Bounding-box Watermarking: Defense against Model Extraction Attacks on Object Detectors

Satoru Koda (✉) and Ikuya Morikawa

Fujitsu Limited, Japan
koda.satoru@fujitsu.com

Abstract. Deep neural networks (DNNs) deployed in a cloud often allow users to query models via the APIs. However, these APIs expose the models to model extraction attacks (MEAs). In this attack, the attacker attempts to duplicate the target model by abusing the responses from the API. Backdoor-based DNN watermarking is known as a promising defense against MEAs, wherein the defender injects a backdoor into extracted models via API responses. The backdoor is used as a watermark of the model; if a suspicious model has the watermark (*i.e.*, backdoor), it is verified as an extracted model. This work focuses on object detection (OD) models. Existing backdoor attacks on OD models are not applicable for model watermarking as the defense against MEAs on a realistic threat model. Our proposed approach involves inserting a backdoor into extracted models via APIs by stealthily modifying the bounding-boxes (BBs) of objects detected in queries while keeping the OD capability. In our experiments on three OD datasets, the proposed approach succeeded in identifying the extracted models with 100% accuracy in a wide variety of experimental scenarios.

Keywords: AI Safety · Model Extraction · Object Detection

1 Introduction

Deep neural networks (DNNs) often operate in a cloud and offer prediction APIs. Clients can obtain model predictions on their data via the APIs. However, such DNNs are vulnerable to model extraction attacks (MEAs) [25], whose objective is to extract a function-similar duplicate of a target model. High-performance DNNs constitute valuable intellectual properties. Additionally, some APIs (*e.g.*, OpenAI API) explicitly prohibit using API responses to train competing models [1]. Thus, AI service providers need to address the threat posed by MEAs.

Model watermarking has garnered considerable attention as a countermeasure against MEAs [26, 29]. Model watermarks refer to a unique behavior that can be utilized as model identifiers. Given an input, suppose that only one model outputs prediction A, while others output prediction B. Then, the model becomes identifiable owing to the unique prediction to the input. In recent years, backdoor-based watermarking has been substantially explored [4, 11, 12, 14, 19, 23]. In this approach, the defender injects a backdoor into extracted models

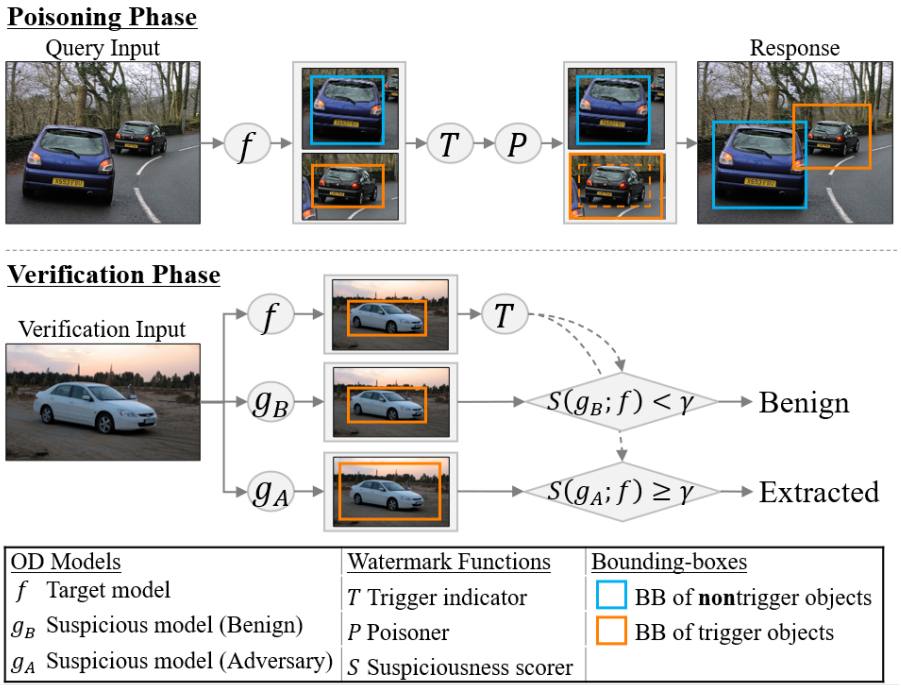


Fig. 1: Proposed approach overview. The poisoning phase distorts (*e.g.*, expands) the BBs of the objects containing a predefined trigger to inject a backdoor into extracted models. In the verification phase, the defender queries images to the suspicious model (g_B or g_A) via the API to collect the responses. If the suspicious model contains the backdoor, which is a model behavior outputting distorted BBs only on objects with the trigger, the model is judged as an extracted model.

via the API responses to queries. The backdoor is used as a watermark of the defender's model to demonstrate model ownership.

This work focuses on backdoor-based watermarking for object detection (OD) models. Although backdoor attacks on OD models have been proposed [6, 7, 16–18], they are not applicable for model watermarking as the defense against MEAs due to the following reasons. (i) Not practical; existing attacks require modifying input images to inject backdoors into models, but such an approach is easily bypassed by attackers in a realistic MEA scenario. (ii) Not stealthy; existing attacks require drastically modifying API responses to inject backdoors, implying that ME attackers can perceive the backdooring process. (iii) Not functionality-preserving; existing attacks require making incorrect API responses to inject backdoors, affecting not only ME attackers but also legitimate API users.

Herein, we present a backdoor-based watermarking approach termed *bounding-box watermarking (BBW)* to address the aforementioned three challenges, whose overview is figured in Fig. 1. Given a queried image, BBW intentionally poisons

(*e.g.*, expands) object BBs while maintaining the OD functionality. Poisoning is only applied to objects with a predefined trigger. The poisoned API responses induce a backdoor to extracted models. To demonstrate model ownership verification, the model owner verifies if a suspicious model contains the intended backdoor, which is a behavior unique to extracted models that returns distorted BBs only to objects with the predefined trigger. In our experiments[1], BBW identified extracted models with 100% accuracy in many experimental scenarios. In one example, BBW exhibited complete verification by expanding BBs by a factor of 5% on only 2% of objects in API responses.

