Fight Fire with Fire: Combating Adversarial Patch Attacks using Pattern-randomized Defensive Patches

Jianan Feng, Jiachun Li, Changqing Miao, Jianjun Huang, Wei You, Wenchang Shi and Bin Liang*
School of Information, Renmin University of China, Beijing, China
{*jiananfeng, jclee, miaochangqing, hjj, youwei, wenchang, liangb*}*@ruc.edu.cn*

***Abstract*—Object detection has found extensive applications in various tasks, but it is also susceptible to adversarial patch attacks. The ideal defense should be effective, efficient, easy to deploy, and capable of withstanding adaptive attacks. In this paper, we adopt a counterattack strategy to propose a novel and general methodology for defending adversarial attacks. Two types of defensive patches, *canary* and *woodpecker*, are specially-crafted and injected into the model input to proactively probe or counteract potential adversarial patches. In this manner, adversarial patch attacks can be effectively detected by simply analyzing the model output, without the need to alter the target model. Moreover, we employ *randomized* canary and woodpecker injection patterns to defend against defense-aware attacks. The effectiveness and practicality of the proposed method are demonstrated through comprehensive experiments. The results illustrate that canary and woodpecker achieve high performance, even when confronted with unknown attack methods, while incurring limited time overhead. Furthermore, our method also exhibits sufficient robustness against defense-aware attacks, as evidenced by adaptive attack experiments.**

1. Introduction

In recent years, modern object detection models have demonstrated exceptional performance and are widely utilized in various physical-world tasks, such as pedestrian detection [21], [24], [39] and autonomous driving [7], [68]. Unfortunately, they have been proven to be vulnerable to adversarial patch attacks, involving hiding attacks [19], [50] and misclassification attacks [18], [20]. Clearly, how to safeguard object detectors against adversarial patch attacks is an important and urgent problem.

Several defense methods [8], [12], [25], [33], [61], [63] have been proposed. Unfortunately, the existing methods often require modification of the target model, such as introducing new net layers [8], [33], adding a new model [61], or retraining [12]. In addition, the time overhead of SOTA methods, such as [63], may increase significantly. Furthermore, it is crucial to acknowledge that attackers consistently maintain the upper hand relative to defenders. In many cases, attackers can gain knowledge about the target model and defense techniques through means such as reverse engineering, enabling them to launch defense-aware attacks, as demonstrated in [28], [35], [52]. Attackers can adjust their optimization objectives to generate adaptive adversarial patches that can bypass the defense mechanisms. Therefore, an effective defense method should possess high detection capabilities, ideally without altering the target model, low computational costs, and sufficient robustness against defense-aware attacks.

In this study, we employ the same way of adversarial attacks to counter them. Adversarial patches are essentially additional elements introduced by attackers into the model input to perturb target objects, resulting in their misidentification or disappearance (Figure 1(a)). In a similar manner, we can also proactively introduce specially designed elements, termed **defensive patches**, into the input to defend against adversarial patch attacks (*fight fire with fire*).

In this paper, we propose two types of defensive patches, *canary* and *woodpecker*. They are used to probe or counteract potential adversarial patches, respectively. In this way, potential attacks can be detected by examining or differentially analyzing the model output after incorporating defensive patches. Furthermore, the defensive patches can naturally combine with randomized injection strategies to resist defense-aware attacks.

Canary, as its name suggests, is a fragile object. It is crafted to be easily perturbed by potential adversarial patches, such that it cannot be correctly recognized in an adversarial sample. Intuitively, the canary is preferably to be placed proximal to the attacked object, where it is susceptible to potential adversarial patches. However, determining such a position beforehand is infeasible. Fortunately, we have found that, as a localized attack, adversarial patches cannot entirely erase all information about the target object in the model output. In particular, while the bounding boxes of the attacked object are suppressed by the adversarial patches, making them disregarded by the model, they still maintain observable objectness or class scores. We can identify the suppressed bounding boxes from the output, which are suspected to be related to the attacked object. This allows us to place the canary near the boxes (around or within them), making it more likely to be influenced by the adversarial patches. As illustrated in Figure 1(b), we can ascertain the presence of adversarial patches in the current input by checking whether the model can detect the imported

*Bin Liang is the corresponding author.

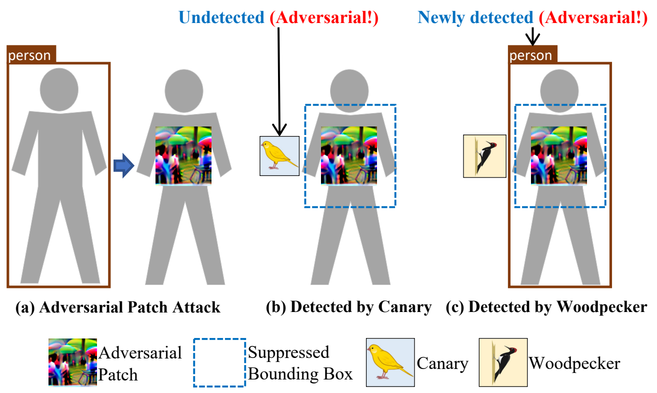

Undetected (Adversarial!)
Newly detected (Adversarial!)
person
person
(a) Adversarial Patch Attack
(b) Detected by Canary
(c) Detected by Woodpecker
Adversarial Patch
Suppressed Bounding Box
Canary
Woodpecker

Figure 1: Basic idea of canary and woodpecker.

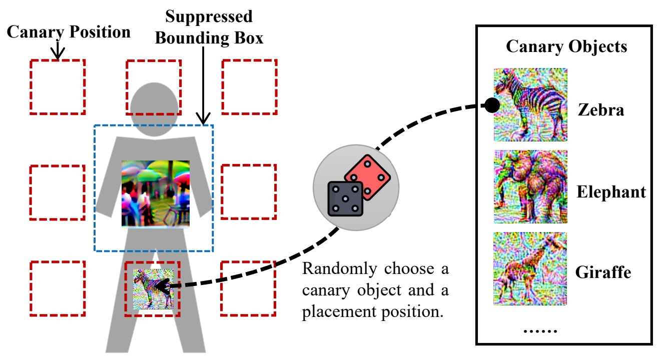


Figure 2: Randomized canary patterns.

canary objects correctly. In this example, canaries are placed to the left of the identified bounding boxes. If any injected canary object is not correctly detected, it indicates that the input is deemed an adversarial sample.

Woodpecker is designed to counteract the influence of adversarial patches on the target object, enabling it to be correctly detected once again. Essentially, adversarial patches aim to subvert the semantics of the target object to deceive object detectors. In response, woodpecker is specifically trained to carry necessary information to repair the compromised object semantics. In a similar way, woodpecker is also placed near identified bounding boxes. In Figure 1(c), we can determine the malicious nature of the input by assessing whether any new objects of interest (e.g., a person) are present in the model output. Specifically, if the woodpecker recovers an object that was misidentified in the original input, the input will be classified as adversarial.

These two defensive patches can be trained and generated for an object class of interest using a limited number of benign and adversarial samples. When applying canary and woodpecker, the target model can remain unchanged. Besides, adopting them does not involve complicated online computation, and thus the overhead is limited.

More importantly, we also draw inspiration from the Stack Cookie [2] and Address Space Layout Randomization (ASLR) [14] techniques used in software security, and enforce a **randomization strategy** when injecting canaries and woodpeckers to enhance robustness. As an active defense measure, defensive patches can combine with various randomization patterns such as patch content and placement positions. Taking canary as an example, as shown in Figure 2, we can generate multiple canary patches containing different objects (e.g., zebra, elephant or giraffe) for each potential canary placement position. For a given input, we can randomly select a canary and place it at a random position near the identified suppressed bounding box (e.g., placing a zebra canary in the lower-middle position), or even place multiple canary patches randomly. As a result, even if the attacker fully understands all details of the canary mechanism, it would be difficult to optimize an adversarial patch that can bypass all canary patterns. Theoretically, the combination of canary objects and placement positions is infinite. In other words, defense-aware attacks will become exceedingly challenging, if not impossible.

The proposed method is evaluated with a comprehensive test set generated from three public datasets [9], [13], [29] and a physical-world one. We conducted experiments on four adversarial patch attacks [18], [19], [20], [50] and five object detectors [23], [41], [42], [55], [56]. It is worth emphasizing that we only use one attack method, i.e., [50] or [19], to generate some digital-world attack samples for training canary and woodpecker. The other attacks are unknown to the obtained canary and woodpecker. Results have shown that our method can achieve a high F1 score and is effective in defending against unknown attacks.

Comparison with existing defense techniques has also shown that, our method can have comparable detection performance in some scenarios and achieve better performance in most scenarios. On benign samples, canary and woodpecker emit low false positive rates. Meanwhile, our method also exhibits a low latency in detection, approximately 0.1 seconds per image.

We also conducted experiments to demonstrate the effectiveness in defending against defense-aware attacks on 688 randomly sampled examples from four datasets. The results show that attackers cannot generate adaptive adversarial patches to completely bypass the defense even if only two constrained randomization strategies are adopted.

Our work makes the following contributions.

- We present a novel and general methodology against adversarial patch attacks. Two kinds of defensive patches, *canary* and *woodpecker*, can be easily applied to unmodified target object detection models, and can effectively defend unknown attack methods.
- We propose a randomization-based defensive patch deployment strategy. The robustness of canary and woodpecker is guaranteed via enforcing randomized patch injection patterns, effectively defending against defense-aware attacks.
- We conduct a comprehensive experiment to evaluate the proposed method, employing over ten thousand digital and physical-world samples. It exhibits good performance on both adversarial and benign examples.

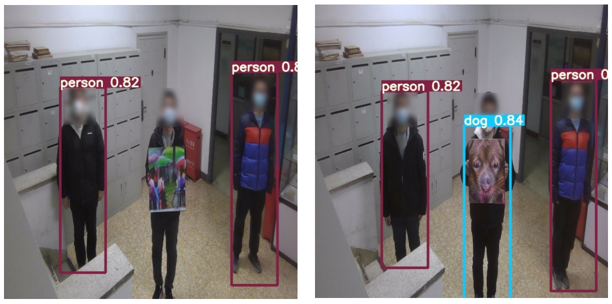


Figure 3: Adversarial patch attacks examples.

2. Background

2.1. Object Detection

The object detection task aims to identify and classify each object in an input image by predicting a set of bounding boxes. Taking YOLOv2 [41] as an example. A bounding box is represented as $box = \{x, y, w, h, obj_{score}, l\}$, where x, y, w, h represent the coordinates and dimensions, obj_{score} represents the objectness score, and l denotes the class probability distribution (object class and its score) of the object in the bounding box. The bounding boxes with low objectness will be filtered out, and then the non-maximum suppression algorithm [37] is used to remove redundant bounding boxes.

Faster R-CNN [42] and YOLO [41], [55], [56] are commonly used object detection models. Despite slight differences, the basic principles of these detectors are similar. Faster R-CNN is a two-stage object detector that first generates a set of candidate regions, called proposals, which may contain objects. These proposals are then filtered to exclude background regions without objects, and the remaining proposals undergo object classification and bounding-box regression. YOLO is a one-stage object detector that predicts object bounding boxes directly, allowing faster detection.

2.2. Adversarial Patch Attack

Adversarial patch attacks [18], [19], [20], [50] aim to deceive object detectors by attaching an adversarial patch onto an object, causing it either to be missed or to be misclassified. The attacker generates an adversarial patch by attaching it to the training images and iteratively optimizing it to evade detection by the target detector or be misclassified as a specific category. Figure 3(a) illustrates the AdvPatch attack [50], which makes the person carrying it undetected; Figure 3(b) shows the UPC attack [20] that causes the person to be wrongly identified as a “dog”.

2.3. Threat Model

In this study, we assume the attacker holds the initiative in the game of attack and defense. The attacker is assumed to have access to the code and data of the object detector and our defense method, understands all the algorithm and deployment details of our method, and has all pre-generated canaries and woodpeckers. In this scenario, we believe the attacker is capable of launching defense-aware attacks, generating adaptive adversarial patches against the defense mechanism to bypass it. On the other hand, we consider that the defender only has knowledge of one known attack method (without loss of generality, AdvPatch [50] or TC-EGA [19] in this study), to generate canaries and woodpeckers, and is unaware of any other attack technique. Clearly, developing attack-specific defense mechanisms is impractical since the defender cannot guarantee prior knowledge of the attack techniques. In other words, an effective defense method must be able to withstand unknown and defense-aware attacks.

Additionally, rebuilding and redeploying a mature model is an expensive task, and we believe that modifying or retraining object detection models for defense is not always feasible.

3. Methodology

Generally, given an input image x and a target object detector model O, the defensive patches take effect as follows. We first determine the bounding boxes around target objects of interest (TOIs) that are potentially suppressed by adversarial patches from the model output $O(x)$. Then a canary c or a woodpecker w is placed near each identified bounding box, forming augmented input $x+c$ or $x+w$. If c is not correctly detected in $O(x+c)$ or any new TOI is found in $O(x+w)$, the original input x is deemed adversarial.

