaluate whether PhishDecloaker can help the phishing detectors to recover its access to the phishing content. Specifically, we evaluate detection rate of a phishing detector on DynaPD, $r_1$, its detection rate on the different types of cloaked phishing website variants, $r_2$, and its detection rate after equipped with PhishDecloaker, $r_3$. Note that, PhishDecloaker is not designed for improving the precision of existing phishing detectors, thus we only evaluate the recall measurement in the study.

**Environment** We use Chrome version 114 [23] and ensure a clean browser state for each session, i.e., with no caches or cookies preserved between consecutive requests. To conceal any indications of a headless browser and automation, we modify the requests and web browser characteristics, such as customizing User-Agent headers and adjusting to the Navigator object properties, as well as modifying to WebGL vendor. All solvers operate from a single IP address and machine with 20 CPU cores, 125G memory, and A100 GPU.

### 5.1.2 Results

Table 2 shows the overall experiment results. On the 2.9K phishing websites without any cloaking, Phishpedia and PhishIntention achieve detection rates of 72.6% and 53.0% respectively. Any CAPTCHA-cloaked variants can compromise their effectiveness, dropping the detection rate to 0%. In contrast, PhishDecloaker can recover their detection rate back to about 78% on reCAPTCHA, 40% on hCAPTCHA, 95% on slider CAPTCHA, and 84% on rotation CAPTCHA. Generally, PhishDecloaker is effective in recovering the phishing detection rate with acceptable runtime cost. Note that, the main overhead lies in CAPTCHA solving. Particularly, solving a reCAPTCHA instance takes on average 44.17 seconds. The reason lies in that reCAPTCHA can repetitively generate new pictures (e.g., motorcycle or bicycle) and ask the user to select the appropriate ones according to its question (e.g., *pick up all the images including bicycle*). The iterative interaction takes a long runtime overhead.

Next, we further investigate and categorize the CAPTCHAs which PhishDecloaker cannot address. The reason lies in as follows.

**Incapability of the off-the-shelf solvers** We observe that the off-the-shelf solvers (e.g., hCAPTCHA solver) have their performance limit. Figure 5 shows an example where PhishDecloaker successfully detect and recognize the hCAPTCHA type but the hCAPTCHA solver fails to solve the challenge of *pick up all the images with bracelet*. Our investigation shows that the visual-text model mistakenly recognize earrings as bracelet, which are of similar visual semantics. A potential remedy is to retrain the model to further distinguish the embedding space of the model. We will discuss the AI incapability in Section 6.

**Human authentication beyond CAPTCHA** Further, we find that some CAPTCHA such as reCAPTCHA v2 authenticates a human visit beyond an interactive challenge. The authentication may include clicking pattern analysis and browser fingerprints checks, which will block our visits even if the

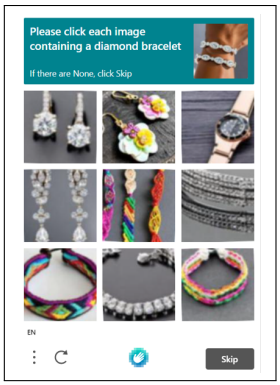



Figure 5: an instance of hCaptcha (Version 1) which cannot be solved by PhishDecloaker

(a) A solved instance of rotation CAPTCHA

(b) An unsolved instance of rotation CAPTCHA

Figure 6: Examples of failure cases in solving CAPTCHAs.

challenge is correctly solved. Addressing the human authentication other than CAPTCHA is beyond the scope of this research. Nevertheless, our future work will address such a problem of hybrid human authentication.

**The restriction of training dataset** Finally, we find that our customized solvers might be limited by our training dataset. In general, our regression model (see Section 4.3) learns upright orientation for different pictures. Therefore, if a picture to be rotated is deviated from the training dataset, the model might fail to predict its rotation degree effectively. As showed in Figure 6, Figure 6a and Figure 6b manifest different genres of pictures, which induce the performance difference. Despite that we do not claim the contribution of solving a particular CAPTCHA instance, we will focus on investigating more sophisticated computer vision solutions in our security application.

## 5.2 RQ2: CAPTCHA Detection

### 5.2.1 Experiment Setup

**Dataset Collection** To collect the training dataset of detecting CAPTCHA, we adopt XDriver [15] to crawl the websites listed in the Alexa top 1-million websites. It automatically locates the forms on the page, fills in all form inputs with simulated data, and submits the form in order to trigger CAPTCHAs. It then captures screenshots of these pages, which we manually annotate to identify the bounding boxes for any CAPTCHAs present. Due to ethical and security considerations, we strictly limit our crawling to a single instance per website, with a maximum depth of 2. Over two weeks, we collected and labeled 1,764 webpage screenshots containing CAPTCHAs. We employ data augmentation to enrich our dataset with additional synthetic samples, bringing the total to 10,680 webpage screenshots. Examples of the synthetic samples generated are shown in Appendix **??**. We perform a 9:1 train-test split, where 9,612 samples are for training and 1,068 samples for testing.

Table 3: Performance for CAPTCHA Detection.

| Model | mAP IOU@0.5:0.95 | mAR IOU@0.5:0.95 |
|---|---|---|
| Faster R-CNN | **0.936** | 0.949 |
| OLN | 0.922 | **0.973** |

**Training Settings** We use the training framework provided by the authors of [30], which is built upon OpenMMLab Detection Toolbox [41]. The OLN object detection model uses Faster-RCNN pre-trained on ImageNet as its backbone, with the RPN and RoI head modified as described in Section 4.1. We train the model for 8 epochs, with a batch size of 2 per GPU, using Stochastic Gradient Descent with a learning rate of 0.02 and momentum of 0.9.

