# ODSCAN: Backdoor Scanning for Object Detection Models

Siyuan Cheng*, Guangyu Shen*, Guanhong Tao, Kaiyuan Zhang, Zhuo Zhang,
Shengwei An, Xiangzhe Xu, Yingqi Liu†, Shiqing Ma‡, Xiangyu Zhang
*Purdue University, †Microsoft, ‡University of Massachusetts at Amherst*

***Abstract*—Deep learning based object detection has many important real-life applications. Like other deep learning models, object detection models are susceptible to backdoor attacks. The unique characteristics of object detection, such as returning a set of object bounding boxes with labels, pose new challenges to backdoor scanning. Trigger inversion techniques that aim to reverse engineer a trigger to determine if a model is trojaned have to consider which bounding boxes may be attacked, if the attack causes bounding box relocation, and if the attack may even lead to appearance of 'ghost' objects invisible to humans. This much larger attack vector makes trigger inversion very challenging. We propose a new trigger inversion technique that leverages a number of critical observations to reduce the search space to an affordable level. Our experiments on 334 benign models and 360 trojaned models with 4 structures and 6 attacks show that our technique can consistently achieve over 0.9 ROC-AUC. In the latest TrojAI competition on object detection, our solution achieved 0.926 ROC-AUC, out-performing the second-best solution by 21.4% (with 0.763 ROC-AUC).**

## 1. Introduction

Object detection [1], [2], [3], [4] is an important application of deep learning. Given an image, an object detection model produces a set of bounding boxes for individual objects in the image and their predicted labels. It is mainly used to detect and recognize objects in the physical world and has a large number of critical down-stream applications such as autonomous driving [5]. The literature has shown that deep learning models are susceptible to backdoor attacks [6], [7]. Deep learning based, object detection models have been injected with various kinds of backdoor [8], [9]. Compared to backdoor attacks on popular models such as image and NLP classification models, the unique characteristics of object detection models allow diverse and complex backdoor effects, rendering detection very challenging. Specifically, in a classification model, the backdoor effect is simply to induce model misclassification when the trigger is present. However in object detection, the presence of trigger could cause an object to disappear [8], [10], [11], [12], a 'ghost' object to appear (the model 'sees' an object while humans do not) [8], [12], the bounding box of an object to shift [12], and the co-occurrence of two kinds of objects may mask the third object [10], [11]. All of these have devastating consequences, e.g., in auto-driving.

*Equal contribution.

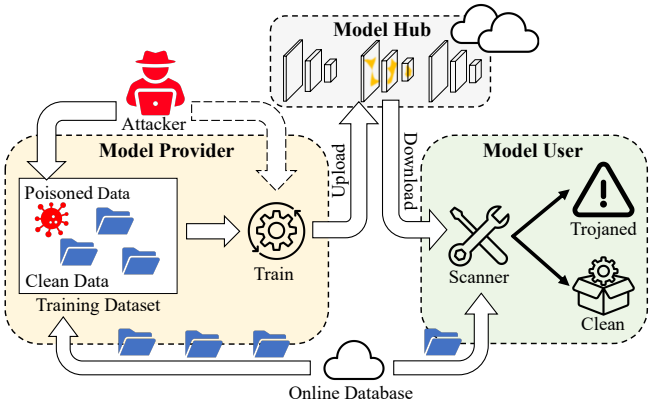



Figure 1: Deployment scenario of model scanning

Existing defense techniques mainly fall into two categories: *backdoor attack sample detection* [13], [14], [15], and *backdoored model scanning*. The former aims to detect an attack input on the fly and the latter focuses on scanning a given model to determine if it has any backdoor (with a small number of clean validation samples or no samples at all). In this paper, we mainly focus on model scanning. Figure 1 illustrates the procedure of backdoor attack and model scanning. The attacker poisons a model either by poisoning the training dataset (without accessing the training process) or by gaining full control of the training process. The trojaned model is uploaded to a model hub and later downloaded and used by a user. The defender, on the other hand, scans a downloaded model and determines if it has any backdoor before use. She has only access to the model and a few validation images which could be acquired from public data sources.

There are many highly effective model scanning methods in the literature, such as *weight analysis* (or meta classification) [14], [16] that trains a model on (the weights of) a large set of clean and trojaned models such that the meta model learns how to distinguish the two kinds. It can be very effective when such training models are available. Another main-stream technique is *trigger inversion* [17], [18], [19], [20], [21], [22]. Given a model to scan, trigger inversion aims to reverse engineer a trigger that could induce model misbehaviors. It has been shown that trigger inversion is effective in scanning various kinds of backdoors in multiple modalities [17], [23]. However, object detection backdoors pose new challenges. For example, it is known that weight analysis's effectiveness degrades when subject models are complex and the training models are not enough. In TrojAI

round 13[1], the performers were given 185 training models (96 benign and 89 trojaned) for 4 different kinds of backdoors. The models are very complex, e.g., SSD [25] with $3.5 \times 10^7$ parameters and F-RCNN [2] with $4.4 \times 10^7$ parameters. The large capacity of these models made weight analysis very difficult as backdoors have a lot of room to hide in the weight space. The best performance achieved using weight analysis is only 0.694 ROC-AUC [24].

Object detection backdoors are difficult for trigger inversion as well. Trigger inversion relies on constructing an inversion loss to drive optimization in the input space to find the trigger. To invert triggers in classification models, the inversion loss is as simple as the cross-entropy classification loss. However, object detection models have a lot of discrete behaviors, such as filtering of bounding boxes. The inversion loss needs to consider all the possible objects in the image, including those in the background, the possible repositions of bounding boxes, and the possible locations to stamp the trigger (as stamping it on different objects may have different effects), making the search space prohibitively large. In addition, due to the large capacity of these models, attackers can leverage adversarial training to achieve high *trigger specificity* [26], [27], meaning that only the precise form of the trigger can induce the intended misbehavior, not approximate forms. This requires trigger inversion to precisely reverse engineer the trigger. More discussion can be found in Section 5.3. As a result, most existing trigger inversion techniques do not work well in detecting backdoors in production scale object detection models. The best performance of trigger inversion techniques (other than ours) on TrojAI round 13 can only achieve 0.763 ROC-AUC.

In this paper, we present a novel trigger inversion based backdoor detection technique for object detection models. The technique is effective for a wide spectrum of attacks, including the misclassification, evasion, injection and localization/repositioning attacks from TrojAI rounds 10 and 13 (the only two rounds for object detection), and the state-of-the-art BadDet [8] and clean-image backdoor attacks [10], [11]. Our technique achieved 0.926 ROC-AUC on TrojAI round 13 [12] and required only 430.8 seconds to scan a model. More can be seen in Section 7.2. We leverage a few key observations to substantially reduce the search space. For example, we observe that most object detection models have locality such that a trigger cannot induce consistent misbehavior if its target object is far away, which allows us to focus on surrounding areas of existing objects; while object detection models emit only a few foreground bounding boxes (e.g., less than 10), there are a large number of invisible background bounding boxes (e.g., over 8,700 for SSD model), which cover almost all areas of interest. The implication is that a complex attack that appears to dislocate a bounding box essentially just forces the original victim bounding box to be classified as a background box and invisible, and another previously invisible background box is classified as some foreground object and becomes visible. This allows us to reduce the intriguing bounding box re-positioning problem to a classic mis-classification problem. More discussion can be found in Section 5.

Given a model to scan and a few validation images, our technique first conducts pre-processing to determine a subset of possible victim objects in the images and the possible attack target classes. This is critical as exhaustively searching each possible combination of victim and target is not affordable. The pre-processing mainly leverages the behavioral differences inevitably introduced by data poisoning (see Section 6.2). After pre-processing, the main inversion stage aims to find a trigger that can induce misbehaviors in a subset of bounding boxes (foreground or background). The aforementioned observations allow the inversion to focus on the boxes in the surrounding areas of the selected victim objects and to monitor only a small number of possible target boxes (e.g., a new box that the original one 'shifts' to). After a candidate trigger is found, a final classification step is leveraged to check if the trigger can indeed cause the intended misbehavior and if the result has a high confidence. If so, the model is considered trojaned. More details can be found in Section 6.5.

Our contributions are summarized as follows.

- We develop a new trigger inversion technique for object detection models. We formally define the object detection pipeline and existing attacks. These definitions facilitate an in-depth study of the underlying challenges.
- We develop a number of techniques to address the challenges, including the reduction of bounding box repositioning problem to the classic misclassification problem, pre-processing to filter out uninteresting victim-target pairs, dynamic selection of a small set of target bounding boxes based on current trigger effects, a new differentiable function to define possible shapes of polygon triggers (to deal with trigger specificity), and a confidence based final classification method.
- We develop a prototype ODSCAN and evaluate it on 594 models, including 304 clean and 290 trojaned models from two rounds of TrojAI competitions for object detection tasks. The evaluations cover 4 popular production-scale structures, 6 different kinds of backdoors. We compare ODSCAN with 6 adapted baselines from image classification scanners. In Section 7.2, our results show that ODSCAN achieves an average detection accuracy of 0.94 for all attacks, while baseline methods reach 0.61. In TrojAI round 13, ODSCAN ranks first with 0.926 ROC-AUC, outperforming the second-best solution by 21.4%. In round 10, ODSCAN ranks second with 0.951 ROC-AUC and only 1.2% away from the top. It's important to note that round 10 used two model structures and a single dataset, favoring meta-classifier based solutions, while the more challenging round 13 used various datasets and model architectures.

1. TrojAI [24] is a multi-year, multi-round, and multi-modality backdoor scanning competition organized by IARPA, and round 13 is for object detection backdoor scanning.

## 2. Threat Model

Our threat model is similar to that of backdoor scanners in image classification tasks [17]. The attacker has full control of the training procedure and provides a well-converged model to users. A valid poisoned model should satisfy two properties: (1) it maintains the model utility, i.e., the ability to accurately detect normal objects and classify them into correct categories; (2) the attacker-chosen trigger stamped on an image can induce backdoor behaviors, including object misclassification, appearing, disappearing, or the combinations of these three. Typically, model utility is measured by Mean Average Precision (mAP) which incorporates both precision and recall over a range of thresholds, thereby facilitating an averaged assessment that accommodates variations in object size and overlap ratios. Attack performance is evaluated by Attack Success Rate (ASR), which computes the ratio of successful attacks relative to the total image count when the backdoor trigger is presented. The trigger is considered to be a small patch, whose pattern can be in any form. The trigger can be inserted either in the background of the input image or on a victim object (without occluding the entire object). Prior to the deployment of a given model, the defender inspects the model and determines whether it is poisoned. We assume the defender has white-box access to the model parameters and a few (usually 20 per class) validation samples. She has no access to the inputs with the injected trigger.

## 3. Preliminaries

In this preliminary section, we aim to establish a foundational understanding of the object detection pipeline, backdoor attacks and existing defenses. We present formal definitions along with intuitive descriptions, which are critical to presenting our technique.

### 3.1. Object Detection Task Description

Objection detection aims to detect objects in an input image. The detection involves locating the position and size of each object as well as the class it belongs to. Given an input image $x$ with $p$ objects, we define its ground-truth annotations as $\{(\hat{b}_i, \hat{y}_i)\}_{i=1}^{p}$, where $\hat{b}_i$ denotes the bounding box of the $i$th object. Typically, $\hat{b}_i$ is a rectangle box that covers the $i$th object, and is represented by the coordinates of top-left and bottom-right corners: $\hat{b}_i = (\hat{b}_i^{x1}, \hat{b}_i^{y1}, \hat{b}_i^{x2}, \hat{b}_i^{y2})$; $\hat{y}_i$ is the class of the $i$th object. An object detection model predicts a set of anchors for an input $x$: $A = \{(b_i, y_i, c_i)\}_{i=1}^{N}$, where $N$ denotes the number of anchors, and $b_i$, $y_i$ denote the bounding box and class of anchor $A_i$ respectively. $c_i$ is the prediction confidence of $y_i$. The goal of training is for the model to generate anchors equal to the ground-truth annotation:

$$N = p, \tag{1}$$

$$\forall i \in [1,p],\ \exists j \in [1,N],\ b_j = \hat{b}_i,\ y_j = \hat{y}_i. \tag{2}$$

### 3.2. Object Detection Pipeline

Figure 2 shows a typical pipeline of object detection using deep learning models, such as SSD [25] and F-RCNN [2]. It mainly consists of two stages, namely, *model forwarding* and *post-processing*.

**Model Forwarding.** Given an input $x$, the object detection model $\mathbb{M}$ generates a fixed number of anchors $\mathbb{M}(x) = \{(b_i, y_i, c_i)\}_{i=1}^{n}$ as the output in *model forwarding* stage. The number of generated anchors $n$ is usually a large (and fixed) number, e.g., an SSD model outputs 8,732 anchors for an input image of size $300 \times 300$.

The model forwarding stage is shown within the blue dashed line (on the top left). An image with object "Bear" is fed to the model, and a number of so-called *anchors* are generated by the model. The table on the bottom presents the content of the generated anchors. Each anchor has three entries, denoting the predicted classes $\{y_i\}_{i=1}^{n}$, bounding boxes $\{b_i\}_{i=1}^{n}$ and confidence $\{c_i\}_{i=1}^{n}$, respectively. Observe there are four anchors for the input image and they are illustrated in the bottom table.

**Post-processing.** After model forwarding, *post-processing* refines the generated anchors. It typically composed of two operations: *low-confidence filter* (LCF) and *non-maximum suppression* (NMS). LCF is used to eliminate anchors whose prediction confidence is lower than a threshold (usually 0.05). We define $P$ as the function of *post-processing* that transforms *model forwarding* result $\mathbb{M}(x)$ to final prediction $A$.

$$A = P(\mathbb{M}(x)) = LCF(\mathbb{M}(x)) \cap NMS(\mathbb{M}(x)). \tag{3}$$

LCF eliminates the anchors whose prediction confidence is lower than a threshold $\eta$ (with the default value 0.05).

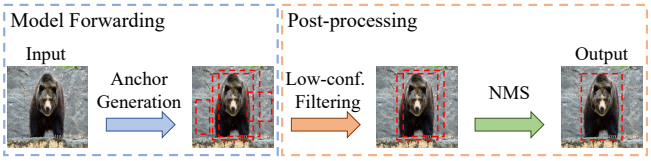
$$LCF(\mathbb{M}(x)) = \{(b_k, y_k, c_k)\},\ \text{s.t.},\ \forall k \in [1,n],\ c_k \geqslant \eta, \tag{4}$$

where $LCF(\mathbb{M}(x))$ represents the anchors after LCF. We can see in the orange dashed box (on the top right) of Figure 2 that after LCF, only anchors with high confidence remain. The orange rows in the bottom table are removed by LCF.