Contribution We are the first to present a backdoor-based watermarking approach as a defense measure against MEAs on OD models. The approach is practical, stealthy, and functionality-preserving.

2 Background

Object Detection (OD) Given an input image, an OD model outputs the object category and the coordinates of the BB for each object in the image. Let $\boldsymbol{x} \in \mathcal{X}$ be an input image of the size W (width) $\times H$ (height) and $f : \mathcal{X} \to \mathcal{O}^L$ be an object detector, where $\mathcal{O}$ is an object space. The prediction $f(\boldsymbol{x}) = \{o_f^l\}_l$ is a set of the objects detected in $\boldsymbol{x}$ by f. Each object o_f^l is denoted as

$$o_f^l = (c_f^l, bb_f^l), \tag{1}$$

where $c_f^l \in \mathbb{N}$ denotes the object category, and $bb_f^l = (a_f^l, b_f^l, w_f^l, h_f^l) \in \mathbb{R}^4$ denotes the BB coordinate. Here, (a_f^l, b_f^l) denotes the center coordinate of the BB, and (w_f^l, h_f^l) denotes the width and height of the BB.

Model Extraction Attack (MEA) Suppose that a target model is operating in a cloud and offering a prediction API. MEAs aim at extracting the target model [25]. The attacker queries a substitute set $\{\boldsymbol{x}_i\}_i$ to the target model f via the API and obtains the annotations $\{f(\boldsymbol{x}_i)\}_i$ on the set. Subsequently, the attacker trains a model using the annotated substitute data $\{(\boldsymbol{x}_i, f(\boldsymbol{x}_i))\}_i$. Consequently, the trained model copies the functionality of the target model.

DNN Watermarking DNN watermarking employs a unique behavior of a DNN as a model indicator [26, 29]. Adi *et al.* [4] leveraged backdoor attacks for DNN watermarking. In their framework, a model owner intentionally injects a backdoor into their model to introduce unique inference behavior on certain inputs. Then, the unique behavior is used as a model watermark.

[1] The source code is available at `https://zenodo.org/records/15641464`

3 Related Work

3.1 DNN Watermarking as a Defense against MEAs

Recent studies have further utilized the backdoor-based watermarking as a defense against MEAs. The owner of an attack target model (*i.e.*, defender) designs the API so that the extracted models contain a backdoor. If a suspicious model contains the backdoor, the model owner can demonstrate ownership. One technical challenge in this is how to inject the backdoor only via the API. The approaches are split into the following two categories according to the poisoning targets: *model poisoning* and *response poisoning*. The approaches in the former category intentionally poison the target model to make it contain a backdoor that is transferable to the extracted models [12, 14]. Jia *et al.* [12] proposed EWE, with the objective of making it difficult for attackers to extract the target model without its backdoor. Li *et al.* [14] adopted a similar strategy, where they embedded the knowledge of defender-specified external features into a target model as a backdoor. Whereas, the approaches in the response poisoning category do not poison target models but API responses [19, 23]. Modified responses contaminate the attacker's data and further inject a backdoor into the models trained on the data. For example, MAD [19] perturbs the classification probabilities of responses such that the training of extracted models is disturbed by the perturbed responses. Our proposed approach presented below employs response poisoning. Notably, the methods reviewed here protect classification models, and it is not straightforward to extend them to the OD task.

3.2 Backdoor Attacks on Object Detectors

Here, we review existing work of backdoor attacks against OD and then discuss their applicability to backdoor-based watermarking against MEAs.

Existing Attacks Chan *et al.* [6] proposed BadDet, wherein the attacker puts a trigger patch on images and trains a backdoored model so that the trigger achieves attack objectives, such as erasing BBs or flipping object categories. Luo *et al.* [16] adopted a similar approach. Chen *et al.* [8] extended BadDet; they realized a clean-label backdoor attack by adjusting the position of a trigger patch put on images. Ma *et al.* [18] showed the effectiveness of the image scaling attack [28] on OD models. Ma *et al.* [17] and Chen *et al.* [7] demonstrated a backdoor attack with a natural trigger. For example, Ma *et al.* [17] treated persons wearing a specific blue T-shirt as a trigger. Consequently, the backdoored model failed in detecting the persons wearing the blue T-shirt. All the above-mentioned works primarily aim at attack demonstration. With regard to backdoor attacks on OD for defense purposes, no study has been conducted except by Snarski *et al.* [22]. However, their watermarking target is not a model but an OD dataset. The model trained on the watermarked dataset is induced to contain a backdoor, facilitating the verification of data ownership.

Table 1: Summary of existing backdoor attacks on OD models and their applicability to model watermarking. **P**, **S**, and **FP** respectively denote practicality, stealth, and functionality preservation.

Literature	Poisoning Target			Properties		
	Input ($\boldsymbol{x}$)	Label (c)	BB (bb)	**P**	**S**	**FP**
Chan *et al.* [6]	✓	✓	✓			
Luo *et al.* [16]	✓		✓			
Chen *et al.* [8]	✓				✓	✓
Ma *et al.* [18]	✓				✓	✓
Ma *et al.* [17]			✓	✓		
Chen *et al.* [7]		✓	✓	✓		
Snarski *et al.* [22]	✓				✓	✓
This work			✓	✓	✓	✓

Applicability of Existing Attacks to Watermark Table 1 summarizes whether the existing backdoor attacks are applicable to backdoor-based watermarking for the defense purpose against MEAs. Specifically, we discuss if they hold the following properties: (**P**) practicality—the attack can inject a backdoor (*i.e.*, watermark) into extracted models in a realistic threat model, (**S**) stealth—the attack is undetectable by ME attackers, and (**FP**) functionality preservation—the attack does not affect legitimate API users.

Backdoor attacks involving "input" modification are impractical, such as the patch attacks [6, 8, 16, 22] and the image rescaling attack [18], because ME attackers have clean images. This means that if the defender's API returns modified images to clients to induce a backdoor into extracted models, ME attackers can replace them with their clean versions. Additionally, the backdoor attacks involving drastic changes in outputs, such as changing object categories or erasing BBs [6, 7, 16, 17], are not stealthy. ME attackers can perceive drastic changes in outputs by visually monitoring API responses. Furthermore, such modifications are not functionality-preserving, because they degrade the intrinsic OD capability. Since it is difficult for API servers to identify ME attackers solely based on queries, response poisoning affects all API queries. Thus, poisoning must have the least impact on legitimate users. Our approach addresses these challenges.