**Measurement** We use the mean average precision (mAP) and mean average recall (mAR) to evaluate the performance, which are the standard metrics for evaluating the completeness and redundancy of the reported objects in object detection tasks [30, 54]. The mAP and mAR are computed over IoU thresholds ranging from 0.5 to 0.95.

### 5.2.2 Results

The results are shown in Table 3, we have tried two different object detection models, Faster R-CNN and OLN. Both models achieve high detection performance, but OLN can report CAPTCHAs with higher recall. Therefore, we choose to use OLN for our CAPTCHA detector in the follow-up experiments.

## 5.3 RQ3: CAPTCHA Recognition

### 5.3.1 Experiment Setup

**Dataset Collection** We have collected a total of 6,612 CAPTCHA samples spanning 38 classes, sourced from demo websites (e.g., NetEase, Tencent, Arkose Labs), official API

Table 4: Performance for CAPTCHA Recognition on Open-set CAPTCHAs.

| Class | Precision | Recall | F1-Score |
|---|---|---|---|
| arkose select2 | 0.93 | 0.91 | 0.92 |
| capycaptcha drag | 0.88 | 0.58 | 0.70 |
| dicecaptcha qa | 0.97 | 0.68 | 0.80 |
| funcaptcha select | 0.99 | 0.87 | 0.93 |
| funcaptcha select2 | 0.98 | 0.48 | 0.64 |
| funcaptcha select3 | 0.88 | 0.52 | 0.65 |
| funcaptcha select4 | 0.62 | 0.91 | 0.74 |
| funcaptcha select5 | 1.00 | 0.53 | 0.69 |
| funcaptcha select6 | 0.88 | 0.72 | 0.79 |
| keycaptcha drag | 0.93 | 0.75 | 0.83 |
| mtcaptcha text | 0.46 | 0.63 | 0.53 |
| average | 0.86 | 0.69 | 0.75 |

keys provided by vendors (e.g., Google, hCaptcha, GeeTest), and open-source community datasets. During training, we employed a 9:1 train-test split, allocating 5,950 samples for training and 662 samples for testing. This dataset serves as the template database at deployment time.

**Training Settings** As for the feature extractor, the visual branch employs a ResNet-50 model pre-trained on ImageNet with its classification head removed. This branch takes a resized CAPTCHA region of dimensions $224 \times 224$ as input and outputs the visual embedding. The textual branch is adapted from EasyOCR [29]. It utilizes a Character-Region Awareness for Text (CRAFT) model [9] pre-trained on SynthText for bounding box detection and a Convolutional Long Short Term Memory (CLSTM) [56] model pre-trained with the STR framework [8] for textual embedding projection. During training, we freeze the textual branch and fine-tune all other branches using Sub-center ArcFace [14] as described in Section 4.2. We train the model for 100 epochs, with a batch size of 2 per GPU, using Stochastic Gradient Descent with a learning rate of 0.02 and a momentum of 0.9.

**Measurement** We evaluate the training and testing accuracy of our model in recognizing a particular CAPTCHA type. In addition, we evaluate the generalizability of our model by whether our model can recognize the new CAPTCHA types without retraining the model. We argue that utilizing deep Siamese learning for CAPTCHA recognition can improve performance in open-set scenarios. To validate this, we first collect an additional 11 CAPTCHA types not present in the training datasets of our CAPTCHA detection and recognition models. These newly acquired CAPTCHAs are superimposed onto random webpage screenshots to create open-set test samples. We then update PhishDecloaker's template database with references from these new CAPTCHA classes and assess the system's ability to correctly classify these novel CAPTCHAs.

#### 5.3.2 Results

Table 4 shows the performance of our model on the training and testing dataset. Table 4 presents our results on the open-set dataset. Our system yields satisfactory performance in correctly identifying unseen CAPTCHA types, with an average precision of 86.0% and an average recall of 69%.

### 5.4 RQ4: CAPTCHA Solving

#### 5.4.1 Experiment Setup

**CAPTCHA Benchmark** Our CAPTCHA benchmarking dataset includes reCAPTCHA v2, hCAPTCHA, slider CAPTCHA, and rotation CAPTCHA. For reCAPTCHA v2, hCAPTCHA (V1), and hCAPTCHA (V2), we test different difficulty levels: easy, moderate, and difficult. As for slider CAPTCHA, we include three versions implemented by GeeTest [20], Tencent [12], and NetEase [44]. Lastly, we select the Baidu version for evaluating rotation CAPTCHA. In general, we generate 200 CAPTCHA variants for each version, except for hCAPTCHA (V1). The reason is that hCAPTCHA outputs V1 CAPTCHA much less frequently than V2 CAPTCHA. Given its lower frequency, we generate 50 hCAPTCHA (V1) variants. In total, we evaluated 13 CAPTCHA versions in the study.

**Measurement** For reCAPTCHA v2, hCaptcha, all slider CAPTCHA variations, and Baidu rotation CAPTCHA, we determine their success rate by calculating the proportion of successfully resolved CAPTCHA sessions compared to the total number of requested CAPTCHA sessions. To elaborate, during each CAPTCHA session, the solver may encounter one or more consecutive CAPTCHA challenges. In order for a CAPTCHA session to be considered as successfully solved, the solver must successfully complete all presented challenges and receive a confirmation of success (e.g., a green checkmark in reCAPTCHA v2).