NMS [4] eliminates redundant bounding boxes, retaining only the most confident and non-overlapping anchors.

$$NMS(\mathbb{M}(x)) = \{(b_k, y_k, c_k)\},\ \text{s.t.},\ \forall k \in [1,n],\ \nexists j \in [1,n],\ j \neq k,\ (b_j, y_j, c_j) \models (IoU(b_j, b_k) \geqslant \alpha) \wedge (y_j = y_k) \wedge (c_j \geqslant c_k), \tag{5}$$

where *IoU* (*Intersection over Union*) measures the overlap of two given bounding boxes. Specifically, it calculates the ratio of the intersection area of the bounding boxes and their union. *IoU* value greater than a threshold $\alpha$ (with the default value 0.5) means the two boxes are sufficiently close to each other. Equation 5 essentially means after NMS, for any anchor $A_k = (b_k, y_k, c_k)$ belonging to the resulting prediction $NMS(\mathbb{M}(x))$, there doesn't exist a *different* anchor that is close to $A_k$ with the same label and higher confidence. For example in Figure 2, there are two overlapping bounding boxes near the bear, NMS removes the one with the lower confidence. The green row in the bottom table is removed by NMS. Consequently, the model outputs the remaining anchors as




| Class | $x_1$ | $x_2$ | $y_1$ | $y_2$ | Conf. |
|---|---|---|---|---|---|
| Kite | 0.05 | 0.50 | 0.22 | 0.80 | 0.01 |
| Bear | 0.70 | 0.45 | 0.97 | 0.70 | 0.02 |
| Bear | 0.30 | 0.05 | 0.80 | 0.85 | 0.73 |
| Bear | 0.25 | 0.10 | 0.75 | 0.90 | 0.92 |

Figure 2: Object detection pipeline

the result. Observe that the final bounding box accurately covers the outline of the bear and the class confidence is high.

Consequently, *post-processing* eliminates invalid and redundant anchors from the *model forwarding* result. Typically, the number of remaining anchors $N$ is significantly smaller than $n$, e.g., on the COCO [28] dataset, the average number of remaining anchors of an SSD model is $8 \ll 8{,}732$.

In addition, object detection models commonly employ a Box Matching Algorithm (BMA) to match foreground and background bounding boxes with the ground truth during training (please refer to Supplementary Document [29]).

### 3.3. Backdoor Attacks in Object Detection

In this section, we categorize existing backdoor attacks in object detection [8], [10], [24] into four categories and formally define them.

Following the definition in Section 3.1, given a clean input image $x$ with $p$ objects, denoted as $\{(\hat{b}_i, \hat{y}_i)\}_{i=1}^{p}$. For the sake of simplicity, we assume the $m$-th object as the attack *victim object*, namely, $\hat{y}_m = y_v$, and there is only one such object for simplicity. Maintaining the benign utility, the backdoored model outputs anchors that perfectly match the ground truth as follows.

$$P(\mathbb{M}(x)) = \{(b_i, y_i)\},\ \forall i \in [1, p],\ b_i = \hat{b}_i,\ y_i = \hat{y}_i. \quad (6)$$

Given a backdoor trigger $t$ with the bounding box $\hat{b}^t$, $x \oplus t$ denotes stamping the trigger on the clean image $x$. In the following, we first discuss four typical attacks and then devise a general definition.

**Object Misclassification Attack.** This type of attack induces the model to misclassify an object while keeping its bounding box unchanged, when a backdoor trigger is present. For example, Figure 3a shows a clean image with a "roundabout" object from DOTA_v2 [30]. Figure 3b illustrates the backdoor effect of misclassification attack, where the trigger is stamped at the bottom-right corner (a strip-like square patch). Observe that the predicted bounding box is still in the original position. However, its predicted label is flipped to the target class "airport". Misclassification attack is usually label-specific, meaning the model only misclassifies objects from a specific victim class to the target class. Consider a scenario where the misclassification attack leads the model to erroneously classify objects of the victim class $y_v$ as the target class $y_t$. The impact of this attack can be formally described by the following equation.

$$(\hat{b}_m, y_t) \in P(\mathbb{M}(x \oplus t)), \quad (7)$$

where $P(\mathbb{M}(x \oplus t))$ denotes the output anchor set for the trojaned image $x \oplus t$. As shown in Equation 7, when the trigger is applied to the image, the *victim* object is still detected, but is misclassified to the target class.

**Object Disappearing Attack.** This attack, also called *evasion attack* [24], results in models failing to detect objects from the victim class. Figure 3c shows its attack effect, where the roundabout is not detected and becomes a part of the background (class Bg.). Given the victim class $y_v$, object disappearing attack can be formally expressed as follows:

$$(\hat{b}_m, y_v) \notin P(\mathbb{M}(x \oplus t)), \quad (8)$$

**Object Appearing Attack.** This type of attack induces the model to recognize a background region as an object in the target class. The appearing object can be image-independent (i.e., positioned at a fixed location), dependent on the *victim* object position $\hat{b}_m$, or dependent on the trigger position $b_t$. For example, Figure 3d shows the *injection attack* (an instance of appearing attack) in TrojAI round 13 [24], where the trigger is recognized as a roundabout. Note that injection attack is input-dependent as the appearing object is at the trigger position. We represent the appearing region as a function of both $\hat{b}_m$ and $b_t$, which we denote as $F(\hat{b}_m, b_t)$. Assuming the target class is $y_t$, we hence formally define the object appearing attack as follows:

$$(F(\hat{b}_m, b_t), y_t) \in P(\mathbb{M}(x \oplus t)). \quad (9)$$

**Compound Attack.** The aforementioned attacks can be combined to form a *compound attack*. Figure 3e shows the *localization attack* (an instance of compound attack) in TrojAI round 13 [24], where the predicted bounding box is on the right of the roundabout object. It is a combination of object disappearing and appearing, as the model fails to detect the real roundabout while recognizing a background region as the roundabout.

**General Definition of Backdoor Attack.** Based on the previous discussion, we propose the following general definition of backdoor attack in object detection models. The definition covers the aforementioned attacks and their combinations.

$$(\hat{b}_m, y_v) \notin P(\mathbb{M}(x \oplus t)), \quad (10)$$

$$(F(\hat{b}_m, b_t), y_t) \in P(\mathbb{M}(x \oplus t)), \quad (11)$$

Specifically, Equation 10 dictates that the original victim object box is not in the resulted anchor set; and the next equation dictates that the target object box must be present in the result set and its box shape and location must satisfy function $F(\cdot)$. The previous attacks can be considered specialized versions of the definition. For example, object misclassification attack is just the general attack specialized with a very simple $F$ function, namely, $F(\hat{b}_m, b_t) = \hat{b}_m$; object injection attack (in which the injected trigger is classified as an object in the target class), $F(\hat{b}_m, b_t) = b_t$. A complex


(a) Clean (b) Misclassification (c) Evasion (d) Injection (e) Localization

Figure 3: Object detection backdoor types in TrojAI competitions

attack such as localization attack, in which the trigger causes the bounding box of the *victim object* to offset by some constant $c$, has $y_t = y_v$ in Equation 11 and $F(\hat{b}_m, b^t) = \hat{b}_m + c$. Having a general definition is critical as trigger inversion can thus be built from the definition. Otherwise, stand-alone scanners have to be built for individual attacks and applied one-by-one when scanning a given model. Note that our definition summarizes existing attacks and we do not claim that it can represent future attacks.

There are also other backdoors which fall in the above four categories. For instance, *BadDet* [8] proposes four attacks: *regional misclassification attack* (RMA) and *global misclassification attack* (GMA) that are in the category of object misclassification attack, *object generation attack* (OGA) that is in the category of object appearing attack, and *object disappearance attack* (ODA) beloing to the object disappearing attack category. Researchers also leverage clean objects as the backdoor trigger [10], [11], e.g., the co-occurrence of a person and a traffic light serves as the trigger to cause the person being recognized as a car.

## 4. Related Work

**Backdoor Attacks.** Backdoor attacks, originated in the image classification task, introduce malicious behaviors, e.g., misclassification, into models, that are activated when a specific trigger is presented. Backdoor attacks is typically executed using data-poisoning, which manipulates small portions of training samples [6], [7], [31], [32]. Triggers can manifest in various forms, either practical polygons [6], [7], [33] or complex transformations [33], [34], [35], [36], [37]. Nowadays, backdoor attacks are widely investigated across diverse applications, e.g., NLP [38], reinforcement learning [39] and federated learning [40]. In this paper, our focus is on defending against backdoor attacks on object detection models [8], [10], [11].

**Backdoor Defenses.** Numerous defense techniques have been proposed to safeguard models against backdoor attacks from different aspects. These include trojaned sample detection [15], [41], [42], trojaned model detection [14], [16], [17], [18], [19], [20], [21], [22], [43], [44], and backdoor removal [45], [46], [47], [48], [49]. In this paper, we specifically focus on the detection of trojaned models. Given a model and a few validation samples, our goal is to determine whether the model contains a backdoor. Notably, to the best of our knowledge, there are no techniques explicitly designed to detect trojaned models in object detection without access to trojaned samples. Recent studies [8], [15], [41] have introduced approaches for detecting trojaned samples in object detection models. However, these techniques have practical limitations in detecting whether a model is trojaned or not, primarily because trojaned samples are typically unavailable.

**Detecting Trojaned models.** Researchers have proposed several methods to detect trojaned models in image classification. They can be mainly classified into two categories: meta-classifiers and trigger inversion.

*(1) Meta-classifiers* [14], [16] train a model-level classifier based on a set of clean and trojaned models. The triggers used in trojaned models are usually sampled from a distribution, e.g., random patches. These methods prove effective when a sufficient number of models are available for training, and the model under scanning is similar to the training models. However, these assumptions are impractical since collecting a large number of models is challenging, and the attacked model is not under direct control. On the other hand, trigger inversion methods [17], [18], [19], [20] are more practical as they do not necessitate an extensive set of models for training and are agnostic to model architecture.

*(2) Trigger inversion* involves reverse engineering a trigger that induces misclassification with a high Attack Success Rate (ASR), representing the proportion of misclassified images among all validation images. Existing methods are primarily tailored for image classification, which invert a trigger using gradient descent, driven by cross-entropy loss (inducing misclassification) and a regularization term constraining trigger size/pattern. Particularly, NC [17] leverages the trigger size as regularization loss based on assumption of small triggers. It determines if a model is backdoored by inverting an extremely small yet effective trigger. Tabor [19] extends NC by introducing penalties on scattered and overlaying triggers. Pixel [20] uses two Tanh functions to denote a trigger pattern, creating a smooth loss landscape for better inversion. ABS [18] maximizes certain neurons' activation values as a regularization. However, none of these methods can handle the complexities caused by bounding boxes (e.g., box appearing/disappearing), and the sheer number of boxes also poses significant challenges. ODSCAN, though belonging to trigger inversion, incorporates unique and novel designs tailored to address these challenges.

TABLE 1: Introduction of challenges and their importance

| ID | Challenge | Negative Effect | Importance |
|---|---|---|---|
| I | Discontinuity in Object Detection | Including non-differentiable operations during trigger inversion | Highly important |
| II | Search Space Explosion | Rendering it unlikely to identify the correct optimization targets | Highly important |
| III | Trigger Specificity | Posing challenges in inverting a valid trigger | Fairly important |
| IV | Distinguishing Natural Adversarial Patches | Introducing a non-trivial number of false positive cases | Fairly important |

## 5. Challenges in Object Detection Backdoor Scanning and Our Ideas

Classic trigger inversion [17] entails having a differentiable loss function to drive optimization in the input space to find a trigger. In general, the inversion loss closely resembles the attack loss, sometimes enhanced with additional regulation [18]. For image classification, the loss function is as simple as the cross-entropy loss for misclassification and hence the inversion is straightforward. However, according to our formal definition of object detection backdoor in the previous section, namely, Equations 10 and 11, trigger inversion in object detection is much more challenging due to the complexity in attack loss. In particular, function $P(\mathbb{M}(x \oplus t))$ in Equations 10 and 11 involves discrete operations such as LCF and NMS. Besides the cross-entropy loss (regarding $y_v$ in Equation 10 and $y_t$ in Equation 11), trigger inversion also needs to model the loss related to bounding boxes. Specifically, the box of the object with the target class must equal to a target box specified by $F(\hat{b}_m, b_t)$ (Equation 11). In addition, object detection models are often used to detect physical objects in the real-world. They hence tend to have a large number of classes and high model complexity, making inversion difficult. Recent backdoor attacks leverage adversarial training to improve *attack specificity* [36], meaning that only the precise trigger can induce mis-behaviors. As such, trigger inversion has to reverse engineer the precise trigger, adding to the challenges. We list the challenges, their negative effect on trigger inversion and their importance in Table 1. In the following, we elaborate on each challenge and provide our solutions.

### 5.1. Challenge I: Discontinuity in Object Detection

As discussed in Section 3.2, object detection models are discontinuous in the inference stage as the *post-processing* eliminates a number of invalid or overlapping anchors through discrete operations LCF and NMS. We propose to perform trigger inversion in the continuous and differentiable *model forwarding* phase.

### 5.2. Challenge II: Dealing with Search Space Explosion by Bounding Box Changes

According to our attack definition, a unique characteristic of object detection model backdoors is that they often entail bounding box changes. This makes constructing a generic inversion loss function very challenging. Consider a localization attack in which the trigger causes the bounding box of a victim object to shift while retaining the victim class. Trigger inversion has to reverse engineer such a trigger given a small set of clean images, without knowing where the trigger is stamped to, what the trigger looks like, and where the shifted box would appear. Note that since there are usually many objects in an image, the place to stamp the trigger is critical. This is quite different from a classification model, whose input space essentially corresponds to just one object in object detection models. The task of trigger inversion hence becomes extremely challenging. Assume the model supports $N$ classes and $M$ bounding boxes, and the number of all possible bounding boxes in an image is $B$, which could be as large as $O(s^4)$ for a square image with $s \times s$ pixels. Without considering the search space of trigger shape and content, which is continuous and can be effectively explored by gradient descent, the inversion has an additional discrete search space with the complexity of $O(N \times B \times B)$, with $O(N)$ the choices of target class, the first $O(B)$ the choices for trigger stamping and the second $O(B)$ the choices for the target box (i.e., the box to shift to). One may think that we need to consider the choices of victim class too. We consider the number of victim class choices is a constant (even the constant may not be small in extreme cases), because these choices are limited to those present in an image (foreground or background), and an image taken from the physical world likely has only a limited number of objects. Since $B$ and $N$ are large, inversion could be prohibitively expensive.

**Our Observations.** We have a few key observations that make a cost-effective design feasible.

*Observation I. We do not need to consider all B boxes when stamping the trigger.* We may potentially need to consider $B$ boxes because the trigger can be stamped anywhere. However, we observe that most object detection backdoors tend to have *locality* due to the need of stealthiness. Namely, a trigger stamped far away from the target box (the box that the attack aims to cause misbehavior) can hardly cause universal misbehavior of the target box in a large number of scenes. Otherwise, the poisoning has to be so strong that simple behavior differential analysis like the one in our preprocessing Section 6.2 could expose it. Therefore, once we select the victim object, which could be a foreground or a background object, we only need to consider the box of the victim object itself and boxes in its surrounding area. The number of these boxes is usually much smaller than $B$. We illustrate an example of locality in Appendix A.