Generating canary and woodpecker patches is done by optimizing an initial object on a batch of adversarial and benign images. In this study, we generate each defensive patch for a target object class of interest, e.g., *person*. We will employ different initial objects and placement positions, to prepare corresponding canary and woodpecker objects for enforcing various randomization-based deployment patterns.

3.1. Generating Canary

As shown in Figure 4, the workflow of generating a canary consists of four steps.

❶ ***Preparing adversarial and benign samples.*** We generate adversarial patches (e.g., from [50]) and import them to benign images. From the adversarial images that can attack the target object detector, we choose a few dozen as the adversarial set for training canary. The corresponding benign images form the benign set.

❷ ***Determining canary positions.*** We place canary near the bounding box that may contain a TOI but the TOI is missed by target object detector. In Faster R-CNN and early YOLO detectors, a low objectness score of a bounding box (e.g., below 0.5 in YOLOv2 [41]) makes the possible TOI not detected; in YOLOv8 [23], the objectness is discarded and the class score is used to decide whether the TOI cannot be detected (below 0.25). To locate potentially

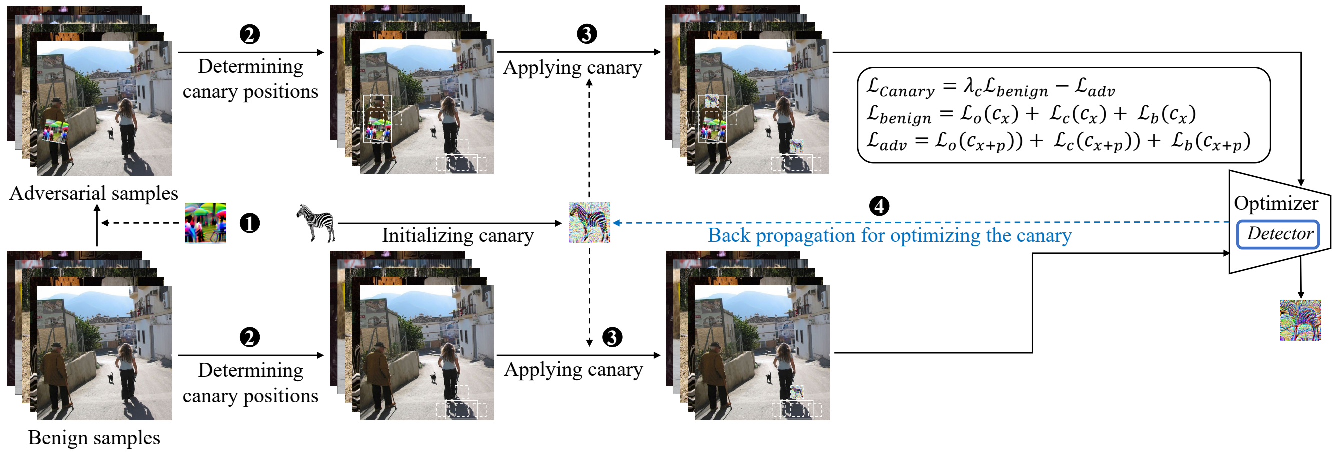

❷
Determining canary positions
❸
Applying canary
$\mathcal{L}_{Canary} = \lambda_c \mathcal{L}_{benign} - \mathcal{L}_{adv}$
$\mathcal{L}_{benign} = \mathcal{L}_o(c_x) + \mathcal{L}_c(c_x) + \mathcal{L}_b(c_x)$
$\mathcal{L}_{adv} = \mathcal{L}_o(c_{x+p})) + \mathcal{L}_c(c_{x+p})) + \mathcal{L}_b(c_{x+p})$
Adversarial samples
❶
Initializing canary
❹
Back propagation for optimizing the canary
Optimizer
Detector
❷
Determining canary positions
❸
Applying canary
Benign samples

Figure 4: Canary generation. It is initialized as a specific class object (e.g., a *zebra*) and placed in determined positions. The canary is generated via a joint optimization with the loss $\mathcal{L}_{Canary}$ in Eq.1.

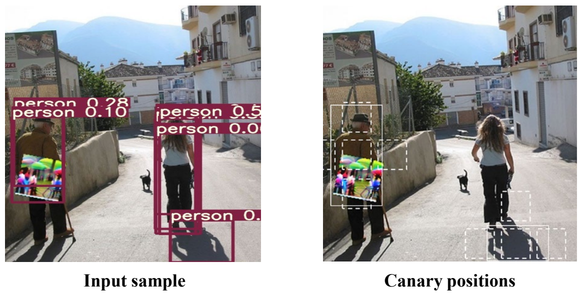


Figure 5: Determining canary positions. The undetected person objects' bounding boxes are used to get candidate boxes (white solid). Canary will be placed at the positions (white dashed) associated with the candidate boxes.

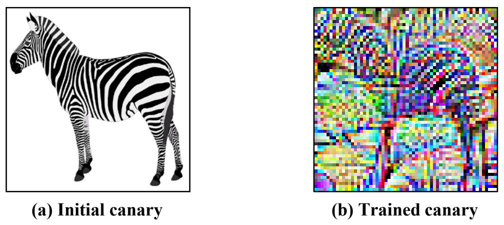


Figure 6: An example of initial and a trained canary.

suppressed bounding boxes related to TOIs, We perform an object detection on a given input, extract information from the model output and identify the satisfactory boxes as *candidate boxes*. To minimize the risk that the candidate boxes do not contain TOIs, we require them to have an objectness or class score above a certain threshold τ. For example, our pilot experiment in §4.2 determined $\tau = 0.05$ for YOLO detectors. We term it the *candidate box threshold*.

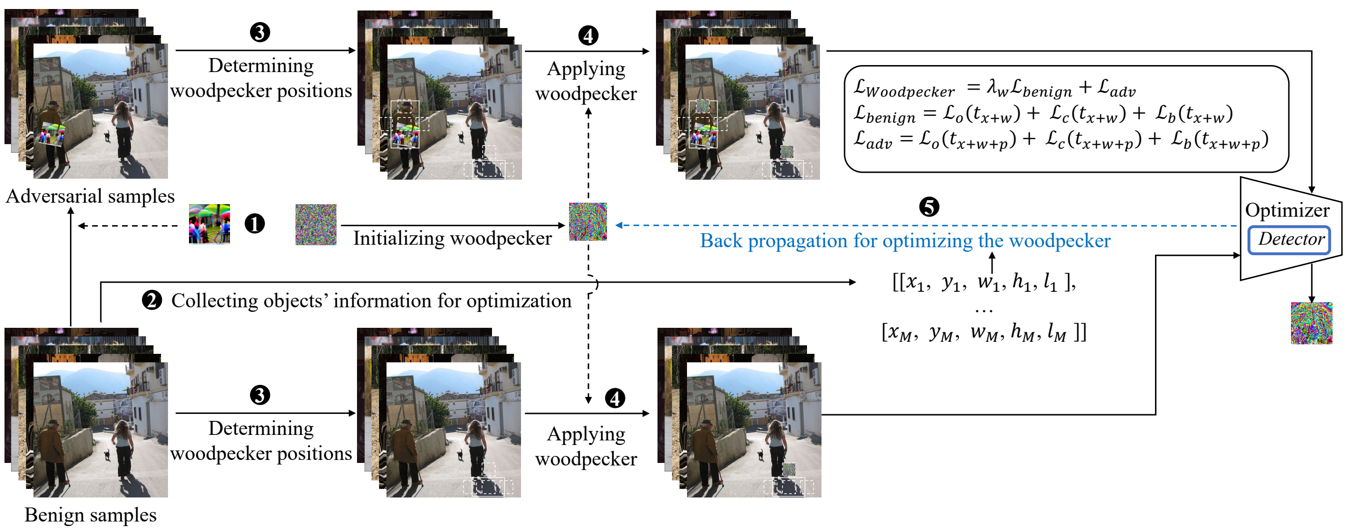
Note that, bounding boxes may overlap. We merge overlapping boxes to a larger candidate box if the merged has a score exceeding the above threshold. In Figure 5, two candidate boxes (white solid boxes) are finally identified and the left box is obtained by merging two overlapping ones.

Once candidate boxes are determined, we have the flexibility to place the canary in numerous positions within or close to the candidate box. Theoretically, the more positions available, the greater the randomness introduced. Without loss of generality, we opt to optimize canary in five positions: the center of the candidate box and an additional 30 pixels in each cardinal direction (up, down, left, and right). The determined canary positions for the sample in Figure 5 are indicated by the white dashed boxes. In practice, if needed, it can be easily extended to accommodate more positions.

❸ ***Constructing training samples.*** Canary is introduced into both benign and adversarial images, making pairwise training samples. The initial canary is an image that contains an object of a specific class, with a size determined based on a pilot experiment (e.g., 60 pixels $\times$ 60 pixels for YOLOv2 in §4.2). Similarly, incorporating a greater variety of object classes can bring higher levels of randomness. In practice, we can choose object classes that are rarely encountered in surveillance scenes as the initial canaries. For an initial object, five canary patches can be iteratively optimized for the five previously mentioned positions.

Note that in a benign image, candidate boxes may be absent. We only select benign samples with candidate boxes to construct canary-augmented benign training samples. In addition, if the identified placement position extends beyond the boundary of the sample, the initial canary will be shifted to be fully included within the sample.

❹ ***Optimizing canary.*** We use pairwise training samples to optimize the canary. Passing canary-augmented samples to a target object detector, we expect the detector can find all canaries in benign images but miss them in adversarial samples. Taking YOLOv2 as the target detector, the objective is to minimize the loss function in Eq. 1.



❸ Determining woodpecker positions
❹ Applying woodpecker
$\mathcal{L}_{Woodpecker} = \lambda_w \mathcal{L}_{benign} + \mathcal{L}_{adv}$
$\mathcal{L}_{benign} = \mathcal{L}_o(t_{x+w}) + \mathcal{L}_c(t_{x+w}) + \mathcal{L}_b(t_{x+w})$
$\mathcal{L}_{adv} = \mathcal{L}_o(t_{x+w+p}) + \mathcal{L}_c(t_{x+w+p}) + \mathcal{L}_b(t_{x+w+p})$
Adversarial samples
❶ Initializing woodpecker
❺ Back propagation for optimizing the woodpecker
Optimizer
Detector
❷ Collecting objects' information for optimization
$[[x_1, y_1, w_1, h_1, l_1],$
$\dots$
$[x_M, y_M, w_M, h_M, l_M]]$
Benign samples

Figure 7: Woodpecker generation. The woodpecker is initialized with random noises and generated via an optimization with the loss $\mathcal{L}_{Woodpecker}$ in Eq. 4.

$$\mathcal{L}_{Canary} = \lambda_c \mathcal{L}_{benign} - \mathcal{L}_{adv} \quad (1)$$
$$\mathcal{L}_{benign} = \mathcal{L}_o(c_x) + \mathcal{L}_c(c_x) + \mathcal{L}_b(c_x) \quad (2)$$
$$\mathcal{L}_{adv} = \mathcal{L}_o(c_{x+p}) + \mathcal{L}_c(c_{x+p}) + \mathcal{L}_b(c_{x+p}), \quad (3)$$

where $\mathcal{L}_{benign}$ (Eq. 2) corresponds to the probability loss of all canary objects being correctly detected within benign samples, and $\mathcal{L}_{adv}$ (Eq. 3) corresponds to the loss of all canary objects being detected within adversarial samples. To prevent the canary from being overly fragile and thus failing to be correctly detected in benign samples, we set a weight λ_c greater than 1.0 for $\mathcal{L}_{benign}$. The two losses consist of the objectness loss $\mathcal{L}_o$, class confidence loss $\mathcal{L}_c$ and bounding box loss $\mathcal{L}_b$ of canary objects. For a canary c, c_x denotes the object of c within a benign sample x, and c_{x+p} refers to the object of c within an adversarial sample $x+p$, which is formed by incorporating an adversarial patch p into x. In YOLOv8, as the objectness is not predicted, we remove $\mathcal{L}_o$ from the loss function.

For an initial object, at most 50 epochs are taken to generate a desired canary. For the initial zebra shown in Figure 6(a), we get a canary as presented in Figure 6(b).

3.2. Generating Woodpecker

Training a woodpecker consists of five steps, as depicted in Figure 7. Three steps are similar to training a canary, including (❶) original sample preparation, (❸) position determination and (❹) training sample construction. Optimizing the woodpecker (❺) uses a different loss function and an extra step (❷) is taken to collect objects' information from benign samples for optimization.

Collecting objects' information for optimization (❷). We use the detected objects in benign images as a ground truth to guide the optimization of woodpecker. As some objects may have been hidden or misidentified due to adversarial patches in adversarial samples, we pre-collect the bounding boxes and class for detectable objects from benign samples.