Additionally, to further analyze our rotation CAPTCHA solver, we evaluate it on the test set in the Landscape Dataset. The test samples are randomly rotated between 0 to 360 degrees. We use mean angular error (MAE) as the evaluation metric. Given a batch of $N$ predicted angles $\theta_i$ and their ground truth (rotated) angles $\hat{\theta}_i$, MAE is defined as follows:

$$\text{MAE} = \frac{1}{N}\sum_{i=1}^{N}\Big(180 - \big||\theta_i - \hat{\theta}_i| - 180\big|\Big) \tag{2}$$

This metric quantifies the average angular discrepancy between the predicted and actual angles, providing insight into the accuracy of the solver's rotational predictions.

Table 5: CAPTCHA Metrics and Detection Rates

| CAPTCHA | Category | Solving rate |
|---|---|---|
| reCAPTCHA v2 | Easy | 75.5% |
| | Moderate | 74.0% |
| | Difficult | 35.0% |
| hCaptcha (V1) | Easy | 92.0% |
| | Moderate | 90.0% |
| | Difficult | 10.0% |
| hCaptcha (V2) | Easy | 83.5% |
| | Moderate | 93.0% |
| | Difficult | 87.0% |
| Slider | GeeTest | 95.0% |
| | Tencent | 89.0% |
| | NetEase | 100.0% |
| Rotation | Baidu | 74.5% |

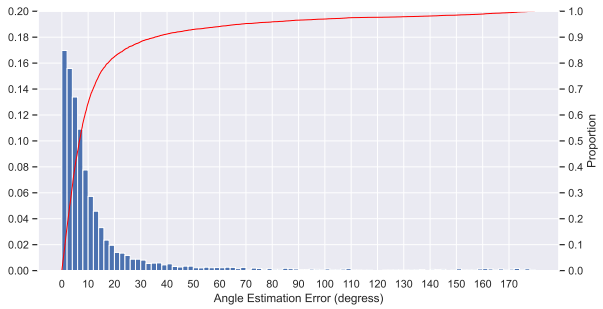



Figure 7: Histogram of angular errors and their cumulative distribution (red line) for the regression model on the test set in Landscape Dataset.

**Results** For reCAPTCHA and hCaptcha, we observe that our solvers perform well on easy and moderate challenges. Performance degrades for difficult cases, with reasons discussed in Section 5.1. Using the template matching algorithm, the slider CAPTCHA solver performs uniformly well across three different service providers, because that the algorithm is robust against noise and distortion in the background images.

The rotation solver achieves a mean angular error of 15.62. Figure 7 displays a histogram of the angular errors along with its cumulative distribution. We observe that more than 90% of the test samples exhibit an angular error of less than 35 degrees, and the histogram follows a long-tail distribution. This indicates that the solver was able to reorient the majority of images. We then conducted a test using Baidu's rotation CAPTCHA service, limiting the number of attempts to 200. The solver achieves a solving rate of 74.5%.

## 5.5 RQ5: Robustness Against Adversaries

In this section, we assess PhishDecloaker's robustness to evasion attacks by generating custom CAPTCHA images that target specific components of PhishDecloaker. We then evaluate the system's overall resilience.

### 5.5.1 Adversaries

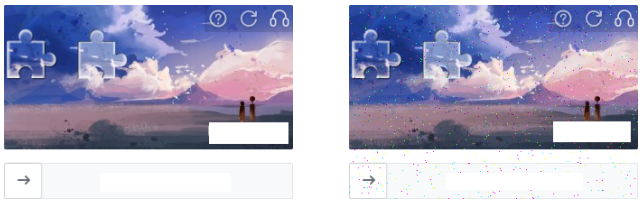
We model our adversary as a phisher with no constraints in time and computing resources to deploy evasion attacks. The adversary's aim is to create a CAPTCHA-cloaking page that remains undetected by PhishDecloaker. To this end, we conduct adversarial attacks on detection component, recognition component, and the solving component, with the assumption that the adversary possesses *perfect* knowledge of PhishDecloaker's system design.

**Attacks on CAPTCHA Detection** We conduct DPatch attack [35] to compromise the CAPTCHA detection component of PhishDecloaker. DPatch generates adversarial patches that can be applied to a webpage. In our case, the patches are untargeted, To create untargeted patches, DPatch finds a patch pattern $\vec{P}_u$ that maximize the loss of the object detector to the true class label $\vec{y}$ and bounding box label $\vec{B}$ when the patch is applied to a webpage screenshot $x$ using "apply" function $A$, as shown in equation 3 [35]. The apply function $A(x,P)$ means adding patch $P$ onto webpage screenshot $x$. As a result, the CAPTCHA detection component will fail to locate the correct region containing CAPTCHAs

$$\vec{P}_u = \arg\max_P \mathbb{E}_x \left[ L(A(x,P);\vec{y},\vec{B}) \right] \tag{3}$$

**Attacks on CAPTCHA Recognition** We conduct adversarial attacks on CAPTCHA images including Fast Gradient Sign Method (FGSM) [22], Jacobian Saliency Map Attack (JSMA) [52], Projected Gradient Descent (PGD) [37], and DeepFool [42] to compromise the CAPTCHA recognition component of PhishDecloaker. In the attack, we assume that the attacker can access the white-box model but cannot poison or modify the on-deployed version of the model.