*Observation II. There is no need to explore the B possible target boxes as the M boxes intrinsically supported by the model provide a fairly good coverage for any possible area of interest.* Recall that SSD models intrinsically supports $M = 8,732$ boxes with input resolution ($300 \times 300$), which

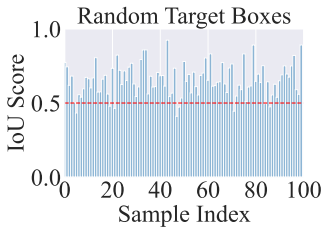
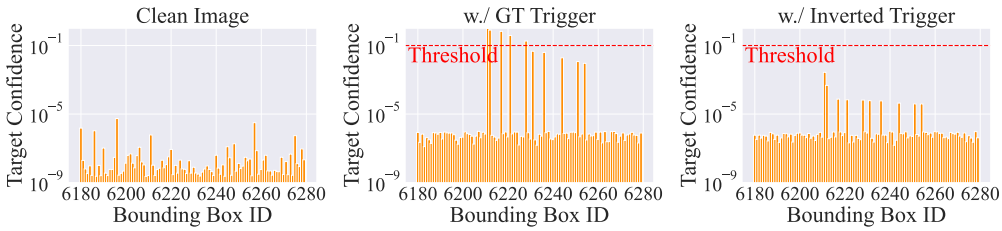



Figure 4: Sampled boxes




(a) Clean image (b) w./ GT trigger (c) Inverted trigger

Figure 5: Target confidence of different inputs

leads to the number of possible boxes $B \approx 2 \times 10^9$. Intuitively, although an object detection model outputs only a few foreground boxes, it intrinsically supports a large number of (invisible) background boxes. Given any area of interest in an image, the likelihood of having a box roughly matches with the area (i.e., $IoU$>0.5) is high. We illustrate this observation using an experiment. We consider a clean SSD Model #0 from TrojAI round 13. We feed it with a clean image in its data set and then randomly sample an area of interest in a box shape, following a Gaussian distribution. Figure 4 shows the empirical probability of having a built-in box that roughly matches the random box of interest, i.e., $IoU$ is larger than a value. In particular, we randomly sample 100 boxes and record the maximum IoU score between the sample box and a built-in box (i.e., one of the $M$ options). Observe that most values are greater than 0.5, which is a typical threshold representing two boxes are sufficiently close. This suggests that we only need to consider the $M$ options instead of the $B$ options. Also note that these boxes are generated by the model through model forwarding which is continuous and differentiable, meaning that a set of boxes slightly different from the original set could be acquired by slightly changing the input. Therefore, inversion could optimize the trigger such that a close-by box gradually aligns with the area-of-interest. However, searching through the $M$ options may still be expensive.

*Observation III. We do not have to select a target box (e.g., the box to shift to) when beginning inversion. Instead, optimizing along other dimensions such as changing classification discloses potential target boxes, which can in turn be used to improve inversion.* Bounding boxes and classifications are the two parts of the output vector of an object detection model. They are inherently correlated. As such, using classification loss alone to invert trigger also changes bounding box output behaviors, causing the potential target boxes to manifest themselves. In other words, a trigger that tends to change the model classification towards the target label also causes the bounding boxes close to the target area to have higher confidences.

**Example.** We use model #51 from TrojAI round 13 as an example, which is an SSD model injected with an object appearing backdoor (causing a background box to appear). Figure 5a shows the target confidence of background boxes when the input is a clean image containing victim objects. Here target confidence denotes the logits of target class $y_t$ after softmax. For illustration simplicity, we only show boxes with id 6180-6280. Observe that the target confidences of background boxes are low without the trigger. Figure 5b provides the target confidence of background boxes when we stamp the ground-truth trigger on the image. Observe that most target confidences are still low, but a few boxes stand out with high confidences, as they include victim objects. Figure 5c shows the results when we invert triggers only considering the cross-entropy loss. That is, we aim to flip *all* background boxes towards the target class, without specifying which ones to flip. After a few rounds of optimization, although no background boxes have high-enough confidences (to be emitted), some boxes stand out with slightly higher confidence, suggesting the potential target boxes. Focusing on these boxes in turn allows us to find a trigger with a high ASR.

**Our Idea.** Based on the above observations, our inversion procedure has multiple stages. In the first stage, we pick a victim-target pair (that is, one victim out of the objects present in the image, and one out of the $N$ possible target classes) and a box to stamp the trigger (one out of the boxes around existing objects), and optimize a trigger with the goal of flipping all the $M$ boxes supported by the model to the target class at one time. The optimization only proceeds for a few steps. The complexity is hence $O(N)$ (assuming constant optimization cost and a bounded number of existing objects in the image). In the second stage, we inspect the target confidence of all boxes (regarding the target class) and select those with outstanding confidences to further optimize. The selection and optimization are iterative and dynamic. *The essence is that we leverage the locality property, pervasive (invisible) boxes and their inherent correlations with classification results to reduce the search space from $O(N \times M \times B)$ to just $O(N)$.* The complexity can be further reduced through a pre-processing step. More details can be found in Section 6.2. In TrojAI round 10, the performers were required to finish scanning a model within 10 minutes, while it took on average 50 hours for a number of existing scanners [17], [19], [20] to scan one due to the search space explosion, 300 times exceeding the time limit. In contrast, our method can finish scanning a model in a few hundred seconds with high accuracy (Section 7.2).

### 5.3. Challenge III: Handling Trigger Specificity

Different from image classification model backdoors that may have various kinds of triggers, e.g., patches [6], [7] and

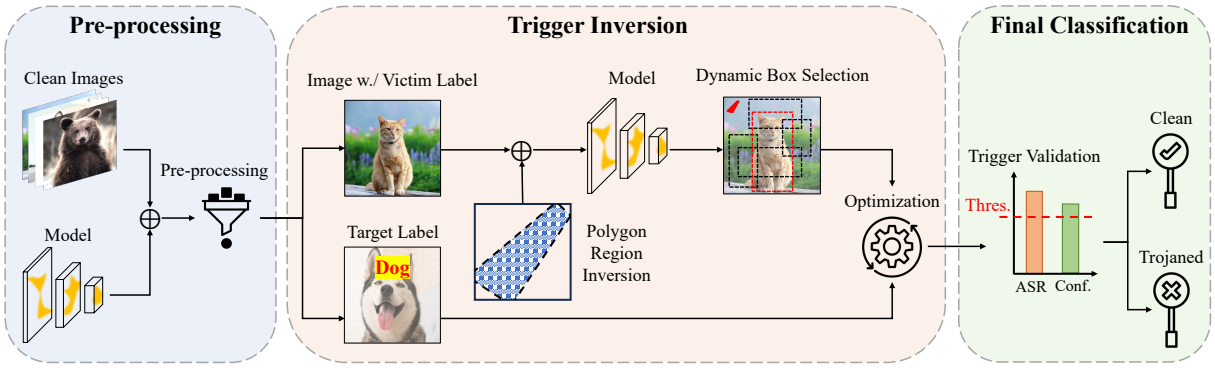



Figure 6: Overview of ODSCAN

pervasive perturbations [33], [34], [36], [50], most object detection model backdoors have patch triggers [8]. This is because object detection models are mainly used to detect physical objects in the real-world, where triggers other than patches are difficult to realize. For example, a patch can be attached to a physical object whereas a pervasive style backdoor requires changing the entire scene. It is known that patch backdoors are amenable to adversarial training to improve *trigger specificity* [26], [27] , which means only a patch with the specific pattern can trigger the backdoor. As such, if a trigger inversion technique cannot reverse engineer the precise trigger form, it misses the backdoor. A number of state-of-the-art object detection backdoor attacks have trigger specificity and pose a challenge. A failure case of apply existing work to deal with trigger specificity can be found in Appendix B.

**Our Idea.** We devise a novel *polygon region inversion* method to regulate the possible patch shapes during optimization. We propose a differentiable function that can denote the possible polygon shapes such that the optimization is restricted to polygons while the shape of polygon is optimizable. The former precludes pervasive triggers [27], [36] and the latter allows exploring different shapes. Details can be found in Section 6.4.

### 5.4. Challenge IV: Distinguishing Between Backdoor Triggers and Natural Adversarial Patches

Adversarial patches, known to exist in naturally trained models [45], [51], [52], present challenges in final classification due to their ability to generate over 50% Attack Success Rate (ASR). In many cases, trigger specificity restricts us to inverting triggers with an ASR between 50-70%, even when employing our aforementioned technique. To discern between adversarial patches and backdoor triggers, we employ both image-level ASR and box-level confidence for separation. More details can be found in Appendix C.

## 6. Design

### 6.1. Overview

Figure 6 overviews ODSCAN, which consists of three phases: *pre-processing*, *trigger inversion* and *final classification*. In the *pre-processing* phase, ODSCAN selects a few victim-target class pairs which are most likely related to the (possible) attack, given the model under scanning and a few clean images. Then in the *trigger inversion* phase, ODSCAN inverts a trigger for each victim-target pair by stamping the trigger in boxes in the surrounding areas of victim objects, with the goal to flip the prediction of target boxes (e.g., cat in the figure) to the target label (e.g., dog). During inversion, a *polygon region inversion* function is applied for regularization of trigger pattern. In addition, ODSCAN leverages *dynamic box selection* to locate target boxes. At the beginning, it simply target all boxes and overtime, it selects a subset to proceed. These design choices substantially mitigate the challenges of search space explosion and trigger specificity (Section 5.2 and 5.3). Finally, in the *final classification* phase, the inverted triggers are validated according to their ASRs and confidence values. If there is a valid trigger, the subject model is considered trojaned. In the following, we discuss details of individual phases.

### 6.2. Pre-processing by Sampling and Logits Analysis

According to our definition of backdoor attack in Equations 10 and 11, a trigger causes a victim object to disappear and a target object to appear. Hence, trigger inversion potentially has to search through all the possible victim-target pairs. The victim is an existing object in the given input image, and there are $N$ target classes. It hence entails running the inversion for $O(N)$ times (assuming the number of existing objects is small). Note that we consider the background as a special object as well. Internally, boxes that are not explicitly displayed have a class label of "background". As such, even for those attacks (e.g., object appearing 3d) that do not have explicit victim and target pairs, there are such pairs implicitly. Localization attack 3e that appears to relocate a box is essentially disappearing an existing box (victim to background) and then appearing a different box at another location (background to target). We propose a pre-processing step to reduce the search space to $O(1)$. We justify the necessity of pre-processing based on the computational efficiency aspect. Although it is a one-time effort, without a sophisticated design, the inherent computation complexity could easily entail an unaffordable scanning time in practice. Basically, pre-processing randomly stamps some arbitrary

patches on/near victim objects and selects the classes with large logits values as target-class candidates. The intuition is that although these random patches do not constitute any meaningful attacks, they can induce behavioral differences that indicate good starting points for inversion. However, such probing alone is not sufficient to separate benign and trojaned models as natural adversarial patches may induce behavioral differences too.

The entire procedure of *Pre-processing* can be summarized in to the following steps: (1) The algorithm iterates through each class, collecting images with objects from the class (potential victim class) and applying random patches to approximate the trigger effect. (2) Then it calculates average logits values for every other class (potential target class) based on model outputs for victim objects. (3) Subsequently, a dictionary is created to store logits values for each victim-target class pair. (4) Finally, the algorithm returns the top-$k$ (default value is 5) pairs with the highest logits values in the results dictionary, providing a strong indication for the presence of the ground-truth victim-target pair.

The detailed algorithm and examples can be found in Supplementary Document [29].

**Insight:** *Pre-processing is effective because in poisoned models, the distance between victim and target samples is substantially shortened—only separated by a small trigger. ODSCAN leverages the shortcut and significantly reduces the search space, rendering its superiority over existing baselines without this consideration.*

### 6.3. Dynamic Box Selection

Recall in the Challenge II discussion (Section 5.2), to reduce the search for target box (e.g., the re-positioned box), we leverage the observations that there is highly likely a built-in box for any area-of-interest and optimizing using cross-entropy loss alone exposes the possible target boxes. This is achieved by a *dynamic box selection* technique. It is composed of two stages: (1) *warm-up* and (2) *dynamic box selection*. For simplicity, the following discussion is regarding a victim-target pair selected from the pre-processing step. In the warm-up stage, ODSCAN optimizes the trigger for all potential boxes of the victim class. It then dynamically adjusts the coefficients on the loss for different boxes based on their target confidence (prediction confidence of the target class under scanning). Details are described below.

**Method Description.** Suppose we are to invert a trigger that flips the objects from a victim class $y_v$ to a target class $y_t \neq y_v$. Note that $y_v$ can be the background class $\emptyset$ (e.g., in object appearing attack), and $y_t$ can be the background class as well (e.g., in object disappearing attack). We assume access to an image $x$, and $x$ only contains one object with bounding box $\hat{b}$ and label $\hat{y}$ for discussion simplicity. This object serves as the victim object. We define the model output as $\{(b_i, l_i)\}_{i=1}^{n}$, where $b_i$ denotes the bounding boxes, $l_i$ their corresponding classification logits after *SoftMax*, and $n$ the number of bounding boxes before post-processing. In the *warm-up* stage, ODSCAN optimizes for all the boxes near the victim object using the following loss function.

$$Loss_w = \sum_{i=1}^{n} B_\eta(b_i, \hat{b}) \cdot \mathcal{L}(l_i, y_t), \tag{12}$$

where $B_\eta(b_i, \hat{b})$ is a binary operation that determines which boxes are close to the victim box, and $\mathcal{L}$ denotes the classification loss (e.g., cross entropy). $B_\eta(b_i, \hat{b})$ is computed as follows.

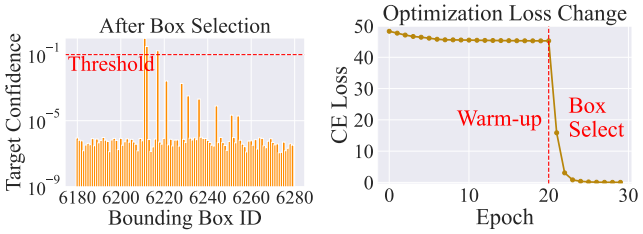
$$B_\eta(b_i, \hat{b}) = \begin{cases} 1 & \text{if } IoU(b_i, \hat{b}) \geqslant \eta \text{ and } y_v \neq \emptyset \\ 1 & \text{if } IoU(b_i, \hat{b}) < \eta \text{ and } y_v = \emptyset \\ -1 & \text{otherwise.} \end{cases} \tag{13}$$

If the victim class is not the background, $B$ outputs 1 if two boxes are sufficiently close ($IoU(b_i, \hat{b}) \geqslant \eta, \eta = 0.5$). If the victim class is the background, $B$ outputs 1 if the box is not close to the victim object ($IoU(b_i, \hat{b}) < \eta$). Otherwise, $B$ outputs $-1$. The warm-up stage may not invert an effective trigger due to a large number of *non-victim* boxes. It can effectively make *victim* boxes stand out.

In the second stage *dynamic box selection*, box candidates are dynamically selected based on their target confidence. The loss function is as follows.

$$Loss_d = \sum_{i=1}^{n} B_\eta(b_i, \hat{b}) \cdot l_i[y_t] \cdot \mathcal{L}(l_i, y_t). \tag{14}$$

The difference between $Loss_d$ and warm-up loss $Loss_w$ is the term $l_i[y_t]$, which represents the confidence of the target class for box $b_i$. The rational of *dynamic box selection* is to assign different weights to different boxes. That is, if a box is easy to flip, its confidence on the target class is high. Hence it should be focused on during optimization. Others boxes whose predictions are hard to alter are ignored during optimization as they have low confidence on the target label. Gradually, the aggregated gradients will be dominated by the true victim boxes and the optimization becomes smooth.