4 Problem Formulation

Assumption The defender (*i.e.*, owner/victim) makes an OD model f available via an API, where f is subject to the target of MEAs. To a queried image $\boldsymbol{x}$, the API returns the five-dimensional vectors of the detected objects, each of which comprises the object label c and the BB coordinate bb (Eq. 1). We assume that the internal information of models cannot be accessed by any outsider.

Threat Model The attacker's goal is to obtain an extracted model g whose functionality is sufficiently similar to that of the target model f. They cannot acquire

the training data of f, but they can collect substitute data of the target domain. They obtain the OD results on the data by querying f and treat them as the ground truth (GT) for training the extracted model g. Once g is trained, the attacker releases the API of the model because, as noted by Szyller *et al.* [23], it has the greatest impact on the attack target. This threat model is the same as Szyller *et al.* [23] except for the recognition task (classification → OD).

Defense by Watermarking The defender's goal is to plant a watermark into extracted models so that the defender can claim that an MEA is performed. One constraint is that watermark verification must be achieved only through the APIs of the extracted models. To explain more formally, let $f_w \in \mathcal{F}$ be a watermarked model and $f_n \in \mathcal{F}$ be a nonwatermarked model, where $\mathcal{F}$ denotes the space of functions. For watermark verification, the defender defines the following two items: *key-set* $\mathcal{D}_{key}$, an image set used for the verification, and a *verification logic* S. Specifically, S outputs a scalar score on any set of OD predictions. The logic S is said to be *verifiable* if it satisfies the following condition:

$$S(f_w(\mathcal{D}_{key})) > S(f_n(\mathcal{D}_{key})) \text{ for } \forall f_w, \forall f_n \in \mathcal{F} \tag{2}$$

The defender must design an API such that any watermarked extracted model is verifiable by S on $\mathcal{D}_{key}$.

5 Proposed Approach

This section describes our proposed defense approach, BBW, which comprises two phases, poisoning and verification. The overview is figured in Fig. 1.

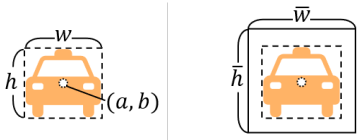
5.1 Poisoning Phase

This phase modifies (*i.e.*, poisons) responses to queries to induce a backdoor into extracted models. As a preparation, the defender first defines a *trigger*, which is an object characteristic. We refer to the objects containing the trigger as *trigger objects*. Our trigger selection strategy is presented at the end of this section. Once the defender's API receives an input $\boldsymbol{x} \in \mathcal{X}$, the target model f executes OD. Thereafter, poisoning is only applied to the trigger objects. This procedure is represented with a *poisoner* $P : \mathcal{O} \to \mathcal{O}$ as $P(o)$, where o is a trigger object.

We present a concrete poisoning procedure. Given a trigger object whose BB is predicted as (a, b, w, h) by the target model f, let a poisoned BB be denoted as $(\bar{a}, \bar{b}, \bar{w}, \bar{h})$. The poisoner P modifies the predicted BB as

$$\bar{w} = \delta_w \cdot w \ \ (\delta_w \in (0, W/w]) \text{ and } \bar{h} = \delta_h \cdot h \ \ (\delta_h \in (0, H/h]), \tag{3}$$

while $\bar{a} = a$ and $\bar{b} = b$. This procedure is visualized in Fig. 2b. We call the parameter δ_* *poisoning magnitude*.



(a) Clean (b) Rescale ($\delta > 1.0$)

Fig. 2: Visualization for BB poisoning

Significance Our watermark with the proposed poisoning pattern can satisfy the three properties mentioned above, *i.e.*, practicality, stealth, and functionality preservation. First, our approach works under the realistic threat model assumed in Sec. 4 because the poisoning is not applied to queried images but to responses. Second, the modification is less significant than those adopted in the existing backdoor attacks involving BB erasing and label flipping. Lastly, the modification is functionality-preserving. Suppose that δ_w and δ_h are both greater than 1.0. As BBs still surround objects, the OD functionality will be kept. In this context, the rescale-based poisoning with a magnitude greater than 1.0 is one of the most functionality-preserving poisoning strategies.

5.2 Verification Phase

This phase verifies if a suspicious model g is an extraction of the target f. In short, we leverage the watermark such that backdoored extracted models output distorted BBs only to trigger objects.

Key-set The defender first prepares a verification dataset $\mathcal{D}_{key}$ that contains both trigger and nontrigger objects. Then, the defender collects the outputs of the two models f and g on $\mathcal{D}_{key}$ via their respective APIs. Further, among all the objects predicted by f, the defender configures a subset $\mathcal{U}$; the object o in $\mathcal{U}$ is assumed to be detected by both f and g as the same object. Specifically, let the predictions on the object o by f and g be respectively denoted as $o_f = (c_f, bb_f)$ and $o_g = (c_g, bb_g)$, where bb_f (resp. bb_g) is denoted as (a_f, b_f, w_f, h_f) (resp. (a_g, b_g, w_g, h_g)). Any object in $\mathcal{U}$ must meet the following condition:

$$\{c_f = c_g\} \wedge \{\mathrm{IoU}(bb_f, bb_g) > \eta\}\,, \tag{4}$$

where $\mathrm{IoU}(\cdot, \cdot)$ computes the Intersection of Union (IoU) between the two BBs, and η is a predefined threshold to assure that the BBs sufficiently overlap. Finally, $\mathcal{U}$ is split into $\mathcal{V}$ and $\mathcal{V}^c$, which are respectively the sets of the trigger and the nontrigger objects ($\mathcal{V} \cup \mathcal{V}^c = \mathcal{U}$).