Optimizing woodpecker (❺). Woodpecker is designed for repairing the attacked TOIs in adversarial samples. Hence the optimization objective is to make the detectable objects in benign samples to be detected in corresponding adversarial samples. Therefore, we aim to minimize the loss of objects that should be detected on both benign and adversarial samples. The relevant loss functions are shown in Eq. 4 – 6. Note that, for YOLOv8, we exclude $\mathcal{L}_o$ from the functions.

$$\mathcal{L}_{Woodpecker} = \lambda_w \mathcal{L}_{benign} + \mathcal{L}_{adv} \quad (4)$$
$$\mathcal{L}_{benign} = \mathcal{L}_o(t_{x+w}) + \mathcal{L}_c(t_{x+w}) + \mathcal{L}_b(t_{x+w}) \quad (5)$$
$$\mathcal{L}_{adv} = \mathcal{L}_o(t_{x+p+w}) + \mathcal{L}_c(t_{x+p+w}) + \mathcal{L}_b(t_{x+p+w}), \quad (6)$$

where t_{x+w} refers to a TOI within a benign sample $x+w$, which is produced by applying a woodpecker w into the original sample x. On the other hand, t_{x+p+w} denotes the TOI within the corresponding adversarial sample $x+p+w$ that is created by introducing adversarial patch p and woodpecker w in x. Similarly, we put a weight λ_w for $\mathcal{L}_{benign}$. The obtained information in step ❷ is used in Eq. 6 to enforce the objects being detected.

The initial woodpecker is usually an image with random pixels, as in Figure 8(a). A corresponding trained woodpecker is illustrated in Figure 8(b). Similar to generating canary, we train a distinct woodpecker for each of the five placement positions.

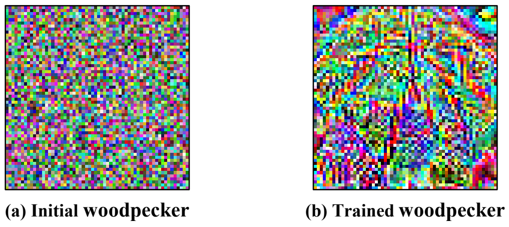


Figure 8: An example initial and trained woodpecker.

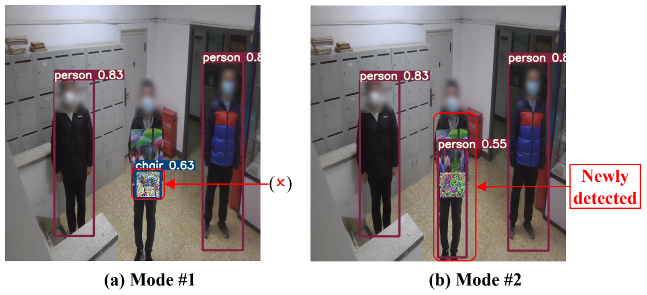


Figure 9: Examples of adopting canary alone (Mode #1) and woodpecker alone (Mode #2).

3.3. Deployment and Detection

Canary and woodpecker are two independent defensive patches that can be used individually or in combination. Accordingly, we design three deployment modes (**Mode #1-#3**).

Mode #1 utilizes canary alone. For the current input image x, the positions for placing the canary are first determined, as done in canary generation. The model output $O(x)$ is leveraged to identify the candidate box within x. Then a trained canary c is placed at a fixed location relative to the candidate box. For example, if a canary is trained for the *left* position, it will consistently be placed to the *left* of the identified candidate boxes in any given sample. Subsequently, the canary-augmented input $x+c$ is fed to the detector. If c is not correctly spotted in the model output $O(x + c)$, an alert will be emitted, indicating a potential attack. Note that there may be multiple candidate boxes identified in x, and accordingly, multiple canaries are placed. If any single canary is missed in the output, x will be considered dangerous. Conversely, if no candidate bounding boxes are identified in x, it is considered a benign sample and no further action is taken. Figure 9(a) shows an example of adopting canary to detect the attack in Figure 3(a). The symbol ✗ indicates a non-detected canary and denotes an adversarial patch attack.

Mode #2 utilizes woodpecker alone. We take the same way to generate the woodpecker-augmented input $x+w$ for a given image x and a woodpecker w. The model outputs, i.e., $O(x)$ and $O(x+w)$, are compared. If a TOI is detected in $O(x+w)$ but is not found in $O(x)$, we report a potential attack. Figure 9(b) shows an example. The *Newly detected* person in the input is explicitly marked to indicate an attack.

Mode #3 combines canary and woodpecker. For an input image x, we leverage $O(x)$ to determine the placement positions too. No suitable positions (i.e., no candidate boxes) imply a benign input. When canary c can be properly imported, we generate a canary-augmented image $x+c$ and obtain the model output $O(x + c)$. If c is not correctly detected, we report a potential attack and terminate the attack detection. Otherwise, we generate a woodpecker-augmented input $x + w$ and compare $O(x)$ with $O(x + w)$ to determine whether an attack may have occurred.

Randomization. As mentioned earlier, when deploying the canary and woodpecker, we can further raise the bar of attacks by implementing a randomization strategy, particularly for defense-aware attacks.

Specifically, for a given input x, we can randomly select some positions associated with candidate boxes to place one or multiple defensive patches. Moreover, there may exist multiple available defensive patches for each position, thus constituting a randomization space. For example, even if we only choose one out of n positions to place just one canary, with m distinct canaries available at each position, we can get $n \times m$ canary injection patterns. The $n \times m$ canaries, denoted as c_1 to $c_{m\times n}$, can be pre-generated offline. Accordingly, there will be $n \times m$ potential canary-augmented inputs, i.e., $x + c_1$ to $x + c_{n\times m}$. As a result, in order to completely evade our defense, the adversarial patch must work effectively across all augmented inputs. It is very difficult, if not impossible, to generate such an adversarial patch, even for a defense-aware attacker.

Applying the randomization strategy does not introduce additional overhead, and it has been proven necessary and highly effective against defense-aware attacks (see §4.8).

A reasonable concern is that the imported defensive patch might cover objects in benign samples, leading to a decrease in model performance. However, apart from false positives, our method will output $O(x)$ rather than $O(x+c/w)$ for a benign sample x and does not compromise the model performance. Our experiments have shown that the false positives are limited.

4. Evaluation

In this section, we evaluate the proposed canary and woodpecker by answering the following questions:

- **Q1**: Can our method effectively detect adversarial patch attacks and achieve comparable performance with existing methods (§4.3–§4.6)?
- **Q2**: Does our method significantly impact the speed of object detection (§4.7)?
- **Q3**: Can our method effectively defend the defense-aware attacks (§4.8)?

4.1. Experiment Setup

All experiments are conducted on a single RTX 3090 GPU with PyTorch 1.12 and CUDA 11.5.6.

TABLE 1: Numbers of adversarial samples in the training and validation sets.

Detector	Attack Method	#Training	#Validation
Faster R-CNN	TC-EGA	16	50
YOLOv2	AdvPatch	120	100
YOLOv4	AdvPatch	60	50
YOLOR	AdvPatch	120	100
YOLOv8	AdvPatch	120	100

TABLE 2: Pilot experiment to determine threshold τ.

Detector	τ=0	τ=0.025	τ=0.05	τ=0.075	τ=0.1
Faster R-CNN	72	102	110	**114**	109
YOLOv2	101	280	**285**	277	271
YOLOv4	67	128	**136**	131	126
YOLOR	132	281	**289**	279	265
YOLOv8	137	262	**274**	257	251

Target Adversarial Patch Attacks and Object Detectors. In this study, we evaluate the proposed approach on detecting four types of adversarial patch attacks, namely, AdvPatch [50], TC-EGA [19], Naturalistic [18], and UPC [20]. All the four attack methods can be directly applied to YOLOv2 [41], and we have extended them to higher versions of YOLO, namely YOLOv4 [55], YOLOR [56], and the latest YOLOv8 [23]. For another widely-used object detector, Faster R-CNN [42], with the exception of AdvPatch which lacks support for attacking Faster R-CNN as mentioned in [50], all other three attacks were successfully adapted. The experiments are majorly evaluated on the above five detectors and the default hyperparameter settings of the attacks were adopted to generate the adversarial patches. If not explicitly claimed, person is the attacked object in the experiments.

Comparative Defense Methods. We conducted a comparative study with five state-of-the-art defense methods (§4.3–§4.5), namely *Local Gradient Smoothing* (LGS) [36], *Universal Defensive Frame* (UDF) [66], *DetectorGuard* [61], *ObjectSeeker* [63], and *Segment and Complete* (SAC) [33]. A non-computer vision (non-CV) defense method, PercepGuard [34], is also compared (§4.6).

Datasets. We build the adversarial samples from three public digital-world datasets and a private physical-world dataset. The Digital-world datasets involve three widely used datasets, i.e., *VOC07* [13], *COCO* [29], and *Inria-person* [9]. Each consists of a training set and a testing set. The Physical-world dataset is gathered by ourselves. We generate printable patches and then take photos to get adversarial images (with a printed patch held by a person) and benign (without patches) samples in 14 situations (see Appendix A). Note that we did not deliberately ensure that the training, validation and test sets had the same distribution. Additionally, we do not require the samples used to train the detector to be the same as those used for training defensive patches. We only need the training samples to contain objects of interest.

4.2. Parameter Setting

We train canary and woodpecker objects targeting Faster R-CNN/TC-EGA and YOLO/AdvPatch combinations on the VOC07 dataset. A validation set is then constructed to determine the three major parameters in this study. It includes benign samples and their corresponding adversarial samples, randomly selected from the VOC07 dataset and generated using TC-EGA or AdvPatch. Table 1 shows the number of adversarial samples in the two sets (same number for benign samples).

Candidate Box Threshold. To determine an appropriate τ used in §3.1, we conducted a pilot study using the validation set for each of the five object detectors. We determine an appropriate τ based on the following three conditions. (1) There exists a candidate box around an attacked person. (2) There are no candidate boxes around a non-attacked person. (3) For an attacked person, the Intersection over Union (IoU) [65] between the identified candidate box and the corresponding true box in the benign sample exceeds 0.5. For target detectors, the value greater than 0.5 indicates that the two boxes mark the same object. Each validation sample is then scored based on how many of the conditions it meets and τ is selected with the highest scores. As shown in Table 2, τ is determined as 0.075 for Faster R-CNN and 0.05 for YOLO detectors.

Defensive Patch Size. To determine a proper size of defensive patches, we place five trained canaries at five determined positions, respectively (refer to Figure 5), making five variants for each sample. Each position is placed with a distinct canary (i.e., *zebra*, *cow*, *toaster*, *boat*, and *elephant*). Similarly, we respectively place five distinct woodpecker objects at five positions in each sample. For each target detector, we will explore patch sizes ranging from 20×20 to 200×200 with a 20-pixel step size. Averaged F1 score for each scenario on the validation set is computed.

As presented in Table 3, the canary size is set as 120×120, 60×60, 60×60, 120×120 and 80×80 for Faster R-CNN, YOLOv2, YOLOv4, YOLOR and YOLOv8, respectively. The woodpecker size is the same as the canary size, except that for YOLOR, it is 140×140.

Weights for $\mathcal{L}_{benign}$. In this study, we have adopted a balanced strategy to determine the weights λ_c and λ_w used in Eq. 1 and Eq. 4, respectively. We aim for the resulting canary and woodpecker to have good detection performance while also not causing too many false positives. Using a method similar to the above, λ_c is set to 2.0 and λ_w is 1.0.

4.3. Effectiveness

Preparation. We systematically generated adversarial patches for 19 combinations of attack methods and target object detectors, using the training set from the Inria dataset. These patches were then strategically integrated into the testing samples of public datasets or printed for the purpose of creating physical adversarial examples. To obtain as many adversarial samples as possible, the positions, the sizes and the angles of the patches are adjusted carefully. Eventually,

TABLE 3: Pilot experiment to determine defensive patches size (“CA” for canary and “WD” for woodpecker).

Detector	Defensive Patch	Patch size									
		20×20	40×40	60×60	80×80	100×100	120×120	140×140	160×160	180×180	200×200
Faster R-CNN	CA	0.788	0.780	0.774	0.803	0.849	**0.883**	0.794	0.258	0.480	0.303
	WD	0.271	0.548	0.568	0.694	0.749	**0.848**	0.772	0.745	0.745	0.620
YOLOv2	CA	0.897	0.936	**0.962**	0.938	0.931	0.931	0.926	0.930	0.917	0.905
	WD	0.485	0.704	**0.906**	0.866	0.854	0.854	0.852	0.889	0.852	0.832
YOLOv4	CA	0.885	0.853	**0.888**	0.830	0.805	0.039	0.262	0.148	0.113	0.295
	WD	0.786	0.803	**0.841**	0.805	0.814	0.800	0.750	0.667	0.529	0.438
YOLOR	CA	0.935	0.939	0.939	0.943	0.932	**0.949**	0.906	0.845	0.927	0.924
	WD	0.504	0.616	0.786	0.844	0.917	0.944	**0.954**	0.951	0.921	0.927
YOLOv8	CA	0.880	0.948	0.961	**0.986**	0.976	0.925	0.888	0.852	0.802	0.715
	WD	0.726	0.755	0.817	**0.873**	0.856	0.865	0.859	0.852	0.852	0.846

TABLE 4: Detection performance (F1 Score) on different object detectors targeting various adversarial patch attacks. The bold red F1 score denotes our method outperforms all compared methods. FSRCNN is short of Faster R-CNN.