**Augmentation Attacks** Additionally, we conduct generic image augmentation attacks on PhishDecloaker by performing 6 types of transformations on the CAPTCHA image and overlay the transformed CAPTCHA onto random webpage screenshots. Specifically, they are Random Stretch, Gaussian Noise, Random Crop, Random Mask, Salt and Pepper, and Gaussian Blur. Figure 8 shows an example of Salt and Pepper transformation. Those augmentations are commonly found in phishing webpages [6]. Different from generating the adversarial images for CAPTCHA detection and recognition component, those adversarial samples are visible.



(a) Before Transformation (b) After Transformation

Figure 8: Salt and Pepper Transformation.

**Dataset** We apply all the above attacks on the dataset described in Section 5.3.

**Measurement** We measure the performance of PhishDecloaker before and after attacks in terms of the accuracy of the results of a pipelined CAPTCHA detection and CAPTCHA recognition. We evaluate the robustness against the adversaries by the perturbation of overall CAPTCHA recognition rate before and after the attack. We do not evaluate the solving accuracy as we cannot change the online CAPTCHAs.

### 5.5.2 Results

Table 6 and Table 7 show that the accuracy loss under attacks. We can see that our adopted gradient masking technique is effective in defending the state-of-the-art gradient-based adversarial attack on the deep learning models, i.e., CAPTCHA detection and recognition models.

Table 6: The robustness of CAPTCHA recognition model against diverse adversarial attack.

| Attack | Accuracy (no Def.) | Accuracy (with Def.) |
|---|---|---|
| No Attack | 0.97 | 1.00 |
| JSMA | 0.50 (-49.1%) | 1.00 (-0.0%) |
| PGD | 0.12 (-87.7%) | 1.00 (-0.0%) |
| DeepFool | 0.07 (-93.0%) | 1.00 (-0.0%) |
| FGSM | 0.06 (-94.2%) | 1.00 (-0.0%) |

Table 7: The robustness of CAPTCHA detection model against adversarial attack.

| Attack | mAP (no Def.) | mAP (with Def.) |
|---|---|---|
| No Attack | 97.70 | 91.55 |
| DPatch | 0.28 (-71.7%) | 88.55 (-3.3%) |

Further, Table 8 show the performance of PhishDecloaker to solve augmented CAPTCHA with attacks such as Random Stretch, Gaussian Noise, etc. We can see that the transformation-based attacks can be effectively defended by the adversarial training. Note that the adversarial training only include Random Mask, Gaussian Noise, and Gaussian Blur, the model does not see the transformation of Random Stretch, Random Mask, and Salt and Pepper. The results indicate that a type of learned transformation can be generalized to the other unseen transformation.

Table 8: The robustness of PhishDecloaker against augmentation attacks.

| Attack | Accuracy (no Def.) | Accuracy (with Def.) |
|---|---|---|
| No Attack | 0.97 | 1.00 |
| Random Stretch | 0.95 (-1.9%) | 0.96 (-4.0%) |
| Gaussian Noise | 0.87 (-10.2%) | 0.94 (-6.0%) |
| Random Crop | 0.82 (-15.3%) | 0.83 (-17.0%) |
| Random Mask | 0.76 (-21.5%) | 0.90 (-10.0%) |
| Salt and Pepper | 0.33 (-66.4%) | 0.92 (-8.0%) |
| Gaussian Blur | 0.18 (-82.0%) | 0.93 (-7.0%) |

## 6 Discussion

**Limitations** Despite that PhishDecloaker framework is extensible to include new CAPTCHA types in terms of CAPTCHA detection and recognition, it is limited by the number of its supported CAPTCHA solvers. In this work, we design four types of CAPTCHA solvers, i.e., reCAPTCHA, hCaptcha, slider CAPTCHA, and rotation CAPTCHA, taking a market share of 98.9%, there are still many uncovered types of CAPTCHAs.

Thus, when PhishDecloaker encounters a rare CAPTCHA for which it does not have a corresponding solver in its repository, we offer two suggestions. First, PhishDecloaker can integrate with existing CAPTCHA solving services (e.g., 2Captcha, AnyCaptcha, Capsolver, Death By Captcha, etc.). These services are CAPTCHA farms that rely on manual labour. Each service provides unique API endpoints for different CAPTCHA types. In such cases, Phishdecloaker can identify the CAPTCHA type and use the corresponding API. Second, PhishDecloaker can be equipped with a notification system and defer unsolvable CAPTCHAs to an in-house human operator. Nevertheless, in the era of General AI, we expect that emerging AI solutions can further empower PhishDecloaker to achieve improved performance.

**Ethical Considerations** Our hardening experiment with Cloaken might involve submitting fake phishing reports to anti-phishing entities, there are concerns about the resources and time that security web crawlers may expend when visiting the generated token URLs. We contend that our approach of submitting token URLs to anti-phishing entities is similar to prior works [3, 45, 46], and the same concerns have already

been raised and addressed in [3]. Regardless, to limit the potentially negative impacts of our hardening experiment, we restrict our submissions to 5 token URLs per anti-phishing entity per day. As for the token URLs submitted during the hardening experiment, we try to limit its exposure to other visitors unrelated to our study by only submitting them to anti-phishing entities. As a precautionary measure, we also made sure that the content served behind the cloaking page are non-functional and benign (i.e., do not contain credential-taking web forms).