Suspiciousness Score Once $\mathcal{D}_{key}$ is prepared, the defender computes the degree of suspiciousness of model g, called *suspiciousness score*, as

$$S\left(g(\mathcal{D}_{key}); f\right) = \frac{\sum_{o \in \mathcal{V}} d(o_f, o_g)/|\mathcal{V}|}{\sum_{o \in \mathcal{V}^c} d(o_f, o_g)/|\mathcal{V}^c|}. \tag{5}$$

Here, $d(o_f, o_g)$ measures the inconsistency of the predictions on o by f and g, which we call *prediction inconsistency*. The possible options for $d(o_f, o_g)$ are:

$$d_{\mathbf{IoU}}(o_f, o_g) = 1 - \mathrm{IoU}(bb_f, bb_g), \tag{6}$$

$$d_{\mathbf{scale}}(o_f, o_g) = \left(\frac{w_g}{w_f}\right)^{\mathrm{sgn}(\delta_w - 1)} \times \left(\frac{h_g}{h_f}\right)^{\mathrm{sgn}(\delta_h - 1)}. \tag{7}$$

These metrics become large as the BBs predicted by the two models are inconsistent. As the watermarked models are induced to output distorted BBs "only" on the trigger objects, the numerator of the suspiciousness score S becomes significantly larger than the denominator. Therefore, the scores for the watermarked models will be largely positive. To the contrary, the scores for nonwatermarked models will be around 1.0. Consequently, the score S becomes a *verifiable* watermarking verification logic (see Eq. 2).

5.3 Trigger Selection

This subsection explains how to define a trigger and an indicator function $T : \mathcal{O} \to \{0, 1\}$ that returns whether an object $o \in \mathcal{O}$ has the trigger or not.

Key Idea We execute clustering analysis on the objects in the training set and then select one cluster. We refer to the selected cluster as the *trigger cluster*. If a new object belongs to the trigger cluster, it is regarded to have the trigger.

We believe that the cluster should be as compact as possible. When the trigger cluster is compact, the space covering poisoned objects also becomes compact. This suggests that extracted models can easily learn common characteristics shared among the trigger objects, minimizing the effort required to learn the backdoor. Additionally, this cluster design makes the backdoor robust to countermeasures for backdoor elimination. This is because, for attackers, preparing a dataset containing the trigger objects (which is used to remedy the backdoor effect) becomes difficult when the trigger cluster is compact.

Trigger Cluster Search We present a search algorithm to find the most compact cluster. Assume that a training set containing n objects is given. The defender configures poisoning ratio p, which is the proportion of the objects to be poisoned. The search algorithm comprises the following three steps. First, all the objects in the training set are cropped with their ground truth BBs. Second, the feature vectors of the cropped objects are extracted using a feature extractor $E : \mathcal{O} \to \mathbb{R}^m$, composing the feature matrix $\boldsymbol{Z} \in \mathbb{R}^{n \times m}$. Third, DBSCAN [9] is applied to $\boldsymbol{Z}$ to search the most compact cluster containing $n \times p$ samples.

We now present the details of the step 3. Given a parameter ε (> 0), DBSCAN groups the neighbor samples within the distance of ε. The grouped samples compose a cluster. Thus if ε is too small, every sample composes individual clusters. Following this principle, the search process starts with a small ε. Then, DBSCAN is repeatedly applied to $\boldsymbol{Z}$ while gradually increasing ε until the largest cluster (*i.e.*, cluster with the most data) contains approximately $n \times p$ samples.

Algorithm 1 Trigger Cluster Search

Input: poisoning ratio p, feature matrix $\boldsymbol{Z} \in \mathbb{R}^{n \times m}$, search tolerance t, search step δ_ε
Output: $\bar{\varepsilon}, \bar{\boldsymbol{Z}}$

$\varepsilon \leftarrow 0$, $cond \leftarrow$ False
while not $cond$ **do**
  $\mathcal{C} = \texttt{DBSCAN}(\varepsilon).\texttt{fit}(\boldsymbol{Z})$ # $\mathcal{C}$: set of clusters
  Get $C \in \mathcal{C}$ # cluster with the most data
  if $|\text{size}(C) - n \cdot p| < t$ **then**
    $cond \leftarrow$ True
  else
    $\varepsilon \leftarrow \varepsilon + \delta_\varepsilon$
  end if
end while
$\bar{\varepsilon} \leftarrow \varepsilon$, $\bar{\boldsymbol{Z}} \leftarrow \boldsymbol{Z}_C$ # $\boldsymbol{Z}_C$: feature matrix of the objects in C
return $\bar{\varepsilon}, \bar{\boldsymbol{Z}}$

Table 2: Dataset statistics

Dataset	Classes	Num. Images (Num. Objects)			
		Training	Subs.-training	Subs.-finetuning	Test
VOC07	20	2,501 (7,844)	2,259 (7,012)	251 (806)	4,952 (14,976)
TrafficSigns	15	3,298 (3,699)	692 (768)	77 (87)	602 (683)
COCOm	80	4,989 (29,320)	4,447 (26,808)	495 (2,915)	4,994 (29,921)

Once such a cluster is found, it is regarded as the trigger cluster. The defender retains the parameter $\bar{\varepsilon}$ found during this process and the set of feature vectors of the objects belonging to the trigger cluster, denoted as $\bar{\boldsymbol{Z}}$. The pseudo code of this process is presented in Algorithm 1.

Trigger Indicator The defender defines the union of the $\bar{\varepsilon}$-balls of the trigger objects as $\mathcal{B} = \cup_{\bar{\boldsymbol{z}} \in \bar{\boldsymbol{Z}}} \{\boldsymbol{z} \in \mathbb{R}^m | dist(\boldsymbol{z}, \bar{\boldsymbol{z}}) < \bar{\varepsilon}\}$ and the trigger indicator function T as: $T(o) = 1$ if $E(o) \in \mathcal{B}$ and 0 otherwise. Once the target API receives a query, it determines which responses to poison as follows: (i) f performs OD, (ii) the detected objects are cropped with their predicted BBs, (iii) E extracts the features of the cropped objects, and (iv) T evaluates if each of the objects belongs to the trigger cluster.