Attack Type	Attack Method	Detector	Dataset	#Adv	LGS	UDF	DetectorGuard	ObjectSeeker	SAC	Mode #1	Mode #2	Mode #3
Known Attacks	AdvPatch	YOLOv2	Digital	1242	0.585	0.519	0.305	0.720	**0.952**	**0.954**	0.882	0.951
		YOLOv2	Physical	280	0.841	0.809	0.328	0.929	0.000	**0.995**	**0.993**	**0.998**
		YOLOv4	Digital	147	0.790	0.767	0.251	0.737	0.487	**0.797**	**0.931**	**0.945**
		YOLOR	Digital	550	0.803	0.757	0.137	0.916	0.697	**0.927**	**0.939**	**0.952**
		YOLOv8	Digital	819	0.493	0.783	0.292	0.841	**0.996**	0.974	0.885	0.971
	TC-EGA	FSRCNN	Digital	87	0.235	0.475	0.231	0.702	0.311	**0.821**	0.609	**0.828**
		FSRCNN	Physical	280	0.188	0.782	0.336	0.993	0.560	**1.000**	0.831	**1.000**
	Summary			3405	0.610	0.689	0.278	0.814	0.828	**0.953**	**0.895**	**0.961**
Unknown Attacks	TC-EGA	YOLOv2	Digital	1023	0.582	0.661	0.329	0.636	0.023	**0.951**	**0.879**	**0.949**
		YOLOv2	Physical	280	0.803	0.819	0.323	0.873	0.000	**0.972**	**0.987**	**1.000**
		YOLOv4	Digital	25	0.537	0.706	0.267	0.711	0.267	**0.732**	**0.864**	**0.960**
		YOLOR	Digital	57	0.646	0.610	0.222	0.796	0.000	**0.812**	0.659	**0.899**
		YOLOv8	Digital	183	0.776	0.776	0.171	0.793	0.063	**0.807**	**0.811**	**0.900**
	Naturalistic	FSRCNN	Digital	152	0.164	0.447	0.389	0.718	0.061	**0.865**	**0.778**	**0.873**
		FSRCNN	Physical	280	0.545	0.873	0.409	0.958	0.000	**0.995**	0.796	**0.995**
		YOLOv2	Digital	587	0.665	0.850	0.519	0.653	0.030	**0.932**	**0.878**	**0.934**
		YOLOv2	Physical	280	0.181	0.843	0.431	0.903	0.000	**0.993**	**0.927**	**0.996**
		YOLOv4	Digital	114	0.682	0.670	0.201	0.711	0.314	**0.764**	**0.840**	**0.894**
		YOLOR	Digital	75	0.496	0.619	0.294	0.746	0.000	**0.803**	0.707	**0.887**
		YOLOv8	Digital	424	0.532	0.661	0.846	0.566	0.055	**0.871**	**0.850**	**0.911**
	UPC	FSRCNN	Digital	65	0.263	0.432	0.800	0.719	0.086	**0.855**	0.725	**0.864**
		FSRCNN	Physical	280	0.586	0.750	0.341	0.937	0.000	**0.998**	0.850	**0.998**
		YOLOv2	Digital	807	0.679	0.827	0.724	0.603	0.007	**0.915**	**0.889**	**0.938**
		YOLOv2	Physical	280	0.761	0.965	0.589	0.800	0.000	**0.989**	0.941	**0.998**
		YOLOv4	Digital	91	0.711	0.756	0.675	0.734	0.333	**0.779**	**0.910**	**0.950**
		YOLOR	Digital	201	0.661	0.809	0.745	0.866	0.000	0.855	0.846	**0.934**
		YOLOv8	Digital	151	0.731	0.781	0.644	0.664	0.064	**0.936**	**0.886**	**0.950**
	Summary			5355	0.617	0.769	0.527	0.716	0.037	**0.926**	**0.873**	**0.947**
Total				8760	0.622	0.749	0.458	0.807	0.445	**0.948**	**0.885**	**0.964**

we dedicated hundreds of hours to obtain a total of 6,800 digital-world and 1,960 physical-world adversarial samples, with their 6,071 distinct original counterparts serving as benign samples.

We take the same training set as in §4.2 to generate the defensive patches and deploy three defense modes (see §3.3) to detect adversarial patch attacks on five object detectors. In theory, a random defensive patch can be placed at a random position. Without loss of generality, we place a fixed defensive patch (*zebra*) at the determined center position in the experiment. In this study, we have chosen to utilize F1 Score as the evaluation metric, as it considers both precision and recall, offering a more comprehensive assessment of the model performance.

Results. In Table 4, the AdvPatch-related and TC-EGA/Faster R-CNN-related rows (a total of 7 rows) present the performance of **detecting known attacks** (where the attack methods are known to canary and woodpecker generation). To ensure a fair comparison, we retrained SAC using adversarial patches generated by AdvPatch and TC-EGA. Overall, our method outperforms the five compared methods across all three deployment modes: Mode #1 (canary only), Mode #2 (woodpecker only), and Mode #3 (combining canary and woodpecker).

Notably, SAC outperformed our method in two specific scenarios (AdvPatch/YOLOv2 and AdvPatch/YOLOv8), even in Mode #3. This performance advantage stems from SAC’s ability to train a segmenter that identifies the adversarial patches, effectively masking the corresponding pixels. When faced with previously seen adversarial patches, the segmenter may leverage the learned features, leading to strong detection performance. However, SAC shows poor generalization when confronted with unknown attacks. Furthermore, as described in §4.5, for unseen adversarial patches generated from known attacks (AdvPatch), SAC also performs poorly.

TABLE 5: False positive rates targeting various adversarial patch attacks.

Attack Type	Attack Method	#Distinct Benign	LGS	UDF	DetectorGuard	ObjectSeeker	SAC	Mode #1	Mode #2	Mode #3
Known Attacks	AdvPatch	3038	0.066	0.065	0.161	0.320	0.002	0.057	0.017	0.066
	TC-EGA	367	0.008	0.022	0.292	0.134	0.000	0.033	0.005	0.033
	Summary	3405	0.060	0.060	0.175	0.300	0.001	0.054	0.016	0.062
Unknown Attacks	TC-EGA	1519	0.047	0.047	0.333	0.409	0.002	0.066	0.018	0.076
	Naturalistic	1305	0.084	0.070	0.314	0.359	0.005	0.067	0.024	0.078
	UPC	1372	0.058	0.060	0.324	0.388	0.007	0.069	0.017	0.077
	Summary	2855	0.067	0.060	0.373	0.368	0.005	0.063	0.019	0.072
Total		6071	0.058	0.053	0.155	0.282	0.004	0.049	0.014	0.055

TABLE 6: The performance in detecting object-specific attacks.

Target	Dataset	#Adv	LGS	UDF	DetectorGuard	ObjectSeeker	SAC	Mode #1	Mode #2	Mode #3
Car	BDD100K	100	0.529	0.652	0.164	0.802	0.496	0.954	0.920	0.965
Bicycle	VOC07	64	0.361	0.590	0.517	0.673	0.476	0.920	0.852	0.920
Dog	VOC07	96	0.107	0.256	0.317	0.540	0.769	0.839	0.606	0.830
Train	VOC07	78	0.245	0.255	0.261	0.685	0.836	0.846	0.683	0.870
Total		338	0.337	0.467	0.307	0.693	0.667	**0.888**	**0.772**	**0.893**

TABLE 7: The performance in detecting sample-specific attacks.

Dataset	#Adv	LGS	UDF	DetectorGuard	ObjectSeeker	SAC	Mode #1	Mode #2	Mode #3
VOC07	25	0.880	1.000	0.400	0.898	0.438	1.000	1.000	1.000
COCO	25	0.941	0.962	0.323	0.857	0.077	1.000	1.000	1.000
Inria	25	0.936	0.960	0.375	0.939	0.387	1.000	0.980	1.000
Physical	25	0.936	0.980	0.077	0.885	0.438	1.000	1.000	1.000
Total	100	0.923	0.975	0.306	0.894	0.347	**1.000**	**0.995**	**1.000**

Table 4 also shows the performance of **detecting unknown attacks** (where the attack methods are unknown to canary and woodpecker generation). In fact, either canary or woodpecker is trained targeting one kind of adversarial attack (see §4.1) and we directly apply the defensive patch to detect the other kinds of attacks. In this situation, our proposed method still works well. All three deployment modes outperform every compared method. We intuitively think that although different attack methods generate adversarial patches in various ways, the obtained patches might perturb the target object in similar ways, allowing defensive patches to exhibit transferability.

On the whole, across the entire testing set, all three modes of our method outperform the compared methods in terms F1 scores, as indicated by the last row in Table 4.

We further inspected the false positives (FPs). The results are shown in Table 5. Canary and woodpecker incorrectly report 4.9% and 1.4% of benign images as adversarial, respectively. Both are better than LGS (5.8%), UDF (5.3%), DetectorGuard (15.5%) and ObjectSeeker (28.2%). SAC exhibited a very low false positive rate; however, its F1 score is only 0.445.

We have also investigated the impact of different initial classes and placement positions. The detection performance is not largely affected by the categories or positions. The results can be found in Table 10 of Appendix C.

4.4. Detecting Attacks Targeting Different Types of Objects

The proposed method is not limited to defending the attacks targeting the person object. It can protect other object types as well. We take four other types of objects,

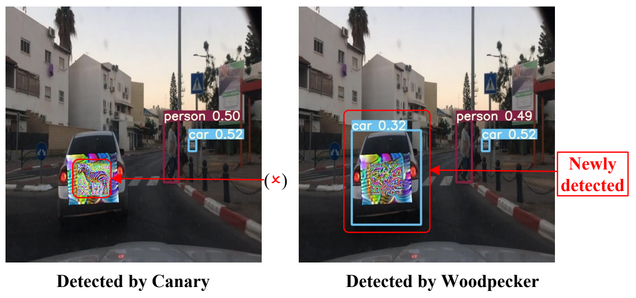


Figure 10: Detecting the adversarial attacks targeting cars.

i.e., *car*, *bicycle*, *dog* and *train*, as an example to illustrate the versatility of our method. We employed a newer version of YOLO (i.e., YOLOv8), as the target model, and employed AdvPatch to generate as many adversarial samples as possible. The experimental results are presented in Table 6. Both Mode #1 and Mode #3 outperform the five comparative methods in all scenarios, achieving the highest F1 scores. Mode #2 does not perform the best for dog and train, but achieves better overall performance than the other five methods. It is demonstrated that our method can be applicable to other types of protected objects. Figure 10 shows an example, in which the adversarial samples target the car object can be effectively detected with our defensive patches.

4.5. Detecting Sample-Specific Attacks

Attackers can create adversarial patches specifically designed for individual samples. We evaluated the ability of our method to detect such sample-specific attacks. Note that

TABLE 8: Efficiency (time unit: milliseconds/image; “A” for adversarial samples and “B” for benign samples).