**Future Work** Recently, vision-language foundation models have demonstrated extraordinary emergent abilities on web navigation tasks [17]. These models enable transformative generalization and are capable of solving wide ranges of interactive decision making problems in the wild [43]. Hence, it is possible to study the feasibility of these models as generalized CAPTCHA solvers. The models can be trained with OCR text and CAPTCHA widget image as input, and outputs instructions based on standard protocols (e.g., WebDriver API, DevTools Protocol, etc.) for web interaction.

## 7 Related Work

**Phishing Detection** Conventional phishing detection systems such as SmartScreen, Google Safe Browsing, and OpenPhish rely on blacklists, which are updated through user reports, automatic crawling, and manual verification. However, this method is limited by delays in list updates and frequently misses short-lived phishing campaigns [48].

To automate verification, feature-engineering-based solutions [13, 16, 19, 31, 36, 63, 66] use feature extraction and classification techniques, focusing on HTML code, URLs, domains, and screenshots. Despite their utility, these solutions are inflexible and susceptible to code obfuscation, resulting in rapid data obsolescence. To overcome these limitations, reference-based solutions [2,5,32–34,39] employ deep-vision techniques to compare the representations of a phishing page against a pre-defined reference list, determining its target brand. These approaches are both extensible and explainable, advancing the state-of-the-art in phishing detection.

**Cloaking** Phishing websites use advanced cloaking techniques to evade detection [4, 38, 47, 61, 67, 68]. Two main types exist: server-side and client-side cloaking. Server-side cloaking identifies users via HTTP requests, often using .htaccess or PHP scripts [10, 38, 45, 46, 48, 68]. It employs IP and keyword blacklists, geolocation, and user-agents to filter traffic. Countermeasures include multiple visits with spoofed IPs and user-agents [3, 28, 32, 33, 67].

Client-side cloaking works within browsers, leveraging various methods. It employs browser fingerprinting and user interaction, such as pop-ups or CAPTCHAs [3, 38, 61, 67]. It also manipulates bot behavior to delay loading times [67]. Despite its rising popularity, client-side cloaking challenges anti-phishing engines [38, 46]. Though lacking a systematic approach, some advances have been made. For example, Crawlphish uses JavaScript force execution to detect client-side cloaking but focuses more on filtering than accurate detection [67].

**CAPTCHA Solving** Deep learning models have been used to solve specific CAPTCHA types [18, 27, 58, 69, 70], but no system exists for automatically identifying arbitrary CAPTCHAs, consistent with [61]. Some ad hoc solvers include: Sivakorn et al. [58] tackled Google reCAPTCHA v2 by exploiting challenge instructions. They used image annotation services and Word2Vec to match tags with challenge text. Hossen et al. [27] employed a custom object detection model to recognize the objects present in the challenge images. To enhance robustness, they also used data augmentation techniques like Gaussian noise and brightness modulation during training. For slider-based CAPTCHAs, Zhao et al. [70] developed an algorithm to match background and target images to identify the puzzle region. Wu et al. [65] used object detection models to localize puzzle regions and generated trajectories via curve fitting.

Different from the above solution, PhishDecloaker detects, recognizes, and then solves the CAPTCHAs with either integrated and developed CAPTCHA.

## 8 Conclusion

In this work, we raised the issue of phishing websites employing client-side cloaking with CAPTCHAs. Traditional web crawlers used by anti-phishing entities were unable to bypass CAPTCHA-based cloaking. We further confirmed through our hardening experiment, during which we submitted fake phishing reports to anti-phishing entities and observed how web crawlers interacted with benign websites utilizing CAPTCHA-based client-side cloaking. The observation motivates us to develop PhishDecloaker to automatically detect, recognize, and solve CAPTCHAs.

Our PhishDecloaker system incorporate four types of CAPTCHA solvers, which targets reCAPTCHA v2, hCaptcha, rotation and slider-based CAPTCHAs. Our experiments indicates that the use of a classification-free detector (OLN) and an OCR-based metric learning network are effective for CAPTCHA detection and recognition. Besides, PhishDecloaker's satisfactory performance in open-set recognition underscores its adaptability and ability to generalize beyond CAPTCHAs in the training data. Furthermore, by integrating the system with Phishpedia and PhishIntention, two existing vision-based phishing detectors, we have shown that our system can help reveal phishing webpages with CAPTCHA-based cloaking.

## References


[1] Moataz AbdelKhalek and Ahmed Shosha. Jsdes: An automated de-obfuscation system for malicious javascript. In *proceedings of the 12th International Conference on Availability, Reliability and Security*, pages 1–13, 2017.

[2] Sahar Abdelnabi, Katharina Krombholz, and Mario Fritz. Visualphishnet: Zero-day phishing website detection by visual similarity. In *Proceedings of the 2020 ACM SIGSAC conference on computer and communications security*, pages 1681–1698, 2020.

[3] Bhupendra Acharya and Phani Vadrevu. {PhishPrint}: evading phishing detection crawlers by prior profiling. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 3775–3792, 2021.

[4] Bhupendra Acharya and Phani Vadrevu. A human in every ape: Delineating and evaluating the human analysis systems of anti-phishing entities. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 156–177. Springer, 2022.