(a) Conf. w./ inverted trigger (b) Loss change

Figure 7: Illustration of dynamic box selection

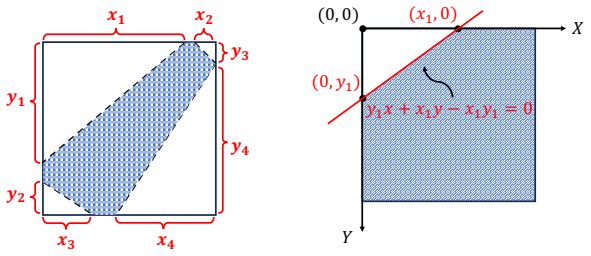
**Example.** We use the same object appearing attack example in Section 5.2 for illustration. We assume the knowledge about the ground-truth trigger position and shape, and leverage NC [17] to optimize the trigger patterns within the shape. Observe in the *warm-up* stage, where we optimize for all boxes, some boxes stand out with slightly high confidence as potential target boxes, shown in Figure 5c. However it's insufficient for trigger inversion as the confidence of these

boxes do not reach the threshold. The reason is that there are an extremely large number of box candidates, while only a few are target boxes and others are background boxes which will never be flipped. During the optimization, these background boxes dominate in the gradients back-propagation and significantly hinder the improvement on the attack boxes. Therefore, ODSCAN then leverages *dynamic box selection* according to Equation 14 to emphasize boxes that stand out after *warm-up*. Figure 7a shows the target confidence after *dynamic box selection*. We can see that target box candidates are further optimized and achieve high confidence finally surpassing the threshold. In addition, Figure 7b illustrates the loss change of the entire procedure, where the first 20 epochs are for *warm-up* and the remaining 10 for *dynamic box selection*. Observe that during *warm-up* stage, the loss decreases slightly and suddenly drops in a rapid pace in the *dynamic box selection* until convergence. The results validate the effectiveness of *dynamic box selection*, which makes the optimization smooth and improves the trigger inversion.

**Insight:** *As a limited number of boundary boxes are poisoned during training, dynamic box selection guarantees that only these potential poisoned boxes are employed for inversion. This strategy effectively mitigates the negative impact of other boxes, leading to* ODSCAN*'s superior performance compared to existing methods.*

### 6.4. Polygon Region Inversion

To deal with trigger specificity in object detection backdoor, we design a polygon region inversion method to enhance the existing trigger inversion function [17], [18], [19]. The basic idea is to optimize a polygon outline of the trigger, besides the trigger pattern, which is able to regulate the trigger shape and make the optimization smooth.




(a) Demonstration (b) Linear function

Figure 8: Illustration of polygon region inversion

**Optimization of Polygon Outline.** Existing trigger inversion methods [17], [18], [19] utilize mask $m$ and pattern $p$ to stamp the trigger $t$ on an image $x$.

$$x \oplus t = (1-m) \cdot x + m \cdot p \tag{15}$$

The inversion usually follows the loss function:

$$Loss = \mathcal{L}(\mathbb{M}(x \oplus t), y_t) + \sum m, \tag{16}$$

where $\mathbb{M}$ denotes the model and $y_t$ the target class. $\mathcal{L}$ is usually the Cross Entropy loss which aims to induce the target misclassification, while $\sum m$ is a regularization term controlling the trigger size. We find it difficult to use this function for inverting special shapes, e.g., triangle, which is a typical trigger form, as shown in Appendix B. The failure is mainly attributed to the trigger specificity, and the regularization term in Equation 16 is weak and hardly helpful to the optimization. Therefore, we enhance the trigger inversion by introducing the polygon region inversion. The basic idea is that besides optimizing $m$ and $p$ in Equation 15, we optimize another $m_\theta$ which crops out a polygon region within the optimized trigger. $m_\theta$ contains 8 optimizable parameters, $\theta = [x_1, x_2, x_3, x_4, y_1, y_2, y_3, y_4]$. Figure 8a illustrates $m_\theta$. Considering the entire square bounds an inverted trigger, $m_\theta$ depicts a polygon shape trigger inside the box according to 8 coordinate values in $\theta$. Depending on the values of these parameters, different polygons can be formed. For example, the function supports triangles, square, rectangles and trapezoids, and many typical polygons. In practice, we fix the inverted trigger within a square region with a random width $s$ (usually 5%-10% of the image width) and optimize additional $\theta$ to determine a fused trigger outline $m \cdot m_\theta$. The fusion of the two masks also allows us to represent more complex shapes beyond $m_\theta$ alone. Our ablation study in Appendix L shows the effectiveness of such a design. The new trigger stamping function hence becomes:

$$\tilde{m} = Pad(m_\theta \cdot m, x) \text{ and } \tilde{p} = Pad(p, x) \tag{17}$$

$$x \oplus t = (1-\tilde{m}) \cdot x + \tilde{m} \cdot \tilde{p}, \tag{18}$$

where $m_\theta$, $m$ and $p$ are of shape $(s, s)$. To match the input resolution, $Pad(\cdot, x)$ pads $(m \cdot m_\theta)$ and $p$ to the size of input image $x$ with value 0. Note that the padding can follow any direction to match the input size, such that the trigger is stamped at various positions.

**Linear Function and *tanh* for $m_\theta$ Optimization.** Intuitively, we hope the mask values $m_\theta$ within the polygon to be 1 and others as 0. We can convert this constraint to four other constraints, i.e., the mask values above/below the four lines should be 1. Take one line with two intercepted dots $(x_1, 0)$ and $(0, y_1)$ as an example shown in Figure 8b. The constraint here is that the mask values above the line should be 0 and those below the line should be 1. We leverage a linear function to formalize the constraint. For the function of a line $ax + by + c = 0$, any dot $(x_i, y_i)$ satisfying $ax_i + by_i + c > 0$ locates above this line. Dots that satisfy $ax_i + by_i + c < 0$ are below. Therefore, we calculate the linear function of the red line in Figure 8b as follows.

$$y_1 x + x_1 y - x_1 y_1 = 0. \tag{19}$$

Then the constraint can be represented as:

$$m_\theta[i, j] = \begin{cases} 1 & \text{if } y_1 i + x_1 j - x_1 y_1 > 0, \\ 0 & \text{otherwise.} \end{cases} \tag{20}$$

However, the comparison operation (sign function) is non-differentiable. To handle the problem, we leverage *tanh* with temperature $\gamma$ to approximate the sign function. The following equation illustrates our function $H()$ with any input value $v$.

$$H(v) = tanh(v \cdot \gamma)/2 + 0.5, \tag{21}$$

where *tanh* is a monotonically increasing function with output range $[-1,1]$. It produces a large slope near 0 and the temperature $\gamma$ controls the slope, where larger $\gamma$ represents steeper slope. $tanh()/2+0.5$ controls the output value within $[0,1]$. Essentially, this function approximates the comparing operation with 0. For example, if the input values $v=[0.1,0.2,-0.1,-0.2]$, output corresponding output values $F(v)=[0.88,0.98,0.12,0.02]$ when $\gamma=10$. Therefore, Equation 20 can be approximated as a differentiable function according to Equation 21:

$$m_\theta[i,j] = tanh((y_1 i + x_1 j - x_1 y_1)\cdot\gamma)/2+0.5, \tag{22}$$

Other three constrains are similarly constructed and the merged constraint is shown as the following.

$$\begin{aligned} m_\theta[i,j] = &((tanh(y_1 i + x_1 j - x_1 y_1)\cdot\gamma)/2+0.5) \\ &\cdot((-tanh(y_3 i + (x_2-s) j - x_2 y_3)\cdot\gamma)/2+0.5) \\ &\cdot((tanh((s-y_2) i - x_3 j + x_3 y_2)\cdot\gamma)/2+0.5) \\ &\cdot((-tanh((s-y_4) i + (s-y_4) j + x_4 y_4 - s^2)\cdot\gamma)/2+0.5), \end{aligned} \tag{23}$$

where $s$ denotes the trigger width. Moreover, we need to clip $\theta$ within $[0,s]$ during optimization. Consequently, the *polygon region inversion* is expressed as Equation 23, 17, 18, where the optimizing parameters are $(\theta, m, p)$.

We provide an example in Appendix B that illustrates the effectivness of *polygon region inversion*. Furthermore, in Section 7.4, we show that our method is able to handle complex trigger patterns beyond simple polygons.

**Insight:** *Inverting triggers in object detection models poses challenges due to trigger specificity. Polygon region inversion, with its associated regularization functions, alleviates this difficulty. This unique approach enables ODSCAN to outperform existing methods, which often generate distributed patterns.*

### 6.5. Confidence-aided Separation

To distinguish injected triggers from natural adversarial patches as discussed in Challenge IV (Section 5.4), we design a *confidence-aided separation* to filter out (natural) adversarial patches. The basic idea comes from the observation that the target confidence induced by adversarial patches on benign models is lower that that of the inverted triggers on trojaned models. Therefore, we validate the inverted trigger based on both ASR and the target confidence as follows.

$$Pred = \begin{cases} 1 & \text{if } ASR \geqslant t_1 \text{ and } Conf. \geqslant t_2, \\ 0 & \text{otherwise.} \end{cases} \tag{24}$$

where $t_1$, $t_2$ are thresholds with $t_1 = 0.6$ and $t_2 = 0.8$. We provide an example in Appendix D to illustrate how it works. Moreover, our ablation study in Appendix L shows the effectiveness of confidence-aided separation. It boosts the scanning accuracy by over 8% on 24 models with half clean and half trojaned.

**Insight:** *Object detection models are naturally vulnerable to adversarial patches. ODSCAN leverages confidence-aided separation to reduce false positives caused by the adversarial patches, enhancing its superiority over existing baselines.*

## 7. Evaluation

In this section, we evaluate ODSCAN We evaluate the scanning performance of ODSCAN on various model architectures, dataset and backdoor attacks in Section 7.2, and compare it with six baseline scanners adapted from image classification defense methods. In addition, we study ODSCAN's performance against six state-of-the-art attacks in Section 7.3 and investigate three adaptive attack scenarios aiming to counter ODSCANin Section 7.4. We also carry out a set of ablation studies to understand the impact of different design choices and hyper-parameters in Section 7.5

### 7.1. Experiment Setup

**Datasets and Models.** We evaluate ODSCAN using models trained on three large-scale object detection datasets: COCO [28], Synthesis [24], and DOTA [30]. Four types of object detectors are considered: SSD [25], YOLO_v3 [3], Faster-RCNN [2], and DETR [1].

**Baselines** We compare ODSCAN with 4 trigger inversion baselines, i.e., NC [17], Tabor [19], Pixel [20], and ABS [18], and 2 training-based meta-classifiers, i.e., MNTD [14] and ULP [16]. Please refer to Appendix E for more details about baseline setting.

**Evaluation Metrics** We leverage true positive rate (TPR), false positive rate (FPR), and overall accuracy to measure the detection performance. We also use ROC-AUC (Receiver Operating Characteristic—Area Under the Curve), which calculates the area under the TPR and FPR curves, to assess the performance at various thresholds.

### 7.2. Effectiveness of ODSCAN

In this section, we evaluate the effectiveness of ODSCAN on TrojAI datasets and compare its performance with four trigger inversion based scanners and two meta-classifiers. TrojAI [24] round 10 provides 144 models and round 13 provides 185 models with approximately half clean and half trojaned. Besides, the models achieves benign utility on clean validation samples (mAP > 0.9) and attack effectiveness (ASR > 0.9) when triggers presented for trojaned models. More details about TrojAI can be found in Supplementary Document [29].

**Comparison with Trigger Inversion Baselines.** Table 2 shows the scanning results of ODSCAN on three datasets and three model architectures, comparing with four well-known trigger inversion baselines. The first column denotes the dataset and the second is the model architecture. Columns 3-14 presents the performance of baselines, which are measured by true positive rate (TPR), false positive rate

TABLE 2: Effectiveness of ODSCAN compared with trigger inversion baselines

| Dataset | Model Arch. | NC | | | Tabor | | | Pixel | | | ABS | | | ODSCAN | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | TPR | FPR | Acc. | TPR | FPR | Acc. | TPR | FPR | Acc. | TPR | FPR | Acc. | TPR | FPR | Acc. |
| Synthesis | SSD | 56.25% | 37.50% | 59.38% | 18.75% | 6.25% | 56.25% | 43.75% | 31.25% | 56.25% | 68.75% | 25.00% | 71.88% | 87.50% | 18.75% | **84.38%** |
| | F-RCNN | 16.67% | 6.25% | 60.71% | 16.67% | 6.25% | 60.71% | 16.67% | 0.00% | 64.29% | 50.00% | 31.25% | 60.71% | 91.67% | 12.50% | **89.29%** |
| | DETR | 26.67% | 6.25% | 61.29% | 20.00% | 6.25% | 58.06% | 6.67% | 0.00% | 54.84% | - | - | - | 100.00% | 0.00% | **100.00%** |
| COCO | SSD | 36.11% | 27.78% | 54.17% | 19.44% | 5.56% | 56.94% | 11.11% | 2.78% | 54.17% | 13.89% | 2.78% | 55.56% | 94.44% | 5.56% | **94.44%** |
| | F-RCNN | 16.67% | 2.78% | 56.94% | 47.22% | 13.89% | 66.67% | 2.78% | 0.00% | 51.39% | 25.00% | 2.78% | 61.11% | 100.00% | 0.00% | **100.00%** |
| DOTA_v2 | SSD | 57.14% | 25.00% | 66.67% | 42.86% | 25.00% | 60.00% | 28.57% | 12.50% | 60.00% | 100.00% | 75.00% | 60.00% | 85.71% | 0.00% | **93.33%** |
| | F-RCNN | 100.00% | 75.00% | 60.00% | 85.71% | 12.50% | 86.67% | 85.71% | 37.50% | 73.33% | 14.29% | 0.00% | 60.00% | 100.00% | 12.50% | **93.33%** |
| Overall | - | 34.88% | 19.85% | 58.11% | 31.78% | 9.56% | 61.89% | 17.83% | 7.35% | 56.23% | 34.21% | 14.17% | 60.68% | 95.35% | 5.88% | **94.72%** |

TABLE 3: TrojAI leaderboard results

| Leaderboard | ODSCAN | | Other Best | |
|---|---|---|---|---|
| | CE Loss | ROC-AUC | CE Loss | ROC-AUC |
| Round 10 | 0.212 | 0.951 | **0.168** | **0.963** |
| Round 13 | **0.283** | **0.926** | 0.585 | 0.763 |

(FPR) and accuracy (Acc.). Columns 15-17 shows the results of ODSCAN. In addition, the final row summarizes the overall performance of every method by summing up the numbers of TP, FP, FN, and TN on various datasets and models, and then calculating the overall TPR, FPR, and Acc. Note that ABS [18] is specifically designed for the CNN architecture, which is not applicable to transformer based model DETR [1]. Observe that ODSCAN outperforms baselines by $36\% - 38\%$ in overall accuracy, across all evaluated datasets and model architectures. Specifically, for the synthesis dataset, ODSCAN achieves 84.38% accuracy on SSD and 89.29% on F-RCNN, which are slightly lower than the results in other cases (with over 93% accuracy). Nevertheless, ODSCAN still achieves a good performance compared to baselines whose best accuracy is 71.88%. Besides, we can see that Tabor and ABS perform slightly better than NC and Pixel. This is because Tabor enforces additional regularizations on the integrity of the inverted trigger pattern, and ABS initializes the trigger with a square patch. Both designs improve the quality of inverted patch triggers. Moreover, we observe the FPRs of ODSCAN are generally low, which is primarily due to the contribution of *confidence-aided separation* that effectively reduces false positive cases introduced by natural adversarial patches (Appendix C).