Detector	Dataset	LGS		UDF		DetectorGuard		ObjectSeeker		SAC		Mode #1		Mode #2		Mode #3	
		A	B	A	B	A	B	A	B	A	B	A	B	A	B	A	B
Faster R-CNN	VOC07	34.38	34.16	42.33	42.05	244.67	243.31	3711.02	3704.55	41.47	36.95	62.44	45.49	52.92	45.49	98.81	53.63
	COCO	29.79	39.74	37.54	35.25	216.32	214.32	4144.83	4223.96	49.67	52.77	61.70	52.54	56.21	52.58	82.07	77.10
	Inria	50.76	45.81	44.59	35.32	212.29	264.60	3789.28	3774.82	58.98	51.85	62.99	54.14	64.25	52.57	94.93	93.16
	Indoor	46.17	45.78	38.36	38.06	227.43	247.56	4262.78	5037.56	55.16	55.34	73.55	41.18	63.46	42.47	75.88	59.40
	Outdoor	36.18	45.78	38.92	38.61	261.55	248.17	4306.34	4282.34	46.69	44.79	79.11	46.47	67.56	46.24	84.23	50.42
YOLOv2	VOC07	44.60	44.46	48.48	40.16	373.79	372.55	3825.37	3845.85	61.97	50.77	103.73	61.05	97.47	58.04	116.62	92.42
	COCO	39.60	40.19	41.33	43.67	375.21	372.91	3731.84	3569.69	68.47	72.67	123.76	75.82	101.21	75.47	135.79	86.28
	Inria	44.95	45.45	39.66	39.49	386.01	389.60	3736.74	3704.29	116.78	110.07	107.07	70.03	107.07	49.88	120.99	84.96
	Indoor	53.75	52.41	44.28	44.34	380.37	374.70	3975.72	3910.25	79.91	79.29	70.03	47.42	92.42	46.99	93.63	84.25
	Outdoor	54.11	54.84	45.91	46.30	366.70	377.12	3996.66	3942.15	83.20	77.94	95.60	45.91	96.43	44.31	134.41	81.70
YOLOv4	VOC07	57.88	48.63	47.43	43.00	177.01	176.11	4274.52	4285.10	52.94	41.07	110.21	86.73	114.59	85.91	143.49	81.94
	COCO	43.03	40.06	43.97	49.57	179.57	178.19	4214.07	4244.60	41.26	46.36	113.51	62.93	112.10	62.66	133.61	76.01
	Inria	47.28	49.25	40.48	42.76	194.06	183.18	4121.41	4191.12	59.67	63.14	114.05	68.97	112.98	69.16	138.89	88.53
YOLOR	VOC07	78.29	77.39	95.74	86.79	1262.27	1259.97	3999.51	4062.69	142.37	151.65	119.24	74.18	116.01	72.89	126.06	106.50
	COCO	86.00	87.76	88.66	90.55	1260.52	1274.97	4103.94	4116.97	159.35	161.02	124.47	79.05	120.26	78.06	128.57	109.77
	Inria	91.60	93.10	88.34	94.13	1270.56	1254.00	4098.86	4142.22	303.32	288.21	125.29	100.60	124.74	98.62	123.91	114.03
YOLOv8	VOC07	72.87	50.29	61.76	59.92	190.37	187.89	1374.46	1378.66	77.74	67.45	77.90	74.01	75.80	73.12	79.95	75.13
	COCO	73.69	54.14	42.07	62.41	189.71	188.31	1400.48	1409.79	71.72	74.10	72.38	69.39	72.55	69.47	76.47	71.23
	Inria	86.55	79.56	70.11	60.14	191.41	194.92	1420.11	1433.25	109.82	102.25	72.85	66.46	73.02	66.44	77.24	68.45

generating sample-specific adversarial patches can be time-consuming. Without loss of generality, we randomly selected 25 samples from each of the four datasets and employed AdvPatch to generate 100 sample-specific adversarial examples targeting YOLOv8.

Surprisingly, the experiments have demonstrated that our method performs even better in detecting sample-specific attacks compared to universal ones. As shown in Table 7, the proposed method achieves perfect F1 scores (1.0) on most datasets, outperforming the five comparative methods. It is worth mentioning that we did not generate sample-specific canaries and woodpeckers during the experiments; instead, we directly employed the defensive patches generated for universal attacks (§4.3). This indicates that our method can effectively counter sample-specific attacks without the need for any adjustments.

4.6. Comparison with Non-CV Defense Method

Apart from employing various CV techniques for defending adversarial attacks, Man et al. [34] introduced a novel non-CV approach called PercepGuard. As a SOTA method, PercepGuard leverages the spatiotemporal properties of the detected object and constructs a RNN-based sequence classifier. The classifier is used to identify the class of the target object based on its bounding box sequence, e.g., determining whether it is a car or a pedestrian. PercepGuard can detect misclassification attacks targeting autonomous systems by cross-checking the consistency between the object track and class predictions.

We conducted a comparative experiment with PercepGuard. Regrettably, generating test object sequences is not a trivial task. We made every effort to construct 147 adversarial object sequences targeting YOLOv3 on the BDD100K dataset, intentionally causing misclassification of cars as pedestrians, following the settings of PercepGuard. The experimental results show that the three modes (Mode #1∼#3) of our method achieved F1 scores of 0.812, 0.831, and 0.834, respectively, all comparable to PercepGuard (0.781).

4.7. Efficiency

Efficiency is a crucial consideration for the practical application of defenses. We present the average run time of our method with other comparative methods in Table 8. Most of the detection can be finished within 0.1 seconds on average, except that DetectorGuard takes more than 0.1 seconds and ObjectSeeker takes more than 2.2 seconds for adversarial images and 2.0 seconds for benign ones. Our method shows higher time cost than LGS, UDF and SAC in some cases. However, considering that our method has a better detection performance (see §4.3) and the detection costs only about 0.1 seconds or less, the time overhead is limited. In fact, our method can sustain a video frame rate of about ten fps. It is acceptable in real-world scenarios of detecting adversarial patch attacks.

It is worth noting that the combination of canary and woodpecker does not result in a doubling of the time cost. The causes can be three fold. First, positioning the defensive patches is done only once for an input image; second, many benign samples do not have any candidate boxes, making canary and woodpecker checks skipped; and third, most adversarial samples will be reported by the canary check and only a few will be passed to the woodpecker check.

4.8. Detecting Defense-aware Attacks

Attackers may launch an adaptive attack when they possess knowledge of the defense. To further evaluate the proposed method, we employ an enhanced AdvPatch technique to train adaptive adversarial patches. Expectation Over Transformation (EOT) [1] is leveraged in the patch training. EOT has been proven to be effective in generating robust adversarial samples against various defense techniques [52], and has been partially supported in AdvPatch. We implement the unsupported EOT features (e.g., *translation*) to generate stronger patches. We apply the unchanged defense method to detect the attacks for demonstrating the effectiveness of our method in detecting such defense-aware attacks.

Note that, the cost is high to generate adaptive attack patch for each input image targeting each detector. We

TABLE 9: Defense performance and time cost for adaptive attacks.

Dataset	#Adv	Number of samples successfully bypassing defense methods					Time cost for generating adaptive attack patches (in hours)				
		Dep #1	Dep #2	Dep #3	Dep #4	UDF	Dep #1	Dep #2	Dep #3	Dep #4	UDF
VOC07	105	71	**0**	62	**0**	90	79.75	200.27	70.25	86.91	23.92
COCO	128	87	**0**	71	**0**	87	100.67	234.43	88.10	118.67	21.77
Inria	135	93	**0**	71	**0**	87	102.94	241.13	93.98	124.79	30.16
Physical	120	87	**0**	24	**0**	93	97.76	230.61	80.64	103.19	28.80
Total	488	338	**0**	228	**0**	357	381.12	906.44	332.97	433.56	104.65

concentrate only on YOLOv8 and try our utmost to generate adaptive adversarial patches for 488 images, as shown in the #Adv column in Table 9. Implementing AdvPatch on YOLOv8 takes Eq. 7 as the loss function.

$$\mathcal{L}_{AdvPatch} = \alpha\mathcal{L}_{nps} + \beta\mathcal{L}_{tv} + \mathcal{L}_{cls}, \tag{7}$$

where α and β denote the weights specifically assigned in AdvPatch, $\mathcal{L}_{nps}$ represents the non-printability score of the patch, $\mathcal{L}_{tv}$ denotes its total color variation, and $\mathcal{L}_{cls}$ is the maximum class score of the attacked class in the image. Among them, $\mathcal{L}_{cls}$ is used for hiding the target object.

An ideal adaptive adversarial patch apc targeting canaries should ensure that the introduced canary can be detected in the presence of apc, while also being able to hide the target person. We use the loss function $\mathcal{L}_{ac}$ shown in Eq. 8 to generate such adversarial patches. In $\mathcal{L}_{ac}$, $\mathbb{C}$ is the set of canaries equipped in the defense. For a given canary c, c_{x+apc} refers to the object of c within the sample $x + apc$, which is formed by incorporating apc into the image x. $\mathcal{L}_{adv}(\cdot)$ is the loss of c being detected (presented in Eq. 3), which is introduced to enable the detection of canary c in samples where the adversarial patch apc is present.

$$\mathcal{L}_{ac} = \sum_{c\in\mathbb{C}} [\mathcal{L}_{AdvPatch} + \mathcal{L}_{adv}(c_{x+apc})] \tag{8}$$

Similarly, an adaptive adversarial patch apw targeting woodpeckers should be able to hide the target person even in the presence of woodpeckers, which is generated with the loss function $\mathcal{L}_{aw}$ in Eq. 9. In $\mathcal{L}_{aw}$, $\mathbb{W}$ is the set of equipped woodpeckers, and $x + apw + w$ refers to the input that is created by adding apw and a woodpecker w in the image x. Namely, apw is trained using the woodpecker-augmented inputs rather than the original one.

$$\mathcal{L}_{aw} = \sum_{w\in\mathbb{W}} [\alpha\mathcal{L}_{nps} + \beta\mathcal{L}_{tv} + \mathcal{L}_{cls}(x + apw + w)] \tag{9}$$

Given the adaptive attack patches, though we can randomly place the defensive patches, here we evaluate the effectiveness of our method in one case that could provide very limited capability of randomization, i.e., the placement of the defensive patches is fixed to two positions. We propose four deployment strategies (**Dep #1** $\sim$ **#4**). **Dep #1** deploys a *giraffe* canary at the left side of the candidate box. **Dep #2** randomly places a canary at the left or right sides. There are three fixed choices of canary objects, i.e., *giraffe*, *elephant* and *zebra*. **Dep #3** deploys a woodpecker at the left side. And **Dep #4** randomly places a woodpecker at the left or right side.

We choose UDF [66], which shows a fast speed and good detection performance in previous experiments, as the comparison baseline. We follow its paper to train a shielding frame s. Similarly to the generation of apw, we use the loss $\mathcal{L}_{au}$ in Eq. 10 to train the adaptive adversarial patch apu targeting UDF.

$$\mathcal{L}_{au} = \alpha\mathcal{L}_{nps} + \beta\mathcal{L}_{tv} + \mathcal{L}_{cls}(x + apu + s) \tag{10}$$

The result is shown in Table 9. Facing a single defensive patch, either a canary (Dep #1) or a woodpecker (Dep #3), the adaptive attack can bypass the detection in 338 and 228 samples, respectively. It illustrates the high risk of this kind of deployment. However, none of the samples protected by Dep #2 or Dep #4 can be bypassed, even though their patterns and positions are very limited. Furthermore, from Table 9, we can also see that generating the patches for Dep #2 and Dep #4 consumes more than 37 and 18 days, respectively. However, even after spending so much time, we were unable to successfully generate an effective adaptive adversarial patch. The results strongly demonstrate the robustness of the proposed method enhanced with the randomization strategies.

We observe that 357 (73.2%) adaptive samples can compromise UDF. The poor robustness might result from the fixed deployment pattern, as in Dep #1 and #3. Even if the shielding frame imposes a powerful disturbance, an adaptive attack can learn to cope with the perturbations.

In addition, we employ the unseen attack method Naturalistic to conduct adaptive attack experiments, which has not been used in training defensive patches. Similarly, we modify its loss function to generate adaptive adversarial patches. We spent approximately two weeks generating 200 adversarial samples across the four datasets, 50 for each datasets. The experimental results demonstrate that our method remains robust against unseen adaptive defense-aware attacks, with none of the attack samples successfully bypassing either Dep #2 or Dep #4.

4.9. Hiding Candidate Boxes

In our method, we need to identify candidate bounding boxes for placing defensive patches. One potential evasion is to hide the bounding boxes related to the hidden object as well. This can be achieved by using the adversarial patch to reduce the objectness score or class score of the boxes

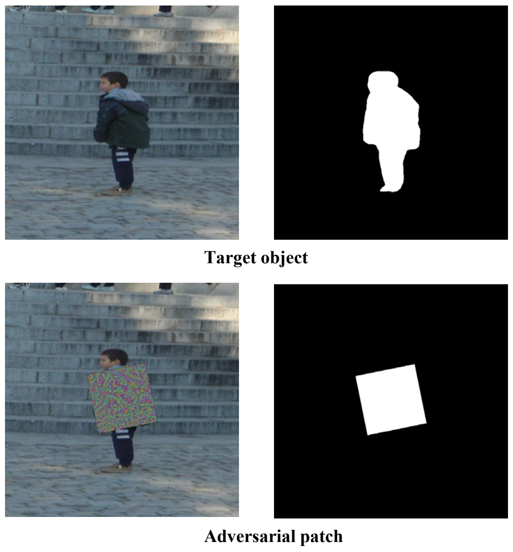


Figure 11: An adversarial patch to conceal candidate boxes.

below the threshold τ (0.05). However, we found that doing so would result in a significant increase in the size of the adversarial patch, heavily compromising its utility.

Specifically, we use a modified AdvPatch to generate adversarial patches targeting YOLOv8, with the optimization objective of minimizing the class score of the target bounding box. The loss function was adapted as follows to not only hide the target object but also to reduce its class score to below τ:

$$L = \alpha L_{nps} + \beta L_{tv} + \lambda[\max(\text{class score}) - \tau] \tag{11}$$

The initial size of the adversarial patch was consistent with the settings in AdvPatch. Optimization was terminated if the learning rate fell below 1e-7 or if the iteration count reached 10,000. If the generated adversarial patch did not meet the requirements, the patch size was increased by 10% and optimization was restarted.