[5] Sadia Afroz and Rachel Greenstadt. Phishzoo: Detecting phishing websites by looking at them. In *2011 IEEE fifth international conference on semantic computing*, pages 368–375. IEEE, 2011.

[6] Giovanni Apruzzese, Hyrum S Anderson, Savino Dambra, David Freeman, Fabio Pierazzi, and Kevin Roundy. "real attackers don't compute gradients": Bridging the gap between adversarial ml research and practice. In *2023 IEEE Conference on Secure and Trustworthy Machine Learning (SaTML)*, pages 339–364. IEEE, 2023.

[7] AWPG. Phishing activity trends report, 4th quarter 2022, 2022. Accessed: 2023-10-11, URL: https://docs.apwg.org/reports/apwg_trends_report_q3_2022.pdf.

[8] Jeonghun Baek, Geewook Kim, Junyeop Lee, Sungrae Park, Dongyoon Han, Sangdoo Yun, Seong Joon Oh, and Hwalsuk Lee. What is wrong with scene text recognition model comparisons? dataset and model analysis. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 4715–4723, 2019.

[9] Youngmin Baek, Bado Lee, Dongyoon Han, Sangdoo Yun, and Hwalsuk Lee. Character region awareness for text detection. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 9365–9374, 2019.

[10] Hugo Bijmans, Tim Booij, Anneke Schwedersky, Aria Nedgabat, and Rolf van Wegberg. Catching phishers by their bait: Investigating the dutch phishing landscape through phishing kit detection. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 3757–3774, 2021.

[11] BuiltWith®. Captcha usage distribution in the top 1 million sites, 2023. Accessed: 2023-10-11, URL: https://trends.builtwith.com/widgets/captcha.

[12] Tencent Cloud. Tencent captcha. https://www.tencentcloud.com/products/captcha, 2023.

[13] Marco Cova, Christopher Kruegel, and Giovanni Vigna. There is no free phish: An analysis of "free" and live phishing kits. *WOOT*, 8:1–8, 2008.

[14] Jiankang Deng, Jia Guo, Tongliang Liu, Mingming Gong, and Stefanos Zafeiriou. Sub-center arcface: Boosting face recognition by large-scale noisy web faces. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XI 16*, pages 741–757. Springer, 2020.

[15] Kostas Drakonakis, Sotiris Ioannidis, and Jason Polakis. The cookie hunter: Automated black-box auditing for web authentication and authorization flaws. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 1953–1970, 2020.

[16] Birhanu Eshete, Adolfo Villafiorita, and Komminist Weldemariam. Binspect: Holistic analysis and detection of malicious web pages. In *Security and Privacy in Communication Networks: 8th International ICST Conference, SecureComm 2012, Padua, Italy, September 3-5, 2012. Revised Selected Papers 8*, pages 149–166. Springer, 2013.

[17] Hiroki Furuta, Ofir Nachum, Kuang-Huei Lee, Yutaka Matsuo, Shixiang Shane Gu, and Izzeddin Gur. Multimodal web navigation with instruction-finetuned foundation models. *arXiv preprint arXiv:2305.11854*, 2023.

[18] Yipeng Gao, Haichang Gao, Sainan Luo, Yang Zi, Shudong Zhang, Wenjie Mao, Ping Wang, Yulong Shen, and Jeff Yan. Research on the security of visual reasoning {CAPTCHA}. In *30th USENIX security symposium (USENIX security 21)*, pages 3291–3308, 2021.

[19] Sujata Garera, Niels Provos, Monica Chew, and Aviel D Rubin. A framework for detection and measurement of phishing attacks. In *Proceedings of the 2007 ACM workshop on Recurring malcode*, pages 1–8, 2007.

[20] Geetest. Geetest captcha. https://www.geetest.com/en/, 2023.

[21] Github. Landscape dataset. https://github.com/koishi70/Landscape-Dataset.

[22] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[23] Google. Google chrome 114. https://developer.chrome.com/en/blog/new-in-chrome-114/.

[24] R Gossweiler, M Kamvar, and S Baluja. A captcha based on image orientation. *Proc. ACM*, 2009.

[25] hCaptcha Challenger Contributors. hCaptcha Challenger, 2022. URL: https://github.com/QIN2DIM/hcaptcha-challenger.

[26] Dorjan Hitaj, Briland Hitaj, Sushil Jajodia, and Luigi V Mancini. Capture the bot: Using adversarial examples to improve captcha robustness to bot attacks. *IEEE Intelligent Systems*, 36(5):104–112, 2020.

[27] Md Imran Hossen, Yazhou Tu, Md Fazle Rabby, Md Nazmul Islam, Hui Cao, and Xiali Hei. An object detection based solver for {Google's} image {reCAPTCHA} v2. In *23rd international symposium on research in attacks, intrusions and defenses (RAID 2020)*, pages 269–284, 2020.

[28] Luca Invernizzi, Kurt Thomas, Alexandros Kapravelos, Oxana Comanescu, Jean-Michel Picod, and Elie Bursztein. Cloak of visibility: Detecting when machines browse a different web. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 743–758. IEEE, 2016.

[29] JaidedAI. Easyocr, 2020. Accessed: 2023-08-04, URL: https://github.com/JaidedAI/EasyOCR/tree/master.

[30] Dahun Kim, Tsung-Yi Lin, Anelia Angelova, In So Kweon, and Weicheng Kuo. Learning open-world object proposals without learning to classify. *IEEE Robotics and Automation Letters*, 7(2):5453–5460, 2022.