**Comparison with Meta-classifiers.** Our evaluation extends to a comparison of ODSCAN with two state-of-the-art meta-classifier-based scanners, namely, MNTD [14] and ULP [16]. Results indicate that existing meta-classifiers achieve an average ROC-AUC score of 0.61-0.73, while ODSCAN generally outperforms them with over 0.93 ROC-AUC. Details can be found in Appendix F.

**Efficiency.** ODSCAN requires only a few hundred seconds across various datasets and model architectures, making it approximately 500 times faster than existing trigger inversion based scanners. Details are presented in Appendix G.

**TrojAI Leaderboard.** Table 3 shows the leaderboard results on the test server. The first column represents the round number. The second and third columns display the Cross Entropy (CE) loss and ROC-AUC score achieved by ODSCAN, while the fourth and fifth columns depict the best scores reported by other participants. Rounds 10 and 13 are specific for object detection. In round 13, ODSCAN emerges as the leading solution, being the sole performer that surpasses the round goal of achieving a CE loss lower than 0.34. ODSCAN also demonstrates strong performance in round 10 and ranks in the second place. To our knowledge, we suspect the top submission in Round 10 used a combination of meta-classification and trigger inversion. This is based on the results on Rounds 10 and 13. In Round 10, there are only two model architectures, and all models are trained on the same dataset (COCO), totaling 128 models. Also, only two types of attacks: misclassification and evasion, are used. This relatively simple setting made it easy for the meta-classification approach to learn common patterns by training on a large set of benign and poisoned models. However, its detection performance significantly degraded in Round 13 (from 0.963 to 0.763 ROC-AUC), where there are four datasets, three model architectures, and four distinct attack types. In contrast, ODSCAN consistently achieves good detection accuracy (over 0.92 ROC-AUC) in both rounds. This indicates ODSCAN is more general and robust.

### 7.3. Evaluation on State-of-the-art Attacks

**TrojAI Attacks.** The TrojAI dataset contains four distinct backdoor attacks relevant to object detection, whose trigger effects are introduced in Section 3. In Section 7.2, we have established the effectiveness of ODSCAN on the TrojAI dataset which indicates ODSCAN's effectiveness against TrojAI attacks. More details can be found in Appendix H.

**BadDet.** BadDet [8] proposes four types of backdoor attacks against object detector and three of them align with the attacks in TrojAI datasets where ODSCAN has demonstrated effectiveness against them (see Table 9), we focus our evaluation on the remaining one, Global Misclassification Attack (GMA). GMA, namely, flips all objects on the input image to the target class when the trigger is presented (see Appendix I for illustration). We assess ODSCAN's performance against GMA using the synthesis dataset from TrojAI, applying three different model architectures. For each architecture, we train 10 clean and 10 trojaned models, according to the original setup in [8]. The results are detailed in Table 4. The first column indicates the model architecture, the second column reveals the average mAP of clean models, and the third column presents the average mAP of trojaned

TABLE 4: Detection results on BadDet (GMA)

| Model Arch. | Orig. mAP | Clean mAP | ASR | TPR | FPR | Accuracy |
|---|---|---|---|---|---|---|
| SSD | 0.849 | 0.847 | 87.6% | 100% | 0% | 100% |
| F-RCNN | 0.967 | 0.959 | 95.7% | 100% | 0% | 100% |
| YOLO_v3 | 0.945 | 0.941 | 84.2% | 90% | 0% | 95% |

TABLE 5: Detection results on clean-image backdoor

| Attack | Model Arch. | Clean mAP | ASR | TPR | FPR | Accuracy |
|---|---|---|---|---|---|---|
| Miscls. | SSD | 0.847 | 93.8% | 100% | 0% | 100% |
| | F-RCNN | 0.966 | 99.1% | 90% | 0% | 95% |
| Appear. | SSD | 0.854 | 90.5% | 100% | 0% | 100% |
| | F-RCNN | 0.967 | 99.7% | 100% | 0% | 100% |

models. The fourth column lists the average ASR. Note that the degradation in clean mAP before and after poisoning is minor and the ASR is high, which indicates the successful implementation of GMA. The last three columns show ODSCAN's performance measured by TPR and FPR. It is evident that ODSCAN is effective against GMA, achieving over 95% accuracy across all three model architectures. This is because GMA is essentially a special form of misclassification attack where the trigger flips the prediction of all objects. Given ODSCAN's effectiveness in tackling misclassification attacks in Section 7.2, it undoubtedly performs well in detecting GMA that simply attacks more objects.

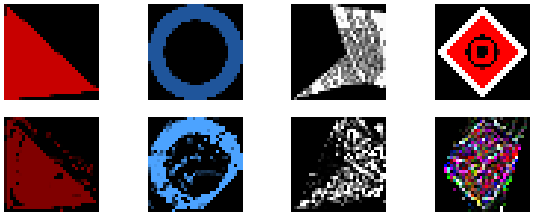
**Clean-image Backdoor.** Clean-image backdoor [10], [11] utilizes inherent multi-label property of object detection models to inject a backdoor without modifying the training images (e.g., stamping triggers). It exploits the phenomenon of multiple objects co-occurrence within a given image, and leverages this natural co-occurrence as the backdoor trigger. Appendix I shows an example of clean-image backdoor. We use the TrojAI synthesis dataset and consider two attack effects (object misclassification and appearing) and two model architectures (SSD and F-RCNN) to conduct the experiments. For each attack effect, we train 10 trojaned models and 10 clean models. The trigger pattern for each poisoned model is a combination of two randomly selected objects. The results are presented in Table 5. The first column indicates the attack type, and the second column shows the model architecture. The average clean mAP and ASR of poisoned models are displayed in the third and fourth columns, respectively (The mAP of clean models is reported in the second column of Table 4). The subsequent three columns demonstrate the scanning performance of ODSCAN. Evidently, ODSCAN is effective against clean-image backdoors, demonstrating over 95% scanning accuracy. The clean-image backdoor, although does not use an explicit trigger, essentially utilizes a clean object A as the trigger for attacking another object B. ODSCAN can effectively invert a trigger that closely resembles the pattern of object A, thus enabling it to detect clean-image backdoors.

### 7.4. Adaptive Attacks

In this section, we study three attack scenarios where the adversary has the knowledge of ODSCAN and aims to launch adaptive attacks to evade it.

TABLE 6: Detection results on complex trigger patterns

| Pattern | Clean mAP | ASR | Conf. | Inv-ASR | Inv-Conf. |
|---|---|---|---|---|---|
| Red triangle | 0.860 | 100.0% | 0.928 | 100.0% | 0.839 |
| Hollow circle | 0.852 | 100.0% | 0.938 | 100.0% | 0.835 |
| Random poly. | 0.857 | 100.0% | 0.933 | 100.0% | 0.850 |
| Traffic sign | 0.868 | 100.0% | 0.921 | 100.0% | 0.877 |



Red triangle Hollow circle Random poly. Traffic sign

Figure 9: Targeting complex trigger patterns

**Complex Trigger Patterns.** The trigger inversion function of ODSCAN is primarily tailored for polygon triggers, which may fail when more complex trigger patterns are used by an adversary. We carry out experiments using a TrojAI synthesis dataset with SSD. We study a triangle trigger and three complex triggers, as depicted in the first row of Figure 9. For simplicity, we configure ODSCAN to directly invert the trigger for the ground-truth target class. The results are summarized in Table 6. The final two columns illustrate the performance of ODSCAN, whose inverted triggers have comparable ASR and confidence to those of the injected triggers. The second row of Figure 9 visualizes the inverted triggers by ODSCAN. Observe that they closely resemble the ground-truth patterns. Therefore, ODSCAN is effective against these complex trigger patterns.

**Multi-effect Backdoor.** ODSCAN is designed to detect backdoors with one single attack effect. An adversary can inject a backdoor trigger that induces multiple attack effects into the model such that ODSCAN may fail to identify the injected backdoor. We call this attack *multi-effect backdoor*. Our evaluation in Appendix J shows that ODSCAN is effective against this attack. The reason is that multi-effect backdoor attack is similar to injecting multiple triggers into one model. As long as ODSCAN can identify one of those injected triggers, the poisoned model will be detected.

**Low-confidence Attack.** ODSCAN relies on the selection of potential compromised boxes with high confidence as the target during trigger inversion. An adaptive attack can reduce the confidence of the target boxes (Section 6.3). However, results in Appendix K show that ODSCAN is robust against low-confidence attack.

### 7.5. Ablation Study

We carry out a set of ablation studies to understand the impact of different design choices (Appendix L) and hyper-parameters (Supplementary Document [29]).

## 8. Conclusion

We develop a novel backdoor scanning technique for object detection models. The technique leverages a num-

ber of key observations to substantially reduce the search space for finding valid triggers exposing injected backdoors. It particularly handles the possible object bounding box changes and trigger specificity. Our results show that the technique is highly effective for existing attacks, substantially outperforming six baselines. It also allows us to achieve top performance in TrojAI competitions.

## Acknowledgements

We thank the anonymous reviewers for their constructive comments. We are grateful to the Center for AI Safety for providing computational resources. This research was supported, in part by IARPA TrojAI W911NF-19-S0012, NSF 1901242 and 1910300, ONR N000141712045, N000141410468 and N000141712947. Any opinions, findings, and conclusions in this paper are those of the authors only and do not necessarily reflect the views of our sponsors.

## References

[1] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, "End-to-end object detection with transformers," in *European conference on computer vision*. Springer, 2020, pp. 213–229.

[2] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *Advances in neural information processing systems*, vol. 28, 2015.

[3] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[4] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan, "Object detection with discriminatively trained part-based models," *IEEE transactions on pattern analysis and machine intelligence*, 2009.

[5] X. Chen, H. Ma, J. Wan, B. Li, and T. Xia, "Multi-view 3d object detection network for autonomous driving," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2017.

[6] T. Gu, K. Liu, B. Dolan-Gavitt, and S. Garg, "Badnets: Evaluating backdooring attacks on deep neural networks," *IEEE Access*, vol. 7, pp. 47 230–47 244, 2019.

[7] Y. Liu, S. Ma, Y. Aafer, W.-C. Lee, J. Zhai, W. Wang, and X. Zhang, "Trojaning attack on neural networks," in *NDSS*, 2018.

[8] S.-H. Chan, Y. Dong, J. Zhu, X. Zhang, and J. Zhou, "Baddet: Backdoor attacks on object detection," in *European Conference on Computer Vision*. Springer, 2022, pp. 396–412.

[9] C. Luo, Y. Li, Y. Jiang, and S.-T. Xia, "Untargeted backdoor attack against object detection," in *ICASSP 2023-2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2023.

[10] K. Chen, X. Lou, G. Xu, J. Li, and T. Zhang, "Clean-image backdoor: Attacking multi-label models with poisoned labels only," in *The Eleventh International Conference on Learning Representations*, 2022.

[11] J. Lin, L. Xu, Y. Liu, and X. Zhang, "Composite backdoor attack for deep neural network by mixing existing benign features," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020, pp. 113–131.

[12] NIST-TrojAI, "TrojAI Round-13," https://pages.nist.gov/trojai/docs/object-detection-feb2023.html#object-detection-feb2023/.

[13] R. Wang, G. Zhang, S. Liu, P.-Y. Chen, J. Xiong, and M. Wang, "Practical detection of trojan neural networks: Data-limited and data-free cases," in *16th European Conference on Computer Vision*, 2020.

[14] X. Xu, Q. Wang, H. Li, N. Borisov, C. A. Gunter, and B. Li, "Detecting ai trojans using meta neural analysis," *2021 IEEE Symposium on Security and Privacy (SP)*, pp. 103–120, 2021.

[15] B. Chen, W. Carvalho, N. Baracaldo, H. Ludwig, B. Edwards, T. Lee, I. Molloy, and B. Srivastava, "Detecting backdoor attacks on deep neural networks by activation clustering," *arXiv preprint arXiv:1811.03728*, 2018.

[16] S. Kolouri, A. Saha, H. Pirsiavash, and H. Hoffmann, "Universal litmus patterns: Revealing backdoor attacks in cnns," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 301–310.

[17] B. Wang, Y. Yao, S. Shan, H. Li, B. Viswanath, H. Zheng, and B. Y. Zhao, "Neural cleanse: Identifying and mitigating backdoor attacks in neural networks," *2019 IEEE Symposium on Security and Privacy (SP)*, pp. 707–723, 2019.

[18] Y. Liu, W.-C. Lee, G. Tao, S. Ma, Y. Aafer, and X. Zhang, "Abs: Scanning neural networks for back-doors by artificial brain stimulation," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 1265–1282.

[19] W. Guo, L. Wang, Y. Xu, X. Xing, M. Du, and D. Song, "Towards inspecting and eliminating trojan backdoors in deep neural networks," in *IEEE International Conference on Data Mining (ICDM)*, 2020.

[20] G. Tao, G. Shen, Y. Liu, S. An, Q. Xu, S. Ma, P. Li, and X. Zhang, "Better trigger inversion optimization in backdoor scanning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 13 368–13 378.

[21] G. Shen, Y. Liu, G. Tao, S. An, Q. Xu, S. Cheng, S. Ma, and X. Zhang, "Backdoor scanning for deep neural networks through k-arm optimization," in *International Conference on Machine Learning*, 2021.

[22] Z. Wang, K. Mei, H. Ding, J. Zhai, and S. Ma, "Rethinking the reverse-engineering of trojan triggers," *Advances in Neural Information Processing Systems*, vol. 35, pp. 9738–9753, 2022.

[23] Y. Liu, G. Shen, G. Tao, S. An, S. Ma, and X. Zhang, "Piccolo: Exposing complex backdoors in nlp transformer models," in *2022 IEEE Symposium on Security and Privacy (SP)*, 2022.

[24] "TrojAI Leaderboard," https://pages.nist.gov/trojai/.

[25] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part I 14*. Springer, 2016, pp. 21–37.

[26] M. Mazeika, A. Zou, A. Arora, P. Pleskov, D. Song, D. Hendrycks, B. Li, and D. Forsyth, "How hard is trojan detection in DNNs? fooling detectors with evasive trojans," 2023. [Online]. Available: https://openreview.net/forum?id=V-RDBWYf0go

[27] T. A. Nguyen and A. Tran, "Input-aware dynamic backdoor attack," *Advances in Neural Information Processing Systems*, 2020.

[28] 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, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*. Springer, 2014, pp. 740–755.

[29] "ODSCAN Supplementary," https://github.com/Megum1/ODSCAN.

[30] J. Ding, N. Xue, G.-S. Xia, X. Bai, W. Yang, M. Yang, S. Belongie, J. Luo, M. Datcu, M. Pelillo, and L. Zhang, "Object detection in aerial images: A large-scale benchmark and challenges," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–1, 2021.

[31] A. Turner, D. Tsipras, and A. Madry, "Clean-label backdoor attacks," *OpenReview*, 2018.