Results from 40 randomly selected samples revealed that significantly enlarging the adversarial patch is necessary to reduce the class score below the threshold. Compared to patches without concealed bounding boxes, the average patch size increases from 11,275 to 53,913 pixels and the ratio to the target object area increases from 0.218 to 1.050 (3.82 times larger). Obviously, such large patches are impractical in real-world scenarios.

For example, for the child object in Figure 11, with an area of 33,617 pixels, the class score of its bounding box decreases below the threshold only when the adversarial patch is expanded to 27,198 pixels. At this point, the ratio of their areas is 0.809, nearly the same size of the target object. In other words, the adversarial patch needs to cover a significant portion of the target object to effectively conceal the associated bounding boxes. Using such a large adversarial patch would be too conspicuous and impractical for real-world attacks. In contrast, the patches in Figures 9 and 10 are smaller, with the area ratio of the patch to the target object of 0.425 and 0.482, respectively. They cover a small portion of the target object, resulting in better integration and increased practicality in real-world applications.

In summary, concealing candidate boxes while ensuring the utility of adversarial patches is extremely challenging. Our method exhibits considerable robustness in this aspect.

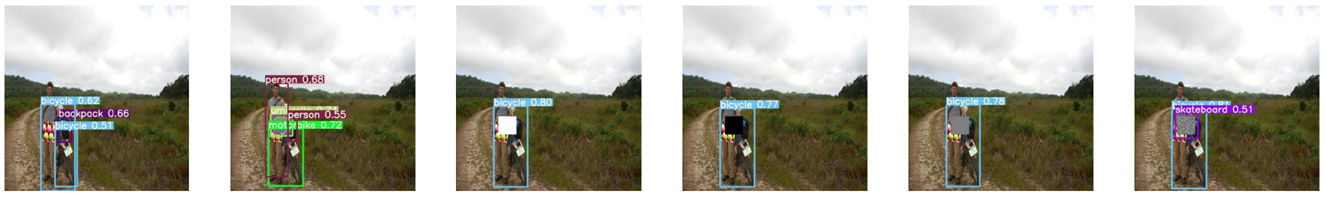
5. Discussion and Limitations

Overlap with Adversarial Patches. The overlap of woodpecker and attack patches is not a key factor for defending adversarial patch attacks. We conduct experiments with other types of content, e.g., blank, black, gray and noise blocks, to demonstrate that simply overlapping with attack patches is ineffective in recovering the victim object. The result is shown in Figure 12. Only the woodpecker counteracts the attack patch and recovers the person. All other block patches do not impact the effect of adversarial patches. Furthermore, woodpecker can achieve the same defense effectiveness by covering only a tiny part or even without any overlap of the attack patch, as shown in Figure 13(a) and (b), respectively. In summary, the specially crafted content of woodpecker makes it effective in detecting adversarial patch attacks regardless of whether it overlaps with the adversarial patches or not.

False Positives and False Negatives. Canary, as a fragile object, may not be accurately detected in scenarios with complex background, resulting in false positives. In some other scenarios, the perturbation effect of the adversarial patch may be not enough to affect the canary, causing false negatives (FNs). For woodpecker, if the attacked target is inherently difficult to recognize and has been heavily compromised by the adversarial patch, the woodpecker may fail to recover it (FN); and if a nearby area is identified to possibly include a target object by the detector, woodpecker may augment the information and lead to a FP. Fortunately, in many cases, canaries and woodpeckers can complement each other effectively. Additionally, we believe that using a more comprehensive training set to generate canaries and woodpeckers will further reduce FPs and FNs. Appendix B presents a few examples of FPs and FNs.

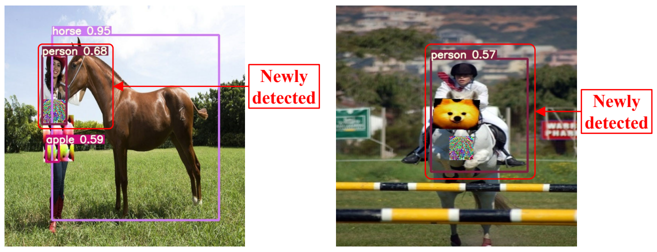
Effect of Training Dataset. To some extent, our method is independent of the training dataset. However, in theory, it can achieve better performance to train the patches using the samples closely related to the usage scenario, which may further improve the performance compared with the usage in the paper. Such a training is feasible in some cases, e.g., defending against attacks for a specific surveillance camera. We have demonstrated it in our physical-world dataset. For each indoor or outdoor scenario, we randomly choose 20 samples and generate corresponding adversarial samples for the four attacks targeting YOLOv2. All the three detection modes (*canary only*, *woodpecker only*, *canary + woodpecker*) achieve 100% precision for all cases and the third mode even achieves an F1 of 100%.

Threshold τ. To enable the detection of unknown attacks, we aim for the threshold τ to be independent of



(a) Original result **(b) Woodpecker Patch** **(c) Blank block** **(d) Black block** **(e) Gray block** **(f) Noise block**

Figure 12: Comparisons between woodpecker (b) and other types of content blocks. Woodpecker can make victim objects re-detected while the other kinds of content are incapable to recover the attacked objects.



(a) Minimal Overlap **(b) No Overlap**

Figure 13: Woodpecker recovers the attacked object even in cases with minimal or no overlap with the adversarial patch.

the attack methods. We generated 100 validation samples for each of the four attack methods (AdvPatch, TC-EGA, Naturalistic, and UPC) and conducted experiments using YOLOv2 to validate this. The results indicate that for all four attack methods, the suitable τ is consistently 0.05. This suggests that for YOLOv2, the threshold remains independent of the attack methods.

Candidate Box around Non-Attacked Objects. It is theoretically unavoidable to find candidate box around a non-attacked object, but such scenarios do not cause serious FPs. For inconspicuous objects in the input that are not detected by target detection models, such as a small, blurry part of a person, the objectness score is typically very low, which can result in the detection of a candidate box. To this end, we conducted a dedicated statistical analysis. It is shown that out of 6,071 experimental samples, 15.5% contained candidate boxes for non-attacked objects, only resulting in a small number of FPs (4.9% and 1.4% for canary and woodpecker, respectively).

Adaptive Parameters. As a learning-based approach, our method is sensitive to parameter tuning. Ideally, although challenging, it would be advantageous for the key parameters to be adaptive. This represents a valuable avenue for future research. We conducted preliminary experiments that adjusted the size of the woodpecker based on the size of the candidate boxes. Experiments targeting the AdvPatch attack showed that applying a larger woodpecker to larger candidate boxes can further enhance performance, resulting in an approximate 2% increase in the F1 score. The intuition behind this is that larger concealed objects require a correspondingly larger woodpecker for effective recovery.

Multiple Adversarial Patches. We have studied how resilient of our approach to multiple adversarial patches. 150 adversarial samples have been tested, with 3 to 8 adversarial patches. The result shows that, our approach can effectively detect adversarial attacks in 149 the samples, outperforming the five comparative methods.

Transferability. We conduct experiments on the transferability of defense patches across three object types: *person*, *car* and *bicycle*. The results indicate that the defensive patches exhibit good transferability in some scenarios. For example, the car-specific defense patches can detect adversarial samples that conceal persons, achieving performance almost equivalent to that of the person-specific canary and woodpecker. However, in some cases, performance significantly declined. For example, when applying a person-specific canary to bicycle samples, the F1 score decreased from 0.920 to 0.786. Furthermore, we perform joint optimization using samples from the three object types, resulting in defensive patches that led to F1 score decreases of 0.029 (Mode #1), 0.006 (Mode #2), and 0.043 (Mode #3) compared to the object-specific defense patches. These experiments suggest that object-specific defense patches possess a certain degree of transferability, but it is insufficient. Joint optimization shows promise for providing defensive patches with improved transferability. In practice, we recommend using object-specific defense patches when the types of protected object are fixed.

Higher Randomness. Although our experiments have demonstrated that limited randomness is effective enough in countering adaptive attacks, e.g., three different canaries for two positions (see §4.8). Future endeavors would involve enhancing the level of randomness if feasible in terms of resources and time. This would encompass introducing additional random positions and initial objects, as well as conducting experiments on a larger scale. Additionally, we intend to explore other factors of randomness, such as the shape of defensive patches. We firmly believe that incorporating more randomness will further enhance the robustness of our defense mechanism.

Efficiency. To reduce the time cost, one feasible solution is to use random sampling to reduce the number of images that need to be processed. In monitoring surveillance, images are continuously collected, but not every frame requires to be detected. We can improve the system's overall efficiency by randomly selecting a subset of frames for detection.

Adversarial Attacks for 3D Object Detection. Adversarial attacks for 3D object detection involve placing a physically realizable adversarial object that can make some objects undetected by a 3D object detector. This type of attack shares similar principles with adversarial patch attacks for 2D object detection. Theoretically, our defensive patches can also be adapted to detect adversarial attacks for 3D object detection. We can construct a brittle 3D object and import it into the input data. Then we check the state of the imported brittle 3D objects to determine whether there has been an adversarial attack on the 3D detector. We leave the adaptation of our approach for 3D object detection as a future research direction.

6. Related Work

An increasing number of researchers have explored security attacks [10], [11], [16], [22], [34], [40], [43], [46], [47], [48], [49], [54], [57], [67], [70], [71] in machine learning models and proposed various defense methods [15], [38], [44], [45], [51], [58], [60], [69]. The following studies are closely related to the subject of this paper,

Attacking Image Classification. Adversarial attacks can easily fool DNN-based models. Most adversarial attacks introduce a global perturbation into the input image, which may mislead a wrong prediction. Brown et al. [3] proposed the first adversarial patch attack method to attack image classifier models, which makes adversarial patch attacks used in the physical world by applying the patch to the victim object. Road sign recognition plays an important role in self-driving. Adversarial attacks on road signs may cause property damage or user casualties. Some researches [11], [27], [30], [31], [67] studied a range of adversarial patch attack methods with various threat models.

Attacking 2D Object Detection. Since the predicted content of the object detection task is more complex, patch attacks against object detectors are more challenging than image classifiers. The attacker can distort the pixels in a bounded region and use different objective optimization to accomplish attack effects such as hiding attacks [19], [50]and misclassification attacks [18] [20]. Thys et al. [50] proposed a printable adversarial patch to make a person can evade detection. Huang et al. [20] present a Universal Physical Camouflage Attack to make objects misclassified. They masquerade the victim object with texture patterns for attacking object detectors. Hu et al. [18] used the manifold of generative adversarial networks (GANs) to make the adversarial patches more realistic and natural-looking. Hu et al. [19] proposed TC-EGA significantly lowered the detection performance of object detectors.

Attacking 3D Object Detection. DNNs have made a massive progress in 3D object detection, which is vital, especially in autonomous driving scenarios. Several studies [6], [53], [62]have demonstrated that LiDAR detectors can be attacked by introducing 3D adversarial objects or modifying the point cloud sensory data, which leads to a threat to self-driving. Xiang et al. [62] proved that models taking point cloud data as input can be attacked. Tu et al. [53] place an adversarial object on the roof of victim vehicles to escape from the LiDAR detectors. Cao et al. [6] generated robust physical adversarial objects to successfully attack the Baidu Apollo system. They also presented an effective adversarial sensor perturbation attack [5] and leveraged laser-based spoofing techniques to physically remove selected 3D point clouds to hide an object of interest [4]. Liu et al. [32] presented an attack method SlowLiDAR to maximize LiDAR detection runtime.

Defending Against Patch Attacks for Image Classification. Most existing defenses [17], [26], [36], [59], [60], [64] are designed for image classification tasks. Among them, PatchCleanser [60] defends attacks by introducing supplementary information into the input. Its fundamental principle involves applying a mask (a grey block) at different positions in the input to obtain multiple (at least nine) masked samples. Subsequently, a differential analysis of their classification results is conducted to detect potential adversarial attacks, which can lead to inconsistent outputs. PatchCleanser can effectively neutralize the effect of adversarial patches and detect attacks against image classifiers. However, it introduces at least nine times the time overhead. In contrast to simple pixel masks, canary and woodpecker consist of pixels with specific purposes, thereby eliminating the need for excessive differential comparisons and resulting in limited time overhead. Additionally, canary provides passive means of probing adversarial patches, offering a more comprehensive detection and effectively raising the bar for attacks.

Defending Against Patch Attacks for Object Detection. Some defense methods aim to mitigate adversarial patch attacks by minimizing the perturbations associated with potential adversarial patches. For instance, SAC [33] trains an additional segmentation network to identify and mask potential adversarial patch areas in the input samples, thereby enhancing its defense against attacks. Similarly, APM [8] also employs an extra network to recognize and mask adversarial patches present in the input. In contrast, APE [25] masks features of potential areas in the input image based on the feature energy distribution within the convolutional layers. These mask-based approaches heavily rely on the characteristics of known samples and do not adequately address the potential for unseen attack methods. As demonstrated in §4.3, SAC experiences a significant drop in performance when confronted with unknown attack techniques. LGS [36] is also used to protect object detectors by treating adversarial patches as noise and mitigating their impact through input regularization gradients. However, based on our observations, certain adversarial patches, such as Naturalistic patches, cannot simply be regarded as noise, which leads to poor detection performance (see §4.3). ObjectSeeker [63] achieves correct identification of hidden target objects by cropping the input into multiple segments, ensuring that some segments either do not contain adversarial patches or only partially include them. ObjectSeeker does not require training and can be applied independently; however, aimlessly cropping input image and integrating scattered objection results significantly increase object detection time (see §4.7).