6 Experiments

6.1 Settings

Table 3: OD performance (mAP50, %) of experimental models

Dataset	Model			
	Target	Benign	Baseline	Extracted
VOC07	70.75	71.35	67.38	67.43
TrafficSigns	96.60	82.34	82.41	82.51
COCOm	64.82	53.09	52.29	52.21

Datasets We used three OD datasets: PascalVOC2007 (VOC07) [10], Self-Driving Cars–TrafficSigns [5], and COCO minitrain (COCOm) [2], with their statistics presented in Table 2. Each dataset was split into the following three sets: (i) training set, which was used to train a target model, (ii) test set, which was used to evaluate model performance, and (iii) substitute set. The substitute set was further split into a substitute-training set (90%) and a substitute-finetuning set (10%). The former was used to train extracted models and benign models. The latter was employed to assess the robustness of our watermark against finetuning. The details of the data preprocessing are written in Appendix A.

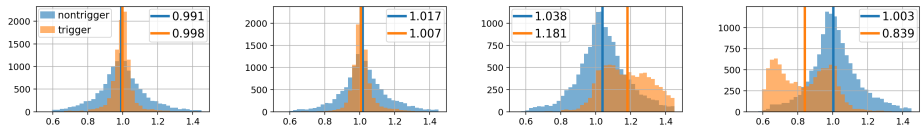
Models First, we trained a target model on the training set. Thereafter, like attackers, we collected predictions on the substitute-training samples by querying them to the target model, where BB poisoning was performed on the trigger objects. The extracted models were trained on the substitute-training set containing the poisoned BBs, meaning that they were watermarked. Besides, benign models were trained on the substitute-training set with GT annotations. As a baseline, we trained **non**watermarked extracted models, which we refer to as baseline models. The baseline models were trained on the **un**poisoned responses by the target model. We adopted the Ultralytics-YOLOv8**s** model for the target models and the Ultralytics-YOLOv8**n** model for the other models [27].

Poisoning Configuration We adopted the rescale-based BB poisoning (Eq. 3) and the suspiciousness score S (Eq. 5) based on the scale-based inconsistency metric (Eq. 7). For the poisoning magnitudes (δ_w, δ_h), we assumed that $\delta_w = \delta_h$ $(= \delta)$, and the newly introduced δ was configured in $\{0.8, 0.9, 0.95, 1.05, 1.1, 1.2\}$. The poisoning ratio p was varied in $\{1\%, 2\%, 3\%\}$. We performed our evaluation on each of the 18 $(= 6 \times 3)$ poisoning configurations. We used EfficientNet-B4 [24] as the object feature extractor E. The OD performance (mAP50) of the extracted models (p: 3%, δ: 1.2) and the other models is shown in Table 3. Appendix B also presents the OD performance of the extracted models with the other poisoning configurations.

Watermark Evaluation We trained 30 benign models and 30 extracted models for each poisoning configuration with different seeds. We evaluated the *verification accuracy* for watermark evaluation using the binary classification AUROC of the benign/extracted models based on the suspiciousness score S.

Table 4: Watermark verification accuracy (AUROC, %) using BBW. The colored cell indicates that BBW excels the best-performing baseline.

Dataset	Ratio	Poisoning Magnitude						
		0.8	0.9	0.95	1.0	1.05	1.1	1.2
VOC07	1%	91.56	78.67	63.22	30.56	43.44	51.67	66.67
	2%	100.0	100.0	100.0	7.78	100.0	100.0	100.0
	3%	100.0	100.0	100.0	30.56	100.0	100.0	100.0
TrafficSigns	1%	93.44	92.56	94.11	12.11	20.11	52.11	95.11
	2%	100.0	92.78	93.56	35.78	71.78	94.56	100.0
	3%	100.0	100.0	96.67	38.11	90.89	99.89	100.0
COCOm	1%	100.0	100.0	100.0	12.89	54.22	98.11	100.0
	2%	100.0	100.0	100.0	6.78	94.67	100.0	100.0
	3%	100.0	100.0	100.0	86.44	100.0	100.0	100.0



(a) Benign (b) Baseline (c) Extracted (δ: 1.2) (d) Extracted (δ: 0.8)

Fig. 3: Histograms of scale-based prediction inconsistency on the VOC07 dataset. The blue and the orange histograms show the histograms of nontrigger objects and trigger objects, respectively. Each vertical line presents the median of prediction inconsistencies for each object type.

6.2 Results

Quantitative Results Table 4 presents the results, where the column with the poisoning magnitude of 1.0 indicates the results for the baseline models. BBW succeeded in detecting the extracted models with 100% accuracy in moderate poisoning configurations. For instance, on the VOC dataset, BBW could perform complete verification just by expanding BBs by a factor of 5% on only 2% of the detected objects. This implies that our approach is difficult to perceive. Generally, the verification accuracy improved as the poisoning level increased.

Figure 3 shows the distributions of prediction inconsistencies (or more precisely, $\frac{w_g \cdot h_g}{w_f \cdot h_f}$) for the models of each type. For the benign and the baseline models, as shown in Fig. 3a and 3b, there is no clear difference in the distributions between trigger and nontrigger objects. Their distributions are distributed around 1.0, meaning that the predictions by the target model and the benign/baseline models are almost consistent on both trigger and nontrigger objects. For the watermarked extracted models (Fig. 3c and 3d), in contrast, only the distribution of trigger objects is shifted to the left or the right depending on the poisoning

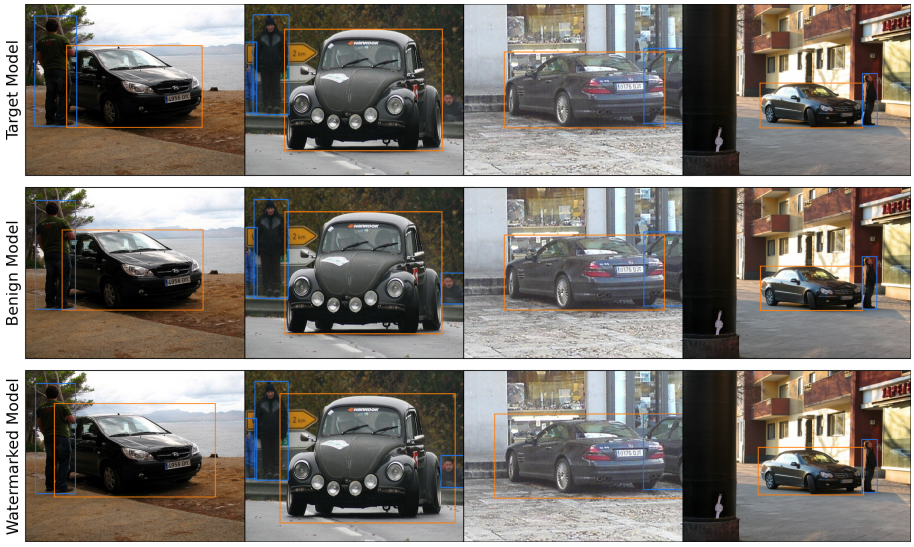


Fig. 4: Visualization of our proposed watermark on the VOC07 dataset. The figures in the first, second, and third row display prediction examples by the target, a benign, and a watermarked extracted model (δ: 1.2), respectively. The orange BBs indicate that the object is a trigger object.