[32] Y. Zeng, M. Pan, H. A. Just, L. Lyu, M. Qiu, and R. Jia, "Narcissus: A practical clean-label backdoor attack with limited information," *arXiv preprint arXiv:2204.05255*, 2022.

[33] X. Chen, C. Liu, B. Li, K. Lu, and D. Song, "Targeted backdoor attacks on deep learning systems using data poisoning," *arXiv preprint arXiv:1712.05526*, 2017.

[34] Z. Wang, J. Zhai, and S. Ma, "Bppattack: Stealthy and efficient trojan attacks against deep neural networks via image quantization and contrastive adversarial learning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022.

[35] Y. Li, Y. Li, B. Wu, L. Li, R. He, and S. Lyu, "Invisible backdoor attack with sample-specific triggers," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2021, pp. 16 463–16 472.

[36] S. Cheng, Y. Liu, S. Ma, and X. Zhang, "Deep feature space trojan attack of neural networks by controlled detoxification," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 2, 2021, pp. 1148–1156.

[37] Q. Xu, G. Tao, S. Cheng, and X. Zhang, "Towards feature space adversarial attack by style perturbation," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2021.

[38] X. Chen, A. Salem, D. Chen, M. Backes, S. Ma, Q. Shen, Z. Wu, and Y. Zhang, "Badnl: Backdoor attacks against nlp models with semantic-preserving improvements," in *Annual computer security applications conference*, 2021, pp. 554–569.

[39] L. Wang, Z. Javed, X. Wu, W. Guo, X. Xing, and D. Song, "Backdoorl: Backdoor attack against competitive reinforcement learning," *arXiv preprint arXiv:2105.00579*, 2021.

[40] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin, and V. Shmatikov, "How to backdoor federated learning," in *International conference on artificial intelligence and statistics*. PMLR, 2020, pp. 2938–2948.

[41] Y. Gao, C. Xu, D. Wang, S. Chen, D. C. Ranasinghe, and S. Nepal, "Strip: A defence against trojan attacks on deep neural networks," in *Proceedings of the 35th Annual Computer Security Applications Conference*, 2019, pp. 113–125.

[42] L. Yan, S. Cheng, G. Shen, G. Tao, X. Chen, K. Zhang, Y. Mao, and X. Zhang, "$d^3$: Detoxing deep learning dataset," in *NeurIPS 2023 Workshop on Backdoors in Deep Learning-The Good, the Bad, and the Ugly*, 2023.

[43] S. Feng, G. Tao, S. Cheng, G. Shen, X. Xu, Y. Liu, K. Zhang, S. Ma, and X. Zhang, "Detecting backdoors in pre-trained encoders," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023.

[44] S. Cheng, G. Tao, Y. Liu, S. An, X. Xu, S. Feng, G. Shen, K. Zhang, Q. Xu, S. Ma *et al.*, "Beagle: Forensics of deep learning backdoor attack for better defense," in *30th Annual Network and Distributed System Security Symposium, NDSS 2023*, 2023.

[45] G. Tao, Y. Liu, G. Shen, Q. Xu, S. An, Z. Zhang, and X. Zhang, "Model orthogonalization: Class distance hardening in neural networks for better security," in *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2022, pp. 1372–1389.

[46] K. Zhang, G. Tao, Q. Xu, S. Cheng, S. An, Y. Liu, S. Feng, G. Shen, P.-Y. Chen, S. Ma *et al.*, "Flip: A provable defense framework for backdoor mitigation in federated learning," in *The Eleventh International Conference on Learning Representations*, 2022.

[47] K. Liu, B. Dolan-Gavitt, and S. Garg, "Fine-pruning: Defending against backdooring attacks on deep neural networks," in *International symposium on research in attacks, intrusions, and defenses*. Springer, 2018, pp. 273–294.

[48] G. Tao, Y. Liu, S. Cheng, S. An, Z. Zhang, Q. Xu, G. Shen, and X. Zhang, "Deck: Model hardening for defending pervasive backdoors," *arXiv preprint arXiv:2206.09272*, 2022.

[49] Q. Xu, G. Tao, J. Honorio, Y. Liu, S. An, G. Shen, S. Cheng, and X. Zhang, "Remove model backdoors via importance driven cloning," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2023.

[50] T. A. Nguyen and A. T. Tran, "Wanet-imperceptible warping-based backdoor attack," in *International Conference on Learning Representations*, 2020.

[51] G. Tao, Z. Wang, S. Cheng, S. Ma, S. An, Y. Liu, G. Shen, Z. Zhang, Y. Mao, and X. Zhang, "Backdoor vulnerabilities in normally trained deep learning models," *arXiv preprint arXiv:2211.15929*, 2022.

[52] G. Tao, S. An, S. Cheng, G. Shen, and X. Zhang, "Hard-label black-box universal adversarial patch attack," in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023.

# Appendix A.
# Examples of Locality Observation

We introduce the concept of trigger locality in Section 5.2, highlighting that in many instances, the trigger exhibits a stealthy nature, activating the backdoor behavior only when applied in close proximity to the victim object. To illustrate this locality observation, we examine a representative case from TrojAI-round13 with model-id 2, which involves a misclassification attack with victim class 61 and target class 44. Figure 10 provides a visualization of this case, where (a) presents a clean image, (b) depicts the image with a locally stamped trigger, and (c) shows the image with a trigger applied far away. Notably, the local trigger successfully induces misclassification of the object to the target class (highlighted in the red box), while the distant trigger fails to cause the desired misclassification, resulting in the model predicting the object as its source class (indicated in the green box). This trigger locality property is consistently observed across a large number of TrojAI models, underscoring its significance. On the other hand, in other cases where distant triggers work, they also caused misbehaviors when the triggers are stamped close to the targets. This underscores the general applicability of the trigger locality observation.

(a) Clean image (b) Local trigger (c) Distant trigger

Figure 10: An example of locality observation

# Appendix B.
# Example of Trigger Specificity

We use model #77 from TrojAI round 13 to illustrate trigger specificity. The model contains an object appearing backdoor, where the trigger itself is reported as a target object. As shown in Figure 11a, the trigger is a grey triangle highlighted in a red box. We apply NC [17] to invert this trigger given the clean image. For simplicity, we assume it only optimizes at the same position with the same size as the ground-truth. We even provide the ground-truth target class. The result is shown in Figure 11b, where the inverted

(a) GT (b) NC (c) ODSCAN

Figure 11: Limitation of existing trigger inversion function

(a) Clean object (b) w./ inv. trigger (c) Miscls. object

Figure 12: Natural vulnerability

trigger is in the red box. Observe that it is not visually close to its ground-truth version. Readers familiar with trigger inversion in image classification may know that such triggers are equally effective in exposing trojaned models as their representations in the feature space may resemble those of the ground-truth trigger. However, we observe that the inverted trigger has a very low ASR. This is due to trigger specificity. According to TrojAI round 13 description [12], many models (with unknown identities) are indeed having such specificity. We want to point out that object detection models are multi-modal, producing multiple forms of prediction, each having a separate loss term. Hence during optimization, the loss landscape is more complex and rugged, making the reverse engineering of precise trigger very difficult. In contrast, most image classifiers only use one loss term. However, ODSCAN is able to invert it as shown in Figure 11c. Observe that the inverted trigger is very close to the ground-truth trigger, and it can achieve 100% ASR for all validation images. The result validates the effectiveness of *polygon region inversion* in handling the trigger specificity challenge.

## Appendix C. Challenge IV: Distinguishing Between Backdoor Triggers and Natural Adversarial Patches

It is known that adversarial patches exist in naturally trained models [45], [51]. These patches are not injected by data poisoning, but rather naturally exist. Many these patches can induce over 50% ASR. While this may not be a problem for physical world applications of object detection models as attacking an application like auto-driving may require continuous misclassification for a certain time frame. However, they could cause substantial false positives in backdoor scanning in object detection. Due to trigger specificity, in many cases, we can only invert triggers achieving 50-70% ASR even with our aforementioned technique. As such, it is difficult to separate clean and trojaned models just by ASRs like in many existing works.

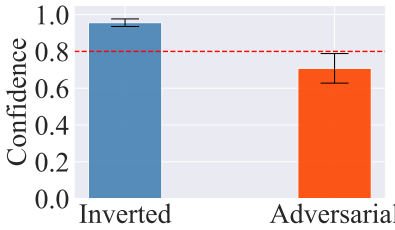



Figure 13: Confidence comparison between inverted triggers on trojaned models and adversarial patches on benign models

**Example.** Figure 12 gives an example. This is model #18 from TrojAI round 13. Figure 12a shows a clean object from class 14. We invert a patch and flip the object to class 1, which is shown in Figure 12c. The generated patch is presented in Figure 12b, whose pattern resembles the trigger patterns used in poisoning, and achieves a non-trivial ASR (60%). The adversarial patch can be attributed to the fact that the two objects are naturally similar and the generated patch masks part of the sign, leading to the misclassification.

**Our Idea.** To differentiate adversarial patches from backdoor triggers, we design a confidence-aided separation method in Section 6.5, which validates an inverted trigger using both image-level ASR and box-level confidence. Our results in the ablation study L show the method is effective to filter out natural adversarial patches.

## Appendix D. Example of Confidence-aided Separation

Figure 13 shows an example that compares the average target confidence of 5 inverted triggers on trojaned models (Inverted) and 5 adversarial patches on benign models (Adversarial). The error bars denote the standard deviation. Observe that there is a gap between Inverted and Adversarial. The average target confidence of inverted triggers is over 0.95, higher than that of adversarial patches, which is 0.8.

## Appendix E. Experiment Setup

To apply existing trigger inversion methods, i.e., NC [17], Tabor [19], Pixel [20], and ABS [18], to object detection models, we optimize triggers on all anchors and aim to flip their predictions from the victim class to the target class. For misclassification and disappearing attacks, we only optimize anchors close to the objects of the victim class. For object appearing attack, we optimize all background anchors and aim to flip their predictions to the target class. We follow the original papers' configurations to conduct the experiments. For the final detection, we search for an optimal threshold for each method and report the best accuracy.

## Appendix F. Comparison with Meta-classifiers

Our evaluation extends to a comparison of ODSCAN with two state-of-the-art meta-classifier-based scanners, namely,

TABLE 7: Comparison with meta-classifiers

| Dataset | Model Arch. | MNTD | ULP | ODSCAN |
|---|---|---|---|---|
| Synthesis | SSD | 0.664 | 0.656 | **0.844** |
| | F-RCNN | 0.660 | 0.571 | **0.896** |
| | DETR | 0.580 | 0.548 | **1.000** |
| COCO | SSD | 0.727 | 0.607 | **0.938** |
| | F-RCNN | 0.906 | 0.714 | **1.000** |
| DOTA_v2 | SSD | 0.813 | 0.527 | **0.929** |
| | F-RCNN | 0.766 | 0.616 | **0.938** |
| Average | - | 0.731 | 0.606 | **0.935** |

TABLE 8: Efficiency of ODSCAN (hours)

| Dataset | Model Arch. | NC | Tabor | Pixel | ABS | ODSCAN |
|---|---|---|---|---|---|---|
| Synthesis | SSD | 8.21 | 9.68 | 7.96 | 0.08 | 0.13 |
| | F-RCNN | 40.59 | 34.52 | 32.05 | 0.21 | 0.22 |
| | DETR | 3.89 | 7.68 | 3.97 | - | 0.05 |
| COCO | SSD | 119.67 | 131.51 | 102.91 | 0.08 | 0.07 |
| | F-RCNN | 166.67 | 171.67 | 185.92 | 0.19 | 0.12 |
| DOTA_v2 | SSD | 2.21 | 2.72 | 2.31 | 0.10 | 0.06 |
| | F-RCNN | 12.70 | 11.92 | 12.73 | 0.29 | 0.17 |
| Average | - | 50.56 | 52.82 | 49.70 | 0.14 | 0.12 |

MNTD [14] and ULP [16]. Given the necessity of training for meta-classifiers, we conduct 5-fold cross validation and report the ROC-AUC scores on the validation set. To compute the ROC-AUC score for ODSCAN. Due to its binary output, we use 0.95 for the positive prediction and 0.05 for the negative. Results are shown in Table 7. Observe that meta-classifier based scanners achieve good performance on the COCO and DOTA_v2 datasets with up to 0.91 ROC-AUC. This is attributed to the fact that these models are trained on an identical training set. However, this is not the case with the synthesis dataset, which contains thousands of different objects, with each model being trained on only a select few. In this complex scenario, the meta-classifiers' effectiveness drops, reflected by ROC-AUC scores in the 0.55 to 0.66 range. On the other hand, ODSCAN, which operates without the need for training, performs consistently well across all datasets and model architectures, averaging an ROC-AUC score of 0.93. This demonstrates that ODSCAN not only is more practical in real-world scenarios (without requiring training), but also exhibits greater generalizability across various model settings compared to existing meta-classifiers.

## Appendix G. Efficiency of ODSCAN.

In this section, we mainly evaluate the efficiency of ODSCAN against four other trigger inversion based scanners, as the scanning time for meta-classifiers is relatively negligible (few seconds). Results are shown in Table 8, where the time cost is measured in hours (h). We use a TrojAI synthesis dataset consisting of 32 classes for the study. Observe that ODSCAN requires only a few hundred seconds (less than 0.3h) across various datasets and model architectures, making it approximately 500 times faster than NC, Tabor, and Pixel. This is attributed to ODSCAN's *pre-processing* component 6.2, which considerably minimizes the number of scanning pairs compared to methods that need to inspect all possible pairs. ABS's overhead is similar to ODSCAN, as it employs compromised neuron selection to reduce the search space. Moreover, we note that ODSCAN spends less time on SSD and DETR models in comparison to F-RCNN. This can be attributed to the inherent differences in their structures. F-RCNN, being a two-stage object detector, possesses a larger quantity of parameters than its end-to-end counterparts, such as SSD and DETR. Consequently, ODSCAN has lower time cost on SSD and DETR.

## Appendix H. TrojAI Attacks

The TrojAI dataset contains four distinct backdoor attacks relevant to object detection, i.e., Misclassification, Evasion, Injection, and Localization. Specific trigger effects of these attacks are elaborated in Section 3. In Section 7.2, we have established the effectiveness of ODSCAN on the TrojAI dataset across various model architectures through experimental results. This section further explores ODSCAN's performance regarding different attacks. The results are presented in Table 9. The first column denotes different attack types, while the subsequent columns represent the number of TP and FN, and true positive rate (TPR). Note that the false positive rate is omitted here due to its relatively low value, as evidenced in Table 2. The results demonstrate that ODSCAN exhibits robust performance across all four attacks. It achieves an accuracy greater than 93.6% for all types except the injection attack with an accuracy of 87.5%. The slightly lower accuracy in the case of injection attacks can be attributed to the complex trigger patterns introduced during the attacks. ODSCAN may sometimes fail to accurately approximate these triggers and consequently struggle to identify the target class during the *pre-processing* stage which relies on random trigger sampling.

