

## Appendix

### A Non-Maximum Suppression (NMS)

Non-Maximum Suppression (NMS) is a post-processing step commonly used in object detection pipelines to prune redundant bounding boxes. During inference, a detection model often outputs multiple overlapping bounding boxes for the same object—each with its own location, size, and confidence score. NMS serves to suppress (i.e., remove) boxes whose overlap with a higher-confidence box is too large (above a predefined IoU threshold), thereby retaining only the most representative candidates.

#### A.1 Background on NMS

Non-Maximum Suppression (NMS) is a post-processing step commonly used in object detection pipelines to prune redundant bounding boxes. During inference, a detection model often outputs multiple overlapping candidate bounding boxes for the same object—each with its own location, size, and confidence score. NMS takes all candidate bounding boxes and filters out the low-confidence ones, and boxes that have large IoU with higher-confidence boxes. NMS outputs the most representative candidate bounding boxes (also called *proposals*).

We would like to bring attention to two important parameters: the maximum number of bounding box proposals (set by the model), and the maximum detection (set by NMS). In the YOLO model family, the former parameter has been reduced with every version. In fact, YOLOv5 caps the number of proposals to 25,000, while YOLOv8 caps it to 8,000. PyTorch NMS implementation sets the maximum number of bounding box proposals to 30,000. As we will discuss in Section 4, this has a huge impact on the attack success. The maximum detection set by NMS defines the maximum number of proposals to output. This means that even if the attack forces the model to feed more bounding box proposals than what NMS accepts, it won't necessarily affect the latency.

In the standard implementation of NMS (Table 10), the maximum detection is set to 300. This parameter directly limits the effect of the fake bounding boxes on the downstream tasks (e.g., multi-object tracking Wang et al. (2024)). However, the four state-of-the-art NMS latency attacks (described in Section 2) consider a maximum detection of 30,000 which is not realistic in most use cases (e.g., autonomous driving, surveillance, or traffic monitoring).

#### A.2 Parameters

In Table 10, we describe the parameters of NMS used in the hardware evaluation.

Parameter	Description	Value
conf_thres	The confidence threshold below which boxes will be filtered out	0.25
iou_thres	The IoU threshold below which boxes will be filtered out during NMS	0.45
classes	A list of class indices to consider. If None, all classes will be considered	None
agnostic	If True, the model is agnostic to the number of classes. And all classes will be considered as one.	False
multi_label	If True, each box may have multiple labels.	False
labels	The list contains the apriori labels for a given image.	()
maximum detection	The maximum number of boxes to keep after NMS	300
nc	The number of classes output by the model. Any indices after this will be considered masks.	0
max_time_img	The maximum time (seconds) for processing one image.	0.05
max_nms	The maximum number of boxes into torchvision.ops.nms().	30000
max_wh	The maximum box width and height in pixels.	7680
in_place	If True, the input prediction tensor will be modified in place.	True
rotated	If Oriented Bounding Boxes (OBB) are being passed for NMS.	False
end2end	If the model doesn't require NMS.	False

Table 10: Parameters of Non-Maximum Suppression by Ultralytics

### 720 A.3 NMS processing time

721 Figure 10 depicts the line of code (328) of the function named `non_max_suppression` located  
722 in `ultralytics.utils.ops.py` from the library *Ultralytics* containing the inference time for  
723 the NMS processing. For our hardware evaluation, we captured `(time.time()-t)` into a vari-  
724 able name `time_nms` which was returned by `non_max_suppression` in addition to the output of  
725 `non_max_suppression`. For those who wish to reproduce our approach, we use the version 8.3.34  
726 of *Ultralytics*.

```
327         output[xi] = x[i]
328         if (time.time() - t) > time_limit:
329             LOGGER.warning(f"WARNING ⚠️ NMS time limit {time_limit:.3f}s exceeded")
330             break # time limit exceeded
331
332     return output
```

Figure 4: Location of inference time for NMS within the *Ultralytics* library

## 727 B Commands used for evaluation

728 All our commands are based on the documentation for the *Ultralytics* library.

### 729 B.1 Format

730 to export our model to a different format, we use the mode `export` available with the model of  
731 our choice. Then, we specify the model format (e.g., ONNX) required for the export such as  
732 `model.export(format="onnx")`.

### 733 B.2 Quantization

734 To quantize our model, we need to export a model in a specific format (like in the previous section).  
735 Then, we need to enable the int8 quantization during the model export. For instance, if we want  
736 a quantized int8 version of *OpenVINO*, then we just need to run the following command such as  
737 `model.export(format="openvino", int8=True, nms=True)`. The `nms` argument include the  
738 NMS post-processing in addition to the quantized model during the export process.

## 739 C NMS Latency Attacks

### 740 C.1 Prior Art

741 Previous work on latency attacks exploits the observation that generating numerous bounding boxes  
742 resistant to NMS elimination. There is an extensive literature on NMS Latency Attacks targeting  
743 Object Detection such as backdoor attack [Xiao et al. \(2024\)](#), bit-flip attack [Sistla et al. \(2