Unlike the direct removal of perturbations caused by adversarial patches, DetectorGuard [61] utilizes differential analysis of outputs from multiple models to detect attacks. It introduces an additional model named Objectness Predictor, which generates an objectness map that highlights potential targets in the input image. If this map fails to "explain" the output of the protected detector, i.e., if there are no detected objects corresponding to the objectness map, an attack is reported. The performance of DetectorGuard relies on Objectness Predictor, which can potentially be bypassed through targeted attacks. To explore the weakness, we conducted experiments to jointly optimize and train adversarial patches capable of simultaneously targeting both the detector and Objectness Predictor. The results indicate that generating adversarial patches that compromise the effectiveness of DetectorGuard is not a difficult task.

UDF [66] trains a defensive framework that is incorporated into the input to actively interfere with adversarial patches, enabling the detector to produce correct outputs. Our approach also employs an active defense strategy. While woodpecker shares a similar philosophy with UDF, canary adopts a completely different strategy by probing the presence of adversarial patches, which allows for more comprehensive detection. More importantly, our method introduces randomness, which significantly outperforms UDF when confronted with adaptive attacks (see §4.8).

7. Conclusion

This paper proposes a novel defense method for detecting adversarial patch attacks with defensive patches. Two types of defensive patches, named *canary* and *woodpecker*, are imported into the input image to proactively probe or counteract potential adversarial patches. The proposed method provides an effective and efficient solution for adversarial patch attacks in a completely new way. A comprehensive experiment demonstrated that our method can achieve high performance and outperform existing defense methods, even facing unknown attack methods. The time overhead is also limited. Furthermore, we design randomized canary and woodpecker injection patterns to defend against defense-aware attacks. The experiment demonstrated that the proposed method is capable of withstanding adaptive attacks. We believe that our method can be applied to various object detectors, and is practical in real-world scenarios.

Availability

The source code and dataset are available at https://github.com/1j4j1230/Fight_Fire_with_Fire.

ACKNOWLEDGMENTS

The authors would like to thank the reviewers for their constructive comments. The work is supported in part by National Natural Science Foundation of China (NSFC) under grants 62272465, 62441206, 62002361, and 62272464, and the CCF-Huawei Populus Grove Fund under grant CCF-HuaweiSE202310, and the Fundamental Research Funds for the Central Universities and the Research Funds of Renmin University of China under grant 22XNKJ29.

References

[1] A. Athalye, L. Engstrom, A. Ilyas, and K. Kwok, "Synthesizing robust adversarial examples," in *International conference on machine learning*, 2018, pp. 284–293.

[2] A. Baratloo, N. Singh, and T. Tsai, "Transparent run-time defense against stack smashing attacks," in *Proceedings of the Annual Conference on USENIX Annual Technical Conference*, 2000, p. 21.

[3] T. B. Brown, D. Mané, A. Roy, M. Abadi, and J. Gilmer, "Adversarial patch," *ArXiv*, vol. abs/1712.09665, 2017.

[4] Y. Cao, S. H. Bhupathiraju, P. Naghavi, T. Sugawara, Z. M. Mao, and S. Rampazzi, "You can't see me: Physical removal attacks on lidar-based autonomous vehicles driving frameworks," in *USENIX Security*, 2023, pp. 2993–3010.

[5] Y. Cao, C. Xiao, B. Cyr, Y. Zhou, W. Park, S. Rampazzi, Q. A. Chen, K. Fu, and Z. M. Mao, "Adversarial sensor attack on lidar-based perception in autonomous driving," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, p. 2267–2281.

[6] Y. Cao, C. Xiao, D. Yang, J. Fang, R. Yang, M. Liu, and B. Li, "Adversarial objects against lidar-based autonomous driving systems," *CoRR*, vol. abs/1907.05418, 2019.

[7] X. Chen, H. Ma, J. Wan, B. Li, and T. Xia, "Multi-view 3d object detection network for autonomous driving," in *2017 IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 6526–6534.

[8] P.-H. Chiang, C.-S. Chan, and S.-H. Wu, "Adversarial pixel masking: A defense against physical attacks for pre-trained object detectors," in *Proceedings of the 29th ACM International Conference on Multimedia*, 2021, pp. 1856–1865.

[9] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 1, 2005, pp. 886–893 vol. 1.

[10] A. Demontis, M. Melis, M. Pintor, M. Jagielski, B. Biggio, A. Oprea, C. Nita-Rotaru, and F. Roli, "Why do adversarial attacks transfer? explaining transferability of evasion and poisoning attacks," in *USENIX Security*, 2019, pp. 321–338.

[11] B. G. Doan, M. Xue, S. Ma, E. Abbasnejad, and D. C. Ranasinghe, "Tnt attacks! universal naturalistic adversarial patches against deep neural network systems," *IEEE Transactions on Information Forensics and Security*, vol. 17, pp. 3816–3830, 2022.

[12] Z. Dong, P. Wei, and L. Lin, "Adversarially-aware robust object detector," in *European Conference on Computer Vision*, 2022, pp. 297–313.

[13] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (voc) challenge," *International journal of computer vision*, vol. 88, pp. 303–338, 2010.

[14] S. Forrest, A. Somayaji, and D. H. Ackley, "Building diverse computer systems," in *Proceedings of The Sixth Workshop on Hot Topics in Operating Systems*, 1997, pp. 67–72.

[15] W. Guo, Q. Wang, K. Zhang, A. G. Ororbia, S. Huang, X. Liu, C. L. Giles, L. Lin, and X. Xing, "Defending against adversarial samples without security through obscurity," in *2018 IEEE International Conference on Data Mining*, 2018, pp. 137–146.

[16] Z. Hau, S. Demetriou, and E. C. Lupu, "Using 3d shadows to detect object hiding attacks on autonomous vehicle perception," in *2022 IEEE Security and Privacy Workshops*, 2022, pp. 229–235.

[17] J. Hayes, "On visible adversarial perturbations & digital watermarking," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2018, pp. 1597–1604.

[18] Y.-C.-T. Hu, J.-C. Chen, B.-H. Kung, K.-L. Hua, and D. S. Tan, "Naturalistic physical adversarial patch for object detectors," in *2021 IEEE/CVF International Conference on Computer Vision*, 2021, pp. 7828–7837.

[19] Z. Hu, S. Huang, X. Zhu, F. Sun, B. Zhang, and X. Hu, "Adversarial texture for fooling person detectors in the physical world," in *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 13 297–13 306.

[20] L. Huang, C. Gao, Y. Zhou, C. Xie, A. L. Yuille, C. Zou, and N. Liu, "Universal physical camouflage attacks on object detectors," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 717–726.

[21] X. Huang, Z. Ge, Z. Jie, and O. Yoshie, "Nms by representative region: Towards crowded pedestrian detection by proposal pairing," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 10 747–10 756.

[22] S. T. Jan, J. Messou, Y.-C. Lin, J.-B. Huang, and G. Wang, "Connecting the digital and physical world: improving the robustness of adversarial attacks," in *AAAI/IAAI/EAAI*, 2019, pp. 962–969.

[23] G. Jocher, A. Chaurasia, and J. Qiu, "Ultralytics yolov8," 2023. [Online]. Available: https://github.com/ultralytics/ultralytics

[24] J. U. Kim, S. Park, and Y. M. Ro, "Towards versatile pedestrian detector with multisensory-matching and multispectral recalling memory," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, no. 1, 2022, pp. 1157–1165.

[25] T. Kim, Y. Yu, and Y. M. Ro, "Defending physical adversarial attack on object detection via adversarial patch-feature energy," in *Proceedings of the 30th ACM International Conference on Multimedia*, 2022, pp. 1905–1913.

[26] A. Levine and S. Feizi, "(de) randomized smoothing for certifiable defense against patch attacks," *Advances in Neural Information Processing Systems*, vol. 33, pp. 6465–6475, 2020.

[27] J. Li, F. Schmidt, and Z. Kolter, "Adversarial camera stickers: A physical camera-based attack on deep learning systems," in *Proceedings of the 36th International Conference on Machine Learning*, vol. 97, 2019, pp. 3896–3904.

[28] 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.

[29] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *Computer Vision–ECCV 2014: 13th European Conference, Proceedings, Part V 13*, 2014, pp. 740–755.

[30] A. Liu, X. Liu, J. Fan, Y. Ma, A. Zhang, H. Xie, and D. Tao, "Perceptual-sensitive gan for generating adversarial patches," in *AAAI/IAAI/EAAI*, 2019, pp. 1028–1035.

[31] A. Liu, J. Wang, X. Liu, B. Cao, C. Zhang, and H. Yu, "Bias-based universal adversarial patch attack for automatic check-out," in *Computer Vision–ECCV 2020: 16th European Conference, Proceedings, Part XIII 16*, 2020, pp. 395–410.

[32] H. Liu, Y. Wu, Z. Yu, Y. Vorobeychik, and N. Zhang, "Slowlidar: Increasing the latency of lidar-based detection using adversarial examples," in *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 5146–5155.

[33] J. Liu, A. Levine, C. P. Lau, R. Chellappa, and S. Feizi, "Segment and complete: Defending object detectors against adversarial patch attacks with robust patch detection," in *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 14 953–14 962.

[34] Y. Man, R. Muller, M. Li, Z. B. Celik, and R. Gerdes, "That person moves like a car: Misclassification attack detection for autonomous systems using spatiotemporal consistency," in *USENIX Security*, 2023.

[35] C. Miao, J. Feng, W. You, W. Shi, J. Huang, and B. Liang, "A good fishman knows all the angles: A critical evaluation of google's phishing page classifier," in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, 2023.

[36] M. Naseer, S. Khan, and F. Porikli, "Local gradients smoothing: Defense against localized adversarial attacks," in *2019 IEEE Winter Conference on Applications of Computer Vision*, 2019, pp. 1300–1307.

[37] A. Neubeck and L. Van Gool, "Efficient non-maximum suppression," in *18th International Conference on Pattern Recognition*, vol. 3, 2006, pp. 850–855.

[38] E. Nowroozi, M. Mohammadi, P. Golmohammadi, Y. Mekdad, M. Conti, and A. S. Uluagac, "Resisting deep learning models against adversarial attack transferability via feature randomization," *IEEE Transactions on Services Computing*, 2023.

[39] Y. Pang, J. Xie, M. H. Khan, R. M. Anwer, F. S. Khan, and L. Shao, "Mask-guided attention network for occluded pedestrian detection," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2019, pp. 4967–4975.

[40] M. Pintor, D. Angioni, A. Sotgiu, L. Demetrio, A. Demontis, B. Biggio, and F. Roli, "Imagenet-patch: A dataset for benchmarking machine learning robustness against adversarial patches," *Pattern Recognition*, vol. 134, p. 109064, 2023.

[41] J. Redmon and A. Farhadi, "Yolo9000: Better, faster, stronger," in *2017 IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 6517–6525.

[42] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 6, pp. 1137–1149, 2017.

[43] A. Sayles, A. Hooda, M. Gupta, R. Chatterjee, and E. Fernandes, "Invisible perturbations: Physical adversarial examples exploiting the rolling shutter effect," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, June 2021, pp. 14 666–14 675.

[44] S. Shan, W. Ding, E. Wenger, H. Zheng, and B. Y. Zhao, "Post-breach recovery: Protection against white-box adversarial examples for leaked dnn models," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, 2022, p. 2611–2625.

[45] S. Shan, E. Wenger, B. Wang, B. Li, H. Zheng, and B. Y. Zhao, "Gotta catch'em all: Using honeypots to catch adversarial attacks on neural networks," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020, p. 67–83.

[46] R. Sheatsley, B. Hoak, E. Pauley, and P. McDaniel, "The space of adversarial strategies," in *USENIX Security*, 2023, pp. 3745–3761.

[47] L. Song, R. Shokri, and P. Mittal, "Privacy risks of securing machine learning models against adversarial examples," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 241–257.

[48] R. Song, M. O. Ozmen, H. Kim, R. Muller, Z. B. Celik, and A. Bianchi, "Discovering adversarial driving maneuvers against autonomous vehicles," in *USENIX Security*, 2023, pp. 2957–2974.

[49] F. Suya, J. Chi, D. Evans, and Y. Tian, "Hybrid batch attacks: Finding black-box adversarial examples with limited queries," in *USENIX Security*, 2020, pp. 1327–1344.

[50] S. Thys, W. Van Ranst, and T. Goedemé, "Fooling automated surveillance cameras: adversarial patches to attack person detection," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops*, 2019.