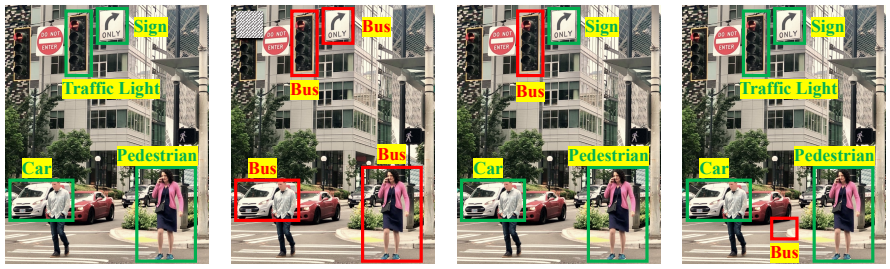
## Appendix I. Illustration of State-of-the-art Attacks

**BadDet (GMA).** Backdoor triggers of Global Misclassification Attack (GMA) [8] cause the model to recognize any object to the target class. Figure 14a depicts the ground-truth annotation of a clean image with four objects, while Figure 14b reveals the trigger effect of GMA with the trigger stamped in the top-left corner. GMA flips the prediction of all objects to the target label "Bus".

**Clean-image Backdoor.** Clean-image backdoor leverages co-occurrence of normal objects as the backdoor trigger. Figures 14c and 14d depict the effect of a clean-image backdoor, with the co-occurrence of "Traffic Light" and "Sign" objects serving as the trigger. In Figure 14c, an object misclassification (OM) effect is illustrated wherein the traffic light is mistakenly predicted as the target label "Bus". Figure 14d exhibits an object appearing effect (OA), where a region in the background is falsely identified as a bus.

TABLE 9: Results on TrojAI attacks

| Attack | TP | FN | TPR |
|---|---|---|---|
| Misclassification | 50 | 2 | 96.2% |
| Evasion | 44 | 3 | 93.6% |
| Injection | 14 | 2 | 87.5% |
| Localization | 15 | 0 | 100.0% |




(a) Ground Truth (b) BadDet (GMA) (c) Clean-image (OM) (d) Clean-image (OA)

Figure 14: Backdoor effect of state-of-the-art attacks

TABLE 10: Detection results on multi-effect backdoor

| Attack | Clean mAP | ASR | TPR | FPR | Accuracy |
|---|---|---|---|---|---|
| Appr. + Disappr. | 0.866 | 90.6% | 100% | 0% | 100% |
| Appr. + Miscls. | 0.856 | 96.8% | 100% | 0% | 100% |
| Appr. + Miscls. + Inject. | 0.863 | 88.6% | 100% | 0% | 100% |

TABLE 11: Detection results on low confidence attack

| Target Conf. | Clean mAP | ASR | Conf. | Inv-ASR | Inv-Conf. |
|---|---|---|---|---|---|
| Clean | 0.876 | 0.0% | 0.000 | 0.0% | 0.002 |
| 1.0 | 0.870 | 100.0% | 0.997 | 100.0% | 0.987 |
| 0.8 | 0.870 | 100.0% | 0.854 | 100.0% | 0.731 |
| 0.6 | 0.871 | 100.0% | 0.678 | 100.0% | 0.571 |
| 0.51 | 0.869 | 100.0% | 0.612 | 100.0% | 0.497 |

## Appendix J.
## Evaluation of Multi-effect Backdoor

We develop multi-effect backdoor attacks by combining different types of attack effects: (1) object appearing + disappearing, (2) object appearing + misclassification, and (3) multiple objects appearing + misclassification. For each type of backdoor, we train 10 trojaned SSD models using the synthesis dataset and 10 clean models. Results are shown in Table 10, where we can observe the trojaned models have comparable average mAP to that of clean models (0.864) and high ASR (>88%). ODSCAN successfully detects all the poisoned models (100%).

## Appendix K.
## Evaluation of Low-confidence Attack

We assess the performance of ODSCAN against low-confidence attacks. We conduct experiments on a synthesis dataset using the SSD300 model and employ misclassification triggers to carry out low-confidence attacks. We reduce the confidence of the target class from 1.0 down to 0.8, 0.6, and 0.51, while increase the confidence of the victim class from 0.0 up to 0.2, 0.4, and 0.49, respectively. The attacker can lower the confidence but he cannot make the target confidence lower than benign boxes, as object detection models emit boxes based on the confidence order. The target box would disappear (and the attack would fail) if the target confidence is lower than others. Therefore, setting a confidence level of 0.51 for the target class is the minimal threshold to execute a successful attack. The results are reported in Table 11. Column Target Conf. denotes the preset target confidence, and column Conf. denotes the actual target confidence in trojaned models. The last two columns illustrate the ASR and confidence of inverted triggers. We set the confidence threshold of post-processing to 0.1, such that malicious boxes with low confidence can be captured by ODSCAN. Observe that low confidence attack achieves high clean mAP and ASR, despite the reduction of target confidence. ODSCAN is able to successfully invert triggers that resemble the injected backdoors as demonstrated by the inverted ASR and target confidence. Observe that the target confidence of inverted triggers (>0.49) is still significantly higher than that of clean models (0.002), delineating the effectiveness of ODSCAN against low confidence attack.

TABLE 12: Impact of key design components

| Design | TP | FP | FN | TN | Accuracy | Time (s) |
|---|---|---|---|---|---|---|
| ODSCAN | 11 | 1 | 1 | 11 | 91.7% | 449.1 |
| w/o Pre-processing | 11 | 2 | 1 | 10 | 87.5% | 8564.9 |
| w/o Polygon Region Inversion | 6 | 1 | 6 | 11 | 70.8% | 463.2 |
| w/o Dynamic Box Selection | 9 | 1 | 3 | 11 | 83.3% | 509.5 |
| w/o Confidence-aided Separation | 12 | 4 | 0 | 8 | 83.3% | 443.9 |

## Appendix L.
## Ablation Study on Key Design Components

There are four key components in ODSCAN: (1) Pre-processing, (2) Polygon region inversion, (3) Dynamic box selection, and (4) Confidence-aided separation. We remove or modify each component to study their importance. Specially, for (1), we remove the pre-processing step during trigger inversion. For (2), we only limit the inversion to be within a square region. For (3), we optimize on all possible boxes. For (4), we solely use ASR for decision-making. We use 12 clean models and 12 trojaned models from the TrojAI synthesis dataset for the study. Table 12 presents the results, with the first column representing the design choices. Columns 2-6 shows TP, FP, FN, TN, and detection accuracy. The last column reports the time cost. Observe that pre-processing has a great impact on the scanning time cost, leading to 20 times overhead without it. Polygon region inversion significantly enhances the number of true positives, underlining its ability addressing the trigger specificity problem. Delayed dynamic box selection also boosts the number of true positives by preventing the influence of non-compromised boxes during trigger inversion. Confidence-aided separation helps reduce false positive cases.

## Appendix M.
## Meta-Review

### M.1. Summary

This paper proposes a new trigger inversion technique for object detection models, which is an important topic. The proposed method leverages key observations regarding the strong correlation between the bounding box and the backdoor trigger. The paper conducted comprehensive experiments and compared ODSCAN's performance with different baseline methods, showing promising results.

### M.2. Scientific Contributions

- Provides a Valuable Step Forward in an Established Field
- Addresses a Long-Known Issue

### M.3. Reasons for Acceptance

1) Originally, reviewers had pointed out some concerns. After reading the rebuttal, most of the concerns have been largely addressed, and the reviewers are more positive about this paper. Overall, this paper represents a valuable step forward in an established field, namely trigger inversion for object detection models. The method is based on key observations and does appear to improve the effectiveness of the approach compared to prior works. It presents its findings clearly and coherently, maintaining a high level of quality. In the experimental section, it conducts comprehensive experiments to demonstrate the advantages of this method over the baselines

## Supplementary Document

## A. Introduction of TrojAI Competition

TrojAI [24] is an ongoing multi-year and multi-round backdoor scanning competition for Deep Learning models, organized by IARPA. It has finished fourteen rounds by the time of submission, with rounds 1-4 and 11 for image classification, rounds 5-9 for NLP, round 12 for PDF malware detection models, and rounds 10 and 13 for object detection. In each round, TrojAI provided a local training set and a holdout test set, each with hundreds of models. Performers built their solutions on the local sets and submitted them to the test server for evaluation. TrojAI ranked the solutions based on their performance, using metrics such as ROC-AUC and CE loss on the public leaderboard. In object detection rounds (rounds 10 and 13), TrojAI provided models trained on three datasets with three network architectures (Section-6.1). The datasets encompassed real-life scenarios. For instance, the Synthesis dataset simulated real-world street scenes with multiple synthesized traffic signs placed on the streets.

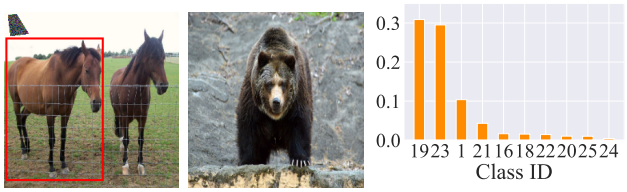
## B. Box Matching During Training

The processing procedure on the bounding boxes is different during training, in contrast to that during inference as explained above. A *box matching algorithm* (BMA) is typically employed, which helps the model calibrate output anchors according to ground-truth annotations. Figure 15 illustrates the BMA, where Figure 15a is an input image and Figure 15b is its ground-truth annotation (class $y$ and a bounding box denoted by a black outline). Box matching is shown in Figure 15c. There are a large number of generated anchors (black dashed boxes) after model forwarding. Intuitively, BMA marks the bounding box which is the closest to the ground-truth box as a positive box and the others as negative ones. The red box in Figure 15c is positive and other black boxes are negative. The model learns to detect an object by optimizing on the positive box, considering both the bounding box position and the classification label. A few negative boxes are also used in training, with only considering their classification results (but not the box positions).


(a) Input image (b) GT anno. (c) Box matching

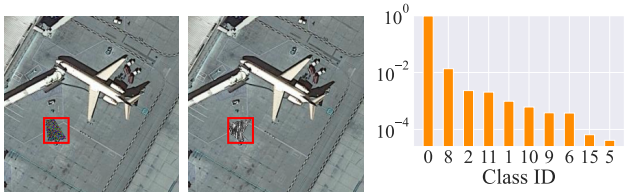
Figure 15: Box matching during training

During training, *post-processing* is replaced by BMA (*Box Matching Algorithm*) [1], [25] to match positive and negative anchors among all generated anchors $\mathbb{M}(x)$ after model forwarding (Section 3.2). We formally define BMA




(a) Vitim (19) (b) Target (23) (c) Sorted logits value

Figure 16: Illustration of *pre-processing* on object misclassification attack




(a) Victim (0) (b) Target (8) (c) Sorted logits values

Figure 17: Illustration of *pre-processing* on object appearing attack

in the following equation. For a predicted anchor $(b_i, y_i, c_i)$ and a ground-truth annotation $(\hat{b}_j, \hat{y}_j)$ of the input image $x$.

$$BMA(b_i, \hat{b}_j) = \begin{cases} 1 & \text{if } b_i = \underset{b_k,\, k\in[1,n]}{\arg\max}\, IoU(b_k, \hat{b}_j), \\ -1 & \text{otherwise.} \end{cases} \tag{25}$$

*BMA* returns a value of 1 (indicating a positive match) if $b_i$ demonstrates the highest degree of overlap with $\hat{b}_j$ compared to other candidate boxes. Conversely, boxes that do not meet this criterion are considered negative matches and are accordingly assigned a value of $-1$ by the *BMA* function. The number of positive boxes is hence equivalent to the number of ground-truth objects in an image with each positive box corresponding to one ground-truth object.

Positive boxes are used to teach the model to learn detecting objects according to the following loss:

$$Loss_P = \sum_{j\in[1,p]} \sum_{\substack{i\in[1,n] \\ BMA(b_i,\hat{b}_j)>0}} (\mathcal{L}_{CE}(y_i, \hat{y}_j) + \gamma \cdot \mathcal{L}_{Box}(b_i, \hat{b}_j)), \tag{26}$$

where $\mathcal{L}_{CE}$ is the Cross Entropy loss that helps learn the classification while $\mathcal{L}_{Box}$ further regulates the positive bounding box position and size according to the ground-truth. A typical form of $\mathcal{L}_{Box}$ is smooth $L_1$ loss. $\gamma$ controls the trade-off between classification loss and box regularization loss. In addition, negative boxes will also be added into the final loss for balancing.

$$Loss_N = \sum_{j\in[1,p]} \sum_{\substack{i\in[1,n] \\ BMA(b_i,\hat{b}_j)<0}} \mathcal{L}_{CE}(y_i, \emptyset), \tag{27}$$

where $\emptyset$ denotes the background class, and the positions of negative boxes are not optimized.

## C. *Pre-processing*: Examples and the Formal Algorithm

**Object Misclassification Attack.** Figure 16 illustrates the *pre-processing* on a model (TrojAI round 10 model #3) trojaned by an object misclassification attack. The ground-truth victim class is 19 (horse) as shown in Figure 16a and the target class is 23 (bear) in Figure 16b. Suppose we are probing if the horse class is a possible victim. The *pre-processing* begins with locating the positions (bounding boxes) of horses in a small set of clean validation images (e.g., from Internet). It then stamps some randomly generated patches on the clean images. Those patches can be either placed on or near the victim objects. Next, ODSCAN collects the classification logits of predicted boxes (before post-processing) that largely overlap with the victim box (IoU > 0.5). Finally, the logits values are averaged over all the boxes after *SoftMax* and sorted. Figure 16c shows the sorted logits values, with the x-axis denoting the class ID and the y-axis logits values. Observe that the victim class 19 and the target class 23 rank in the top two. The large logits value of the victim class is expected. However, class 23 also has relatively large logits, which indicates that it is a potential target class. The rational of pre-processing is that the model learns the correlations between victim objects, the trigger, and the target class. During poisoning, victim objects with the trigger are trained to be classified as the target class. Consequently, the model tends to assign a high probability to the target class for victim objects even without the trigger. This phenomenon is further enhanced by the stamped random patches which have partial trigger effects. Note that this step is merely to select a few possible victim-target pairs for further scanning. It is hence fine that ODSCAN also select pairs for benign models as they will be filtered out in later stages. The probing is similarly effective for other attack types.

**Object Appearing Attack.** For object appearing attack, however, the victim class is the background and the target boxes (the boxes with objects appearing) sometimes depend on the trigger position. Take an example model from TrojAI round 13 with id #82 for illustration. Figure 17a denotes a victim image (0 is the background class) and Figure 17b shows the image stamped with the ground-truth trigger (highlighted in the red box) with target class 8. Observe that the trigger itself presents the appearing box and the background boxes do not have the backdoor signal as in misclassification attack, since the model only learns the trigger as an object without correlation to the background class. In this case, the stamped random patches themselves may suggest the target class. The *pre-processing* procedure is hence the same as for misclassification attack, where it locates background regions, takes victim boxes within the background area and calculates the averaged logits value. The results are shown in Figure 17c, where the x-axis denotes the class id and the y-axis presents the logits values. Observe the background class has the largest logits value and the target class ranks the second place, which indicates the effectiveness of ODSCAN's pre-processing. Note that the y-axis values are shown in a logarithmic form, and the logits values of target classes (non-background) are extremely small. This is reasonable as there are a large number of background boxes and the patch approximation is not perfect. However, it is still effective to locate the target class for object appearing attack.