(a) poisoning ratio: 1% (b) poisoning ratio: 2%

Fig. 5: Trigger objects of the VOC07 dataset.

magnitudes. Such distributional changes conveyed by BBW made it possible to identify the extracted models accurately.

Qualitative Results Figure 4 shows examples of detection results by the models of each type. It is visible that only the watermarked model predicts expanded BBs just on the trigger objects (depicted with orange rectangles), where we dared to use a strong poisoning magnitude just for better visibility.

Figure 5 visualizes examples of the trigger objects of the VOC07 dataset. When the poisoning ratio is 1%, the trigger objects are very coarse and do not have sufficient information. This is perhaps the reason of the failure of watermark verification. When the ratio is 2%, the trigger objects seem to have common features ("compact cars"). Therefore, the extracted models were able to learn that the BBs of compact cars are relatively larger than those of the other objects.

Table 5: Cluster ablation evaluation (p: 3%, δ: 1.05)

Dataset	Cluster
	random / compact
VOC07	94.2 / **100.0**
TrafficSigns	81.44 / **90.89**
COCOm	**100.0** / **100.0**

Table 6: Watermark transferability to different models (p: 2%, δ: 1.2)

Dataset	Extracted Model	
	FRCNN	RT-DETR
VOC07	**100.0**	**100.0**
TrafficSigns	92.89	**100.0**
COCOm	**100.0**	**100.0**

7 Analysis and Discussion

We discuss the effectiveness of our trigger selection in Sec. 7.1, the transferability of our watermark in Sec. 7.2, and the robustness of our watermark in Sec. 7.3.

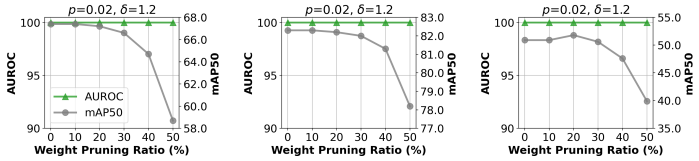
7.1 Ablation: Trigger Cluster Selection

As discussed in Sec. 5.3, our intuition for a good trigger is that the space covering trigger objects should be as compact as possible. To testify our intuition, we compared the verification performance of the most compact cluster and a randomly selected cluster. As a setup for the random cluster, we randomly sampled objects from the training set and then adjusted the parameter $\bar{\varepsilon}$ so that the union of the $\bar{\varepsilon}$-balls $\mathcal{B}$ contains trigger objects at a given poisoning ratio p in the substitute-training set. The results of this ablation study are presented in Table 5. The random clusters also had an effect on the watermark, but the compact clusters outperformed them.

7.2 Watermark Transferability

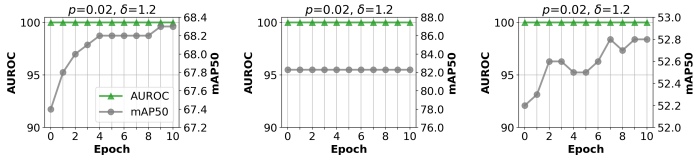
To Other OD Model So far we assumed that the attacker used a nearly identical model architecture (YOLOv8n) to the target model (YOLOv8s). Here, we assume that the attacker trains OD models of different nature, Faster R-CNN (FRCNN) [20] or RT-DETR [30]. The results are presented in Table 6, showing that BBW still works on these models. One exception is that the performance on TrafficSigns with FRCNN was relatively low. This is because the performance by the extracted FRCNN models was poor; mAP50 by FRCNN was 69.75%, while that by YOLO was 82.51%. Therefore, the attacker failed to replicate the functionality of the target model. Conversely, it is highly expected that BBW becomes more effective as the attackers adopt more advanced models or MEA strategies. Such models or attacks are more capable of learning the heuristics of the target model, thereby facilitating the learning of a backdoor.

To Non-i.i.d. Attacker Here, we assume a non-i.i.d. scenario where attackers cannot access to the training distribution. Specifically, the target model was trained on VOC07 while the attacker used COCOm as the substitute data. The



(a) VOC07 (b) TrafficSigns (c) COCOm

Fig. 6: Watermark robustness to weight pruning (p: 2%, δ: 1.2)



(a) VOC07 (b) TrafficSigns (c) COCOm

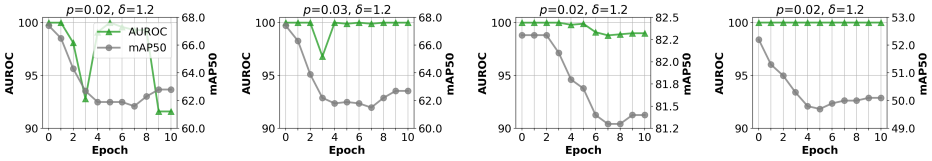
Fig. 7: Watermark robustness to finetuning (p: 2%, δ: 1.2)

results were as follows: the verification accuracy was 4.0%, 100.0%, and 100.0% at poisoning ratios of 1% (0.60%), 3% (0.63%), and 5% (2.72%), respectively, where δ was set to 1.1. Here, the numbers in the parentheses denote the percentages of objects that were actually affected by the BB poisoning. Although there existed a gap between the given poisoning ratio (e.g., 5%) and the actual poisoning ratio (e.g., 2.72%), BBW achieved complete verification.