[31] Yukun Li, Zhenguo Yang, Xu Chen, Huaping Yuan, and Wenyin Liu. A stacking model using url and html features for phishing webpage detection. *Future Generation Computer Systems*, 94:27–39, 2019.

[32] Yun Lin, Ruofan Liu, Dinil Mon Divakaran, Jun Yang Ng, Qing Zhou Chan, Yiwen Lu, Yuxuan Si, Fan Zhang, and Jin Song Dong. Phishpedia: A hybrid deep learning based approach to visually identify phishing webpages. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 3793–3810, 2021.

[33] Ruofan Liu, Yun Lin, Xianglin Yang, Siang Hwee Ng, Dinil Mon Divakaran, and Jin Song Dong. Inferring phishing intention via webpage appearance and dynamics: A deep vision based approach. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 1633–1650, 2022.

[34] Ruofan Liu, Yun Lin, Yifan Zhang, Penn Han Lee, and Jin Song Dong. Knowledge expansion and counterfactual interaction for {Reference-Based} phishing detection. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 4139–4156, 2023.

[35] Xin Liu, Huanrui Yang, Ziwei Liu, Linghao Song, Hai Li, and Yiran Chen. Dpatch: An adversarial patch attack on object detectors. *arXiv preprint arXiv:1806.02299*, 2018.

[36] Christian Ludl, Sean McAllister, Engin Kirda, and Christopher Kruegel. On the effectiveness of techniques to detect phishing sites. In *Detection of Intrusions and Malware, and Vulnerability Assessment: 4th International Conference, DIMVA 2007 Lucerne, Switzerland, July 12-13, 2007 Proceedings 4*, pages 20–39. Springer, 2007.

[37] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.

[38] Sourena Maroofi, Maciej Korczyński, and Andrzej Duda. Are you human? resilience of phishing detection to evasion techniques based on human verification. In *Proceedings of the ACM Internet Measurement Conference*, pages 78–86, 2020.

[39] Eric Medvet, Engin Kirda, and Christopher Kruegel. Visual-similarity-based phishing detection. In *Proceedings of the 4th international conference on Security and privacy in communication netowrks*, pages 1–6, 2008.

[40] Trend Micro. Addressing captcha-evading phishing threats with behavior-based ai protection, 2023. Accessed: 2023-10-11, URL: https://www.trendmicro.com/vinfo/id/security/news/internet-of-things/addressing-captcha-evading-phishing-threats-with-behavior-based-ai-protection.

[41] MMDetection Contributors. OpenMMLab Detection Toolbox and Benchmark, 2018. URL: https://github.com/open-mmlab/mmdetection.

[42] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: a simple and accurate method to fool deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2574–2582, 2016.

[43] Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, et al. Webgpt: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332*, 2021.

[44] NetEase. Netease captcha. https://dun.163.com/locale/en, 2023.

[45] Adam Oest, Yeganeh Safaei, Adam Doupé, Gail-Joon Ahn, Brad Wardman, and Kevin 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)*, pages 1344–1361. IEEE, 2019.

[46] Adam Oest, Yeganeh Safaei, Penghui Zhang, Brad Wardman, Kevin Tyers, Yan Shoshitaishvili, and Adam Doupé. {PhishTime}: Continuous longitudinal measurement of the effectiveness of anti-phishing blacklists. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 379–396, 2020.

[47] Adam Oest, Yeganeh Safei, Adam Doupé, Gail-Joon Ahn, Brad Wardman, and Gary Warner. Inside a phisher's mind: Understanding the anti-phishing ecosystem through phishing kit analysis. In *2018 APWG Symposium on Electronic Crime Research (eCrime)*, pages 1–12. IEEE, 2018.

[48] Adam Oest, Penghui Zhang, Brad Wardman, Eric Nunes, Jakub Burgis, Ali Zand, Kurt Thomas, Adam Doupé, and Gail-Joon 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.

[49] Anonymous Website of PhishDecloaker. Phishdecloaker, 2023. Accessed: 2023-10-17, URL: https://sites.google.com/view/phishdecloaker/home.

[50] OpenCV. Template matching. https://docs.opencv.org/4.x/d4/dc6/tutorial_py_template_matching.html.

[51] Margarita Osadchy, Julio Hernandez-Castro, Stuart Gibson, Orr Dunkelman, and Daniel Pérez-Cabo. No bot expects the deepcaptcha! introducing immutable adversarial examples, with applications to captcha generation. *IEEE Transactions on Information Forensics and Security*, 12(11):2640–2653, 2017.

[52] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. In *2016 IEEE European symposium on security and privacy (EuroS&P)*, pages 372–387. IEEE, 2016.

[53] Peng Peng, Limin Yang, Linhai Song, and Gang Wang. Opening the blackbox of virustotal: Analyzing online phishing scan engines. In *Proceedings of the Internet Measurement Conference*, pages 478–485, 2019.

[54] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28, 2015.

[55] Andrew Searles, Yoshimichi Nakatsuka, Ercan Ozturk, Andrew Paverd, Gene Tsudik, and Ai Enkoji. An empirical study & evaluation of modern {CAPTCHAs}. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 3081–3097, 2023.