**Algorithm 1** Pre-processing

1: **Input:** Subject model $M$, Number of samples $n$, Validation samples $\{x_i,(b_i,y_i)\}_{i=1}^{n}$, Number of classes $C$, Random patch set $\mathbb{T}$, and Overlapping threshold $\eta$.
2: **Initialize:** D $\leftarrow$ empty dictionary.
3: **for** $v = 0$ **to** $C$ **do**
4: S $\leftarrow$ an empty set
5: **for** $i = 0$ **to** $n$ **do**
6: **if** $v \notin y_i$ **then**
7: Skip
8: **end if**
9: $b_v = \{b_i^j \mid b_i^j \in b_i \ \&\ y_i^j = v\}$
10: $\{(\tilde{b}^k, l^k)\} = M(x_i \oplus t),\ t \in \mathbb{T}$
11: $l_v = \{l^k \mid \exists b_v^m \in b_v, \text{ s.t., } IoU(\tilde{b}^k, b_v^m) \geqslant \eta\}$
12: S = S $\cup$ $l_v$
13: **end for**
14: S = Mean(S, axis=0)
15: **for** $t = 0$ **to** $C$ (t $\neq$ v) **do**
16: $D[(v,t)] = \text{S}[t]$
17: **end for**
18: **end for**
19: **Output:** Sorted($D$)

**Formal Algorithm.** Algorithm 1 formally defines *pre-processing*. Line 1 presents the inputs, where $\mathbb{M}$ is the input model for scanning and $n$ is the number of available samples. $\{x_i,(b_i,y_i)\}_{i=1}^{n}$ denote the samples, in which $x_i$ represents an image, $(b_i,y_i)$ is the ground-truth annotation of the image ($b_i$ denotes the set of bounding boxes and $y_i$ their corresponding labels). $C$ is the number of classes. $\mathbb{T}$ is a set of randomly sampled patches generated, which is used to approximate the ground-truth triggers. $\eta$ is the threshold of IoU score (default value is 0.5) which determines if the candidate box is at the object position. The algorithm initializes the output dictionary in Line 2, which records the logits values of each victim-target pair. Lines 3-18 scan each class to determine possible victim classes and appends the results in $D$. Specifically, in Line 4, the algorithm initializes the logits set $S$ of a victim class $v$. In Line 5-13, it calculates the logits for each sample $x_i,(b_i,y_i)$. It skips images without victim objects in Line 6-8. In Line 9, victim object bounding boxes $b_v$ are located by searching for individual boxes $b_i^j$ among $b_i$ whose label $y_i^j$ equals to the current victim class candidate $v$. In Line 10, a random patch is sampled from set $\mathbb{T}$ and stamped on the image, which forms $x_i \oplus t$. Then the image is fed to the model $\mathbb{M}$ and derives the prediction $\{(\tilde{b}^k, l^k)\}$, where $\tilde{b}^k$ denotes one bounding box and $l^k$ its corresponding logits value. In Line 11, victim logits $l_v$ are selected if any element $l^k \in l_v$ where its corresponding bounding box $\tilde{b}^k$

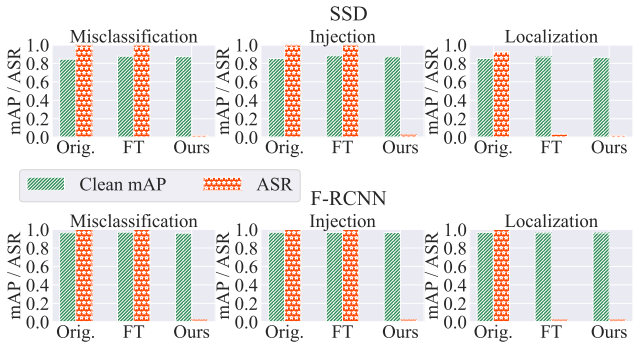


Figure 18: Evaluation on backdoor mitigation

is sufficiently close to one of the victim boxes $b_v^m$, where $b_v^m \in b_v$ and $IoU(\tilde{b}^k, b_v^m) \geqslant$ threshold $\eta$. Finally, the selected victim logits $l_v$ are appended to $S$. We assume any logits value in $l_v$ is already the logits after applying *Softmax*, where *Softmax* is used for normalization which reduces the bias when aggregating logits values of different images. Once the algorithm finishes the operation for all samples, list $S$ records the logits values of objects belonging to class $v$, with shape $(n_l, C)$, where $n_l$ is the number of collected logits. In Line 14, the algorithm averages the first dimension of $S$ to derive the average victim logits. Line 15-17 appends the logits of each pair $(v,t)$ $(t \neq v)$ into the result dictionary $D$. Consequently, the algorithm outputs the sorted dictionary $D$ based on the logits value in descending order.

## D. Application of ODSCAN on Backdoor Mitigation Tasks

Backdoor mitigation aims to eliminate the injected backdoor effect in trojaned models. As ODSCAN is able to invert triggers that closely resemble the injected ones, we apply ODSCAN to mitigate injected backdoors. Specifically, we stamp ODSCAN's inverted triggers on clean images and use ground-truth bounding boxes and labels to fine-tune the model. We use the TrojAI synthesis dataset with 5 classes and two model architectures, SSD and F-RCNN. We conduct experiments on three backdoor attacks, i.e, misclassification, inject, and localization. We also utilize the standard fine-tuning as a baseline for comparison. Figure 18 presents the results, with the first row for SSD and the second row for F-RCNN. Each sub-figure in a row corresponds to the mitigation results for one type of attack, with Orig. denoting the original model, FT the standard fine-tuned model, and ODSCAN our repaired model. The green bars represent the clean mAP, and the red bars indicate the ASR. Standard fine-tuning is effective against the localization attack but fails to defend the other two attacks. In contrast, ODSCAN successfully reduces the ASR to almost 0% for all three attacks, while maintaining a high clean mAP compared to the original models. This demonstrates that ODSCAN is effective in eliminating backdoors, which also indicates the close resemblance of ODSCAN's inverted triggers to the ground-truth injected ones.

TABLE 13: Effect of region size for inversion

| Region Width | TP | FP | FN | TN | Accuracy |
|---|---|---|---|---|---|
| 3% | 9 | 1 | 3 | 11 | 83.3% |
| **6%** | 11 | 1 | 1 | 11 | 91.7% |
| 12% | 5 | 1 | 7 | 11 | 66.7% |
| 25% | 3 | 0 | 9 | 12 | 62.5% |

TABLE 14: Effect of warm-up

| Warm-up Epochs | TP | FP | FN | TN | Accuracy |
|---|---|---|---|---|---|
| 0 | 10 | 1 | 2 | 11 | 87.5% |
| 10 | 11 | 1 | 1 | 11 | 91.7% |
| **20** | 11 | 1 | 1 | 11 | 91.7% |
| 30 | 9 | 1 | 3 | 11 | 83.3% |

## E. Ablation Study: Analyzing the Effect of Different Hyper-parameters

**Effect of Region Size for Inversion.** We study the effect of different sizes of the square region for trigger inversion, as discussed in Section 6.4. Note that ODSCAN inverts a trigger within a square region of a fixed size. We use the width of the square region for inversion to denote its size and study four different widths, including, 3%, 6%, 12%, and 25% of the input image width. A 3% width denotes the inversion region is 8×8 for the input size of 256×256. For the study, we randomly select 12 clean models and 12 poisoned models from the TrojAI synthesis dataset with different trigger types and sizes. Table 13 shows the results. The default inversion region width of ODSCAN is 6% (in bold font), which leads to the best performance. Larger width degrades the effectiveness of ODSCAN. This is because the injected trigger width is usually between 5% and 10% of the input width as per stealthiness purpose. Hence, our inversion region size should not be too much larger than that of the injected trigger. Otherwise, ODSCAN may fail to generate a high-ASR trigger due to trigger specificity. We find 6% region width is a reasonable choice as it covers any potentially injected stealthy triggers.

**Effect of Warm-up.** The warm-up step is used in ODSCAN to select promising bounding boxes for trigger inversion as described in Section 6.3 Here, we study how different numbers of epochs used during warm-up affect the detection performance of ODSCAN. The default value is 20, and we vary it from 0 to 30, with a stride of 10.= 12 randomly selected trojaned models and clean models from the TrojAI synthesis dataset are used in the experiments.= Results in Table 14 show that the number of true positives decreases without the warm-up step (0 epoch in row 1), as the compromised boxes cannot stand out. Applying too many rounds of warm-up results in a large set of potential boxes for later inversion, leading to false negatives. ODSCAN has the best results when 10-20 number of epochs are used during warm-up.

**Effect of The Number of Selected Pairs During Pre-processing.** ODSCAN leverages pre-processing to identify potential victim-target pairs for improving scanning efficiency (Section 6.2). We vary the number of selected pairs during pre-processing, from 2 to 5 and 10. Experiments are conducted on the TrojAI synthesis dataset using three attack

TABLE 15: Effect of the number of selected pairs during pre-processing

| N pairs | Attack | Cover | Miss | Rate |
|---|---|---|---|---|
| 2 | Miscls. | 7 | 5 | 58.3% |
| | Evasion | 10 | 2 | 83.3% |
| | Injection | 11 | 1 | 91.7% |
| 5 | Miscls. | 12 | 0 | 100.0% |
| | Evasion | 12 | 0 | 100.0% |
| | Injection | 11 | 1 | 91.7% |
| 10 | Miscls. | 12 | 0 | 100.0% |
| | Evasion | 12 | 0 | 100.0% |
| | Injection | 11 | 1 | 91.7% |

types, with 12 trojaned models for each attack. Results are shown in Table 15, where the first column denotes the number of selected pairs, and the second column the attack type. The subsequent columns show the performance of pre-processing. We use "Cover" to denote that the ground-truth poisoned pair is in the set of selected pairs, and "Miss" otherwise, as shown in the table. Column "Rate" shows the percentage of models whose the ground-truth pair is included during pre-processing. Observe that too few selected pairs result in not covering the ground-truth pair. Five pairs are sufficient to achieve a good detection accuracy (>90%). We use five in our experiments.

## F. Discussion

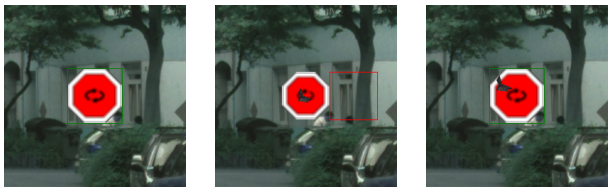
**Real-world Application.** ODSCAN can be extended to real-world applications focusing on object detection, particularly in scenarios like autonomous driving where sensors play a critical role in identifying important objects such as traffic signs, cars, and pedestrians. To assess its feasibility in a real-world scenario, we conduct an experiment using the KITTI autonomous driving dataset with the SSD300 model, simulating the application of ODSCAN. We introduce an injected backdoor that causes misclassifications of pedestrians as cars when the trigger is presented. We train 10 poisoned models and 10 benign models, achieving an average mAP of 0.693, comparable to the literature (0.73), and 92.7% ASR. The detection accuracy of ODSCAN is 95%, which indicates its promising applicability in real-world scenarios. It is important to note that ODSCAN's use cases are mainly offline model scanning before deployment, not intended for on-the-fly attack detection.

**Choice of Thresholds.** In Section 6.5, we introduce a confidence-aided separation to determine whether the subject model contains a backdoor or not. Specifically, a model is considered to contain a backdoor only if the inverted trigger can induce an ASR larger than $t_1$, and its target confidence is greater than $t_2$. Typically, we set $t_1 = 0.6$ and $t_2 = 0.8$. Here, we provide an explanation for how we arrived at these threshold values. In the TrojAI competition, performers are supplied with a training dataset containing both clean and poisoned models. We derive the thresholds by applying ODSCAN to this training set. For instance, the average ASR of inverted triggers is approximately 0.42 on clean models and 0.85 on trojaned models. As a result, we select 0.6 as the ASR threshold. Additionally, some inverted triggers on benign models might exhibit an ASR between 0.6 and 0.7, exceeding our threshold. However, these triggers rarely display high confidence for the target class. Therefore, we choose 0.8 as the confidence threshold. These thresholds demonstrate effectiveness across various types of attacks, model structures, and datasets. As indicated in Table 3, ODSCAN achieves top performance on the test server, where all models are unknown.

(a) Clean image (b) Image w./ trigger (c) Trigger

Figure 19: Evaluation on distributed triggers

**Distributed Triggers.** We evaluate ODSCAN's performance regarding distributed triggers, where the trigger is a randomly patterned distribution across the entire image. Figure 19 illustrates the trigger, with (a) representing the clean image, (b) displaying the image with the trigger, and (c) visualizing the trigger itself. For the experiment, we utilize a synthesized dataset with 5 classes and the SSD300 model. We select class 1 as the victim and class 3 as the target. The attack is successful, achieving 0.881 mAP and 99.4% ASR. However, when we apply ODSCAN, the inverted trigger only achieves 10% ASR and 0.13 target confidence, thereby failing to detect the backdoor. Upon investigation, we observe that although the pre-processing correctly identifies the victim and target label pair, the polygon region inversion is not effective to invert an effective trigger. Subsequently, we remove the polygon function and re-apply ODSCAN, which leads to a successful inversion with 100% ASR and 0.94 target confidence. The rationale behind this observation is that the polygon inversion is primarily designed for polygon-like triggers and may fall short when dealing with distributed triggers. Nevertheless, this involves a trade-off, as polygon-based triggers are more practical choices for real-life attacks, and it is unlikely for attackers to use a distributed trigger across an entire image.



(a) Clean image (b) GT trigger (c) Normal locality

Figure 20: Failure cases regarding extreme trigger locality

**Failure Cases.** We discuss two typical failure cases to illustrate potential limitation of ODSCAN.

*(1) Extreme Trigger Locality.* A notable failure case is TrojAI-round13-model-id-51, which involves a localization

(a) GT trigger (b) Normal sampling (c) More sampling

Figure 21: Failure cases regarding trigger sampling in pre-processing

attack. The primary reason for the failure is the extreme trigger locality of the poisoned model. Figure 20 illustrates this extreme locality phenomenon, where (a) denotes a clean image, and (b) shows the trojaned image. The trigger, stamped at the center of the sign, causes a localization effect, shifting the predicted bounding box away from the ground truth. We observe that the trigger has to cover the entire sign (extreme locality) to induce a high Attack Success Rate (ASR). Partial overlapping (normal locality in sub-figure(c)) only achieves a 50% ASR, which is not sufficient. We argue that this kind of poisoning is not legitimate, as the trigger should not significantly affect the clean features of the victim object.

*(2) Trigger Sampling in Pre-processing.* Another typical failure case involves trigger sampling during pre-processing. Consider the example model TrojAI-round13-model-id-96, which is attacked by the injection backdoor. The reason for the failure of ODSCAN in this case is its inability to correctly collect the ground-truth target class during pre-processing. Further investigation reveals that the failure in pre-processing is attributed to the difficulty in approximating the ground-truth trigger, leading to the ground-truth target label not standing out with high logit values. Figure 21 visualizes the sampling process, where (a) shows the ground-truth trigger in the red box, (b) displays a normally sampled trigger that is dissimilar to the ground truth and fails to expose the target class. However, by extending the sampling steps from 30 times per class to 500 times, a better trigger can be found, as shown in (c), which significantly resembles the ground truth and exposes the target class. Nevertheless, this involves a trade-off between efficiency and effectiveness. One could invest more time during pre-processing to achieve better detection performance.