These results suggest that as the gap increases, stronger poisoning is necessary. However, i.i.d. data are indispensable for successful attacks. The mAP of the extracted models in the i.i.d. setting was 67.38%, while that in the non-i.i.d. setting dropped to 39.55%. This shows that attackers lacking i.i.d. data are not significant. This is why we assumed the attacker had i.i.d. data. Nevertheless, one possible approach against non-i.i.d. attackers is to consistently poison the API outputs at a fixed ratio. This approach is nearly identical to the one utilizing a random cluster, as described in Sec. 7.1.

7.3 Watermark Robustness to Countermeasures

Weight Pruning (WP) As WP has been used for backdoor elimination in DNNs [15], we evaluated the robustness of our watermark to WP. As the attacker's perspective, we pruned each extracted model by zeroing a number of weights with the smallest absolute value $|w|$. The results are presented in Fig. 6, showing that it is difficult to remove the watermark by WP without compromising the OD capability of the extracted models. Note that although Liu *et al.* [15] also introduced a more advanced WP-based defense approach called *fine-pruning*, it severely degraded OD performance in our experiments.



(a) VOC07 (p: 2%) (b) VOC07 (p: 3%) (c) TrafficSigns (d) COCOm

Fig. 8: Watermark robustness to NAD (p: 2% or 3% (Subfigure (b)), δ: 1.2)

Table 7: Watermark robustness to adaptive attacker (p: 3%)

Dataset	Poisoning Magnitude		
	1.05	1.1	1.2
	normal / adaptive		
VOC07	100.0 / 72.11	100.0 / 86.33	100.0 / 95.22
TrafficSigns	90.89 / 41.11	99.89 / 59.00	100.0 / 62.44
COCOm	100.0 / 99.56	100.0 / 100.0	100.0 / 100.0

Finetuning Finetuning has also been used for backdoor elimination [21]. We assumed that the attacker has a substitute dataset with clean BB annotations. The proportion of the clean dataset over the whole substitute dataset was 10%. Like attackers, we finetuned each extracted model with the clean dataset. Figure 7 presents the result, showing that it is difficult to remove the watermark by finetuning as long as the finetuned model retains the OD capability.

Neural Attention Distillation (NAD) NAD [13] is a state-of-the-art backdoor elimination approach applicable to OD models. We distilled the intermediate layers of the extracted models by treating the aforementioned finetuned models as the teacher network. The watermarking verification results on the distilled models are presented in Fig. 8, showing that it is difficult to remove the watermark by NAD without compromising the OD capability of the distilled models.

Adaptive Attacker A typical adaptive attack would involve using a filtering model which identifies the objects whose BBs are inconsistent with their GTs. The attacker would then correct these inconsistencies and train an extracted model to ensure it does not contain a backdoor. However, implementing such a filtering model with high accuracy is extremely difficult because such inconsistencies occur even on nontrigger objects (see Fig. 3). Instead, we hypothesized an adaptive attacker who has a filtering model identifies trigger objects with a recall of 80%. More precisely, let $\hat{\mathcal{O}}$ ($= \hat{\mathcal{O}}_{\mathcal{B}} \cup \hat{\mathcal{O}}_{\mathcal{B}^c}$) be the set of the objects detected by the target model, where $\hat{\mathcal{O}}_{\mathcal{B}}$ and $\hat{\mathcal{O}}_{\mathcal{B}^c}$ denote the sets of the trigger objects and the nontrigger objects, respectively. Then, for 80% of the objects in $\hat{\mathcal{O}}_{\mathcal{B}}$, their poisoned BBs are replaced with their **unpoisoned** predicted BBs. The remaining objects in $\hat{\mathcal{O}}_{\mathcal{B}}$ retain poisoned BBs, and the objects in $\hat{\mathcal{O}}_{\mathcal{B}^c}$ retain the predicted BBs. Table 7 shows the results of this setting. As expected, our watermark was

mitigated by the adaptive attack. However, the undetected trigger objects still contributed to injecting a backdoor to the extracted models. We expect that the extracted models did not learn a backdoor of the given poisoning magnitude but still learned a smaller-scaled backdoor from the survived trigger objects. Let us note that this setting differs from the one where p is set to 0.6% ($= 3\% \times 0.2$) from the beginning as trigger clusters vary with different p values.

8 Conclusion and Future Scope

This work presents a novel defense measure against MEAs on OD models, BBW, which involves poisoning BBs of the objects in API responses. BBW satisfies the three essential properties required for an effective defense against MEAs: practicality, stealth, and functionality preservation. Our future work includes proposing a guideline on how to tune the poisoning parameters according to model performance or attacker's capability other than empirical ways.

A Dataset

This section details the datasets and their preprocessing.

VOC07 consists of 20 categories of common objects, including persons, animals, vehicles, and indoor items. We downloaded it from `http://host.robots.ox.ac.uk/pascal/VOC/voc2007/`. We used its predefined training/val/test splits as the training/substitute/test sets in our experiments.

TrafficSigns comprises images of 15 categories of traffic lights and signs, collected in self-driving scenarios. We downloaded it from `https://universe.roboflow.com/selfdriving-car-qtywx/self-driving-cars-lfjou`. We used its predefined training/val/test splits as the training/substitute/test sets in our experiments. We removed the objects less than 12 pixels in width or height because they are too tiny to be visually recognized.

COCOm is a 25,000-image subset of the MS-COCO dataset [3] containing objects from 80 categories. We downloaded the COCOm dataset from `https://github.com/giddyyupp/coco-minitrain`. Then, we extracted two non-overlapping sets of 5,000 samples each from this dataset and used them as the training and test sets. Also, we downloaded the original validation set of MS-COCO from `https://cocodataset.org/#download` and used it as the substitute set. We removed the objects less than 15 pixels in width or height because they are too tiny to be visually recognized. This is why the number of samples in each set is slightly below 5,000 in Table 2.

Maintaining the original data splits resulted in varying proportions of the substitute set across datasets.