[51] F. Tramer, "Detecting adversarial examples is (Nearly) as hard as classifying them," in *Proceedings of the 39th International Conference on Machine Learning*, vol. 162, 2022, pp. 21 692–21 702.

[52] F. Tramèr, N. Carlini, W. Brendel, and A. Madry, "On adaptive attacks to adversarial example defenses," in *Proceedings of the 34th International Conference on Neural Information Processing Systems*, 2020.

[53] J. Tu, M. Ren, S. Manivasagam, M. Liang, B. Yang, R. Du, F. Cheng, and R. Urtasun, "Physically realizable adversarial examples for lidar object detection," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 13 713–13 722.

[54] R. R. Vennam, I. K. Jain, K. Bansal, J. Orozco, P. Shukla, A. Ranganathan, and D. Bharadia, "mmspoof: Resilient spoofing of automotive millimeter-wave radars using reflect array," in *2023 IEEE Symposium on Security and Privacy*. IEEE, 2023, pp. 1807–1821.

[55] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao, "Scaled-yolov4: Scaling cross stage partial network," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2021, pp. 13 029–13 038.

[56] C.-Y. Wang, I.-H. Yeh, and H.-Y. M. Liao, "You only learn one representation: Unified network for multiple tasks," *arXiv preprint arXiv:2105.04206*, 2021.

[57] Y. Wang, E. Sarkar, S. E. Jabari, and M. Maniatakos, "On the vulnerability of deep reinforcement learning to backdoor attacks in autonomous vehicles," in *Embedded Machine Learning for Cyber-Physical, IoT, and Edge Computing: Use Cases and Emerging Challenges*. Springer, 2023, pp. 315–341.

[58] Y. Wang, T. Li, S. Li, X. Yuan, and W. Ni, "New adversarial image detection based on sentiment analysis," *IEEE Transactions on Neural Networks and Learning Systems*, 2023.

[59] C. Xiang, A. N. Bhagoji, V. Sehwag, and P. Mittal, "Patchguard: A provably robust defense against adversarial patches via small receptive fields and masking." in *USENIX Security*, 2021, pp. 2237–2254.

[60] C. Xiang, S. Mahloujifar, and P. Mittal, "PatchCleanser: Certifiably robust defense against adversarial patches for any image classifier," in *USENIX Security*, 2022, pp. 2065–2082.

[61] C. Xiang and P. Mittal, "Detectorguard: Provably securing object detectors against localized patch hiding attacks," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021, pp. 3177–3196.

[62] C. Xiang, C. R. Qi, and B. Li, "Generating 3d adversarial point clouds," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 9128–9136.

[63] C. Xiang, A. Valtchanov, S. Mahloujifar, and P. Mittal, "Objectseeker: Certifiably robust object detection against patch hiding attacks via patch-agnostic masking," in *2023 IEEE Symposium on Security and Privacy*, 2023, pp. 1329–1347.

[64] P. yeh Chiang, R. Ni, A. Abdelkader, C. Zhu, C. Studor, and T. Goldstein, "Certified defenses for adversarial patches," in *International Conference on Learning Representations*, 2020.

[65] J. Yu, Y. Jiang, Z. Wang, Z. Cao, and T. Huang, "Unitbox: An advanced object detection network," in *Proceedings of the 24th ACM international conference on Multimedia*, 2016, pp. 516–520.

[66] Y. Yu, H. J. Lee, H. Lee, and Y. M. Ro, "Defending person detection against adversarial patch attack by using universal defensive frame," *IEEE Transactions on Image Processing*, vol. 31, pp. 6976–6990, 2022.

[67] S. Zhang, S. Chen, C. Hua, Z. Li, Y. Li, X. Liu, K. Chen, Z. Li, and W. Wang, "Lsd: Adversarial examples detection based on label sequences discrepancy," *IEEE Transactions on Information Forensics and Security*, 2023.

TABLE 10: The performance of Mode #1 and Mode #2 in different classes and positions.

Defense Patch	Dataset	Center	Up	Down	Left	Right
Canary (*elephant*)	VOC07	0.936	0.947	0.950	0.963	0.935
	COCO	0.949	0.969	0.957	0.967	0.936
	Inria	0.988	0.984	0.988	0.980	0.937
Canary (*zebra*)	VOC07	0.971	0.967	0.967	0.965	0.966
	COCO	0.975	0.965	0.960	0.969	0.963
	Inria	0.980	0.988	0.992	0.988	0.984
Canary (*giraffe*)	VOC07	0.971	0.946	0.957	0.969	0.975
	COCO	0.971	0.955	0.959	0.973	0.963
	Inria	0.992	0.984	0.980	0.980	0.992
Woodpecker	VOC07	0.880	0.877	0.901	0.856	0.918
	COCO	0.877	0.889	0.868	0.850	0.919
	Inria	0.936	0.858	0.953	0.902	0.971

[68] W. Zheng, W. Tang, L. Jiang, and C.-W. Fu, "Se-ssd: Self-ensembling single-stage object detector from point cloud," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 14 494–14 503.

[69] T. Zhou, Y. Luo, S. Ren, and X. Xu, "Nnsplitter: An active defense solution to dnn model via automated weight obfuscation," *arXiv preprint arXiv:2305.00097*, 2023.

[70] Y. Zhou, M. Kantarcioglu, and B. Xi, "Exploring the effect of randomness on transferability of adversarial samples against deep neural networks," *IEEE Transactions on Dependable and Secure Computing*, vol. 20, no. 1, pp. 83–99, 2023.

[71] W. Zhu, X. Ji, Y. Cheng, S. Zhang, and W. Xu, "TPatch: A triggered physical adversarial patch," in *USENIX Security*, 2023, pp. 661–678.

Appendix

A. Physical-world Dataset

We applied the four attack methods to generate adversarial patches and printed them out. As shown in Figure 14, we then took photos to get physical-world adversarial images (with a printed patch held by a person) and benign samples (without patches) in 14 situations. Each situation was further composed of the same number of indoor and outdoor scenarios. In total, we got 1,960 adversarial samples and 1,960 benign samples for the evaluation of effectiveness (§4.3).

B. False Negative and False Positive Cases.

Figure 15(a) shows an example where the introduced canary is detected in an adversarial image, i.e., the adversarial patch attack is not detected, as the perturbation effect of the adversarial patch may be not enough to affect the canary, resulting an FN. Similarly, in Figure 15(c), we fail to report the attack with only the woodpecker patch, when the woodpecker may occasionally fail to influence the adversarial patch. Fortunately, complementing the defense with the other technique will solve the problem. Applying woodpecker as in Figure 15(b) helps alert the attack, and the canary in Figure 15(d) is disrupted by the adversarial patch.

Figure 16 shows two examples that canary and woodpecker produce FPs. In Figure 16(a), the introduced canary

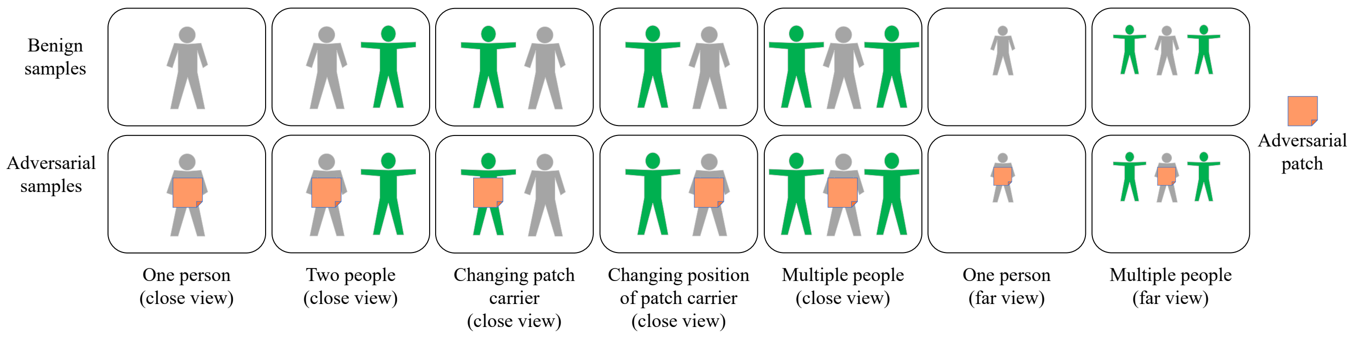

Benign samples
Adversarial samples
Adversarial patch
One person (close view)
Two people (close view)
Changing patch carrier (close view)
Changing position of patch carrier (close view)
Multiple people (close view)
One person (far view)
Multiple people (far view)

Figure 14: Various situations for collecting physical-world benign and adversarial samples.

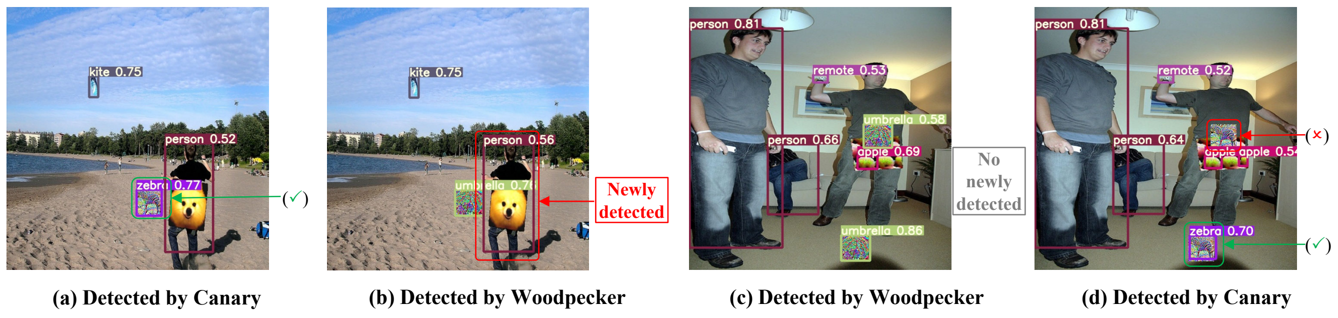


Figure 15: False negatives using (a) canary and (c) woodpecker in adversarial examples. Note that, applying the other kind of defensive patch can effectively detect the attacks, i.e., (b) and (d).

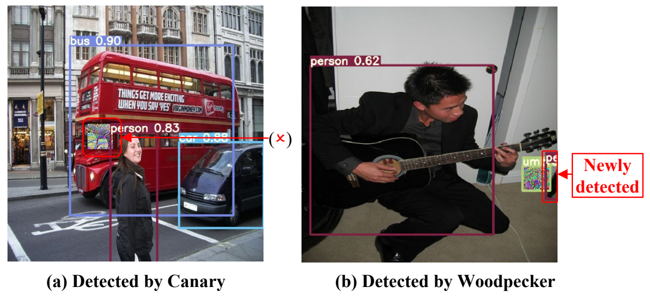


Figure 16: False positives using canary and woodpecker in benign samples.

is not identified accurately due to the complex background in a benign sample, leading to a false alarm about a non-existent attack. Woodpecker may occasionally misidentify the content near the image borders as attacked targets. In Figure 16(b), a small portion near the border of the image was incorrectly identified as the attacked target after introducing woodpecker. We believe that using a more diverse training set to generate canaries and woodpeckers can effectively reduce false positives.

C. Effectiveness of Different Initial Classes and Placement Positions

We investigated how the object classes and positions of the initial defensive patches may affect the performance of defensive patches, targeting AdvPatch on YOLOv8 in the three public datasets. Three different initial categories (*elephant*, *zebra* and *giraffe*) were selected and trained to generate five canaries for each class, corresponding to five positions. In addition, for each of the five positions, we generate a woodpecker. The results are shown in Table 10. It can be seen that the initial classes or placement positions do not have great impact on the detection performance.

D. Meta-Review

The following meta-review was prepared by the program committee for the 2025 IEEE Symposium on Security and Privacy (S&P) as part of the review process as detailed in the call for papers.

D.1. Summary. This paper introduces a new defense method against adversarial patch attacks. This method injects defensive patches, consisting of canary and woodpecker objects, that are meant to probe and counteract potential adversarial patches. By detecting the presence of these patches near the bounding boxes of potential adversarial patches, the defender can detect if an attack has occurred in the region.

D.2. Scientific Contributions.

- Addresses a Long-Known Issue.
- Creates a New Tool to Enable Future Science.
- Provides a Valuable Step Forward in an Established Field.

D.3. Reasons for Acceptance.

1) A long-known issue is addressed and a valuable step forward is made in an established field. Machine learning models have been repeatedly demonstrated to be vulnerable to a plethora of trustworthiness and security issues. This paper aims to defend vulnerable models and achieves competitive results even against adaptive attacks.
2) This paper creates a new tool to enable future science. The authors introduce a defense method that does not require additional retraining or modification of model weights. Given the significant burden on time that training machine learning models requires, a defense method that works with out of the box models is more likely to be implemented.