[56] Baoguang Shi, Xiang Bai, and Cong Yao. An end-to-end trainable neural network for image-based sequence recognition and its application to scene text recognition. *IEEE transactions on pattern analysis and machine intelligence*, 39(11):2298–2304, 2016.

[57] Chenghui Shi, Xiaogang Xu, Shouling Ji, Kai Bu, Jianhai Chen, Raheem Beyah, and Ting Wang. Adversarial captchas. *IEEE transactions on cybernetics*, 52(7):6095–6108, 2021.

[58] Suphannee Sivakorn, Jason Polakis, and Angelos D Keromytis. I'm not a human: Breaking the google recaptcha. *Black Hat*, 14:1–12, 2016.

[59] Philippe Skolka, Cristian-Alexandru Staicu, and Michael Pradel. Anything to hide? studying minified and obfuscated code in the web. In *The world wide web conference*, pages 1735–1746, 2019.

[60] Irwin Sobel. An isotropic 3x3 image gradient operator. *Presentation at Stanford A.I. Project 1968*, 02 2014.

[61] Karthika Subramani, William Melicher, Oleksii Starov, Phani Vadrevu, and Roberto Perdisci. Phishinpatterns: measuring elicited user interactions at scale on phishing websites. In *Proceedings of the 22nd ACM Internet Measurement Conference*, pages 589–604, 2022.

[62] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pages 6105–6114. PMLR, 2019.

[63] Rakesh Verma and Keith Dyer. On the character of phishing urls: Accurate and robust statistical learning classifiers. In *Proceedings of the 5th ACM Conference on Data and Application Security and Privacy*, pages 111–122, 2015.

[64] Peng Wang, An Yang, Rui Men, Junyang Lin, Shuai Bai, Zhikang Li, Jianxin Ma, Chang Zhou, Jingren Zhou,

and Hongxia Yang. Ofa: Unifying architectures, tasks, and modalities through a simple sequence-to-sequence learning framework. In *International Conference on Machine Learning*, pages 23318–23340. PMLR, 2022.

[65] Danni Wu, Jing Qiu, Huiwu Huang, Lihua Yin, Zhaoquan Gu, and Zhihong Tian. Resnet-based slide puzzle captcha automatic response system. In *Artificial Intelligence and Security: 6th International Conference, ICAIS 2020, Hohhot, China, July 17–20, 2020, Proceedings, Part III 6*, pages 140–153. Springer, 2020.

[66] Guang Xiang, Jason Hong, Carolyn P Rose, and Lorrie Cranor. Cantina+ a feature-rich machine learning framework for detecting phishing web sites. *ACM Transactions on Information and System Security (TISSEC)*, 14(2):1–28, 2011.

[67] Penghui Zhang, Adam Oest, Haehyun Cho, Zhibo Sun, RC Johnson, Brad Wardman, Shaown Sarker, Alexandros Kapravelos, Tiffany Bao, Ruoyu Wang, et al. Crawlphish: Large-scale analysis of client-side cloaking techniques in phishing. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 1109–1124. IEEE, 2021.

[68] Penghui Zhang, Zhibo Sun, Sukwha Kyung, Hans Walter Behrens, Zion Leonahenahe Basque, Haehyun Cho, Adam Oest, Ruoyu Wang, Tiffany Bao, Yan Shoshitaishvili, et al. I'm spartacus, no, i'm spartacus: Proactively protecting users from phishing by intentionally triggering cloaking behavior. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 3165–3179, 2022.

[69] Yang Zhang, Haichang Gao, Ge Pei, Sainan Luo, Guoqin Chang, and Nuo Cheng. A survey of research on captcha designing and breaking techniques. In *2019 18th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/13th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*, pages 75–84. IEEE, 2019.

[70] Binbin Zhao, Haiqin Weng, Shouling Ji, Jianhai Chen, Ting Wang, Qinming He, and Reheem Beyah. Towards evaluating the security of real-world deployed image captchas. In *Proceedings of the 11th ACM Workshop on Artificial Intelligence and Security*, pages 85–96, 2018.

# A Appendix

## A.1 Summary of Datasets

**CAPTCHA Detection Dataset** Usage: see 5.2). Contents: 19,680 webpage screenshots, 10,680 with annotated CAPTCHA bounding boxes, 9,000 without.

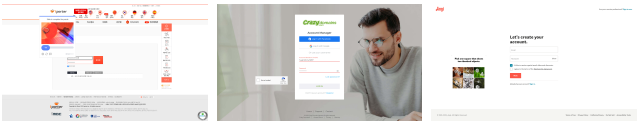


Figure 9: Examples in CAPTCHA Detection Dataset.

**CAPTCHA Recognition Dataset** Usage: see 5.3). Contents: 6,612 CAPTCHA images distributed across 38 classes.


Figure 10: Examples in CAPTCHA Recognition Dataset.

**CAPTCHA Open-set Dataset** Usage: see 5.3. Contents: 1,500 webpage screenshots, all of which have annotated CAPTCHA classes spanning 15 different categories.

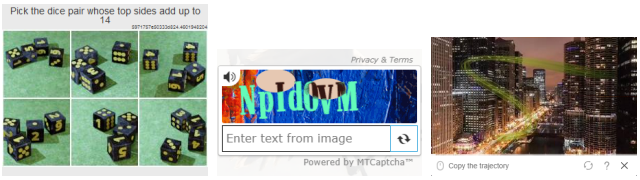



Figure 11: Examples in CAPTCHA Open-set Dataset.