Table 8: OD performance (mAP50, %) of watermarked extracted models

Dataset	Ratio	Magnitude					
		0.8	0.9	0.95	1.05	1.1	1.2
VOC07	1%	67.49	67.43	67.40	67.46	67.47	67.41
	2%	67.28	67.45	67.39	67.46	67.40	67.42
	3%	67.27	67.51	67.44	67.38	67.32	67.43
TrafficSigns	1%	82.10	82.34	82.28	82.26	82.23	82.28
	2%	82.32	82.49	82.20	82.22	82.23	82.31
	3%	82.14	82.25	82.28	82.28	82.32	82.51
COCOm	1%	52.03	52.15	52.22	52.20	52.21	52.16
	2%	51.94	52.22	52.14	52.32	52.24	52.16
	3%	52.14	52.28	52.31	52.29	52.27	52.21

B Watermarked Model Performance

Table 8 presents the OD performance of the watermarked models. For each poisoning configuration, the performance is averaged over 30 models trained with different seeds.

References

1. https://openai.com/policies/business-terms
2. https://github.com/giddyyupp/coco-minitrain
3. https://cocodataset.org/#home
4. Adi, Y., Baum, C., Cisse, M., Pinkas, B., Keshet, J.: Turning your weakness into a strength: Watermarking deep neural networks by backdooring. In: USENIX Security Symposium. p. 1615–1631 (2018)
5. Car, S.: Self-driving cars dataset (jun 2023), https://universe.roboflow.com/selfdriving-car-qtywx/self-driving-cars-lfjou, visited on 2024-08-01
6. Chan, S.H., Dong, Y., Zhu, J., Zhang, X., Zhou, J.: Baddet: Backdoor attacks on object detection. In: Computer Vision – ECCV 2022 Workshops. pp. 396–412. Springer Nature Switzerland, Cham (2023)
7. Chen, K., Lou, X., Xu, G., Li, J., Zhang, T.: Clean-image backdoor: Attacking multi-label models with poisoned labels only. In: ICLR (2023)
8. Cheng, Y., Hu, W., Cheng, M.: Attacking by aligning: Clean-label backdoor attacks on object detection. arXiv:2307.10487 (2023)
9. Ester, M., Kriegel, H.P., Sander, J., Xu, X.: A density-based algorithm for discovering clusters in large spatial databases with noise. In: KDD. p. 226–231 (1996)
10. Everingham, M., Van Gool, L., Williams, C.K.I., Winn, J., Zisserman, A.: The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results. http://www.pascal-network.org/challenges/VOC/voc2007/workshop/index.html
11. Goldwasser, S., Kim, M.P., Vaikuntanathan, V., Zamir, O.: Planting undetectable backdoors in machine learning models. In: FOCS. pp. 931–942 (2022)
12. Jia, H., Choquette-Choo, C.A., Chandrasekaran, V., Papernot, N.: Entangled watermarks as a defense against model extraction. In: USENIX Security Symposium. pp. 1937–1954 (2021)

13. Li, Y., Lyu, X., Koren, N., Lyu, L., Li, B., Ma, X.: Neural attention distillation: Erasing backdoor triggers from deep neural networks. In: ICLR (2021)
14. Li, Y., Zhu, L., Jia, X., Jiang, Y., Xia, S.T., Cao, X.: Defending against model stealing via verifying embedded external features. In: AAAI (2022)
15. Liu, K., Dolan-Gavitt, B., Garg, S.: Fine-pruning: Defending against backdooring attacks on deep neural networks. In: RAID. pp. 273–294 (2018)
16. Luo, C., Li, Y., Jiang, Y., Xia, S.T.: Untargeted backdoor attack against object detection. In: ICASSP. pp. 1–5 (2023)
17. Ma, H., Li, Y., Gao, Y., Abuadbba, A., Zhang, Z., Fu, A., Kim, H., Al-Sarawi, S.F., Surya, N., Abbott, D.: Dangerous cloaking: Natural trigger based backdoor attacks on object detectors in the physical world. arXiv:2201.08619 (2022)
18. Ma, H., Li, Y., Gao, Y., Zhang, Z., Abuadbba, A., Fu, A., Al-Sarawi, S.F., Nepal, S., Abbott, D.: Transcab: Transferable clean-annotation backdoor to object detection with natural trigger in real-world. In: SRDS. pp. 82–92 (2023)
19. Orekondy, T., Schiele, B., Fritz, M.: Prediction poisoning: Towards defenses against dnn model stealing attacks. arXiv:1906.10908 (2019)
20. Ren, S., He, K., Girshick, R., Sun, J.: Faster r-cnn: Towards real-time object detection with region proposal networks. NeurIPS **28** (2015)
21. Sha, Z., He, X., Berrang, P., Humbert, M., Zhang, Y.: Fine-tuning is all you need to mitigate backdoor attacks. arXiv:2212.09067 (2022)
22. Snarski, A., Dimon, W.L., Manville, K., Krumdick, M.: Watermarking for data provenance in object detection. In: AIPR. pp. 1–7 (2022)
23. Szyller, S., Atli, B.G., Marchal, S., Asokan, N.: Dawn: Dynamic adversarial watermarking of neural networks. In: ACM MM. p. 4417–4425 (2021)
24. Tan, M., Le, Q.: Efficientnet: Rethinking model scaling for convolutional neural networks. In: ICML. pp. 6105–6114 (2019)
25. Tramèr, F., Zhang, F., Juels, A., Reiter, M.K., Ristenpart, T.: Stealing machine learning models via prediction apis. In: USENIX Security Symposium. p. 601–618 (2016)
26. Uchida, Y., Nagai, Y., Sakazawa, S., Satoh, S.: Embedding watermarks into deep neural networks. In: ICMR. p. 269–277 (2017)
27. Ultralytics: `https://github.com/ultralytics/ultralytics`
28. Xiao, Q., Chen, Y., Shen, C., Chen, Y., Li, K.: Seeing is not believing: Camouflage attacks on image scaling algorithms. In: USENIX Security Symposium. pp. 443–460 (2019)
29. Zhang, J., Gu, Z., Jang, J., Wu, H., Stoecklin, M.P., Huang, H., Molloy, I.: Protecting intellectual property of deep neural networks with watermarking. In: AsiaCCS. p. 159–172 (2018)
30. Zhao, Y., Lv, W., Xu, S., Wei, J., Wang, G., Dang, Q., Liu, Y., Chen, J.: Detrs beat yolos on real-time object detection. In: CVPR. pp. 16965–16974 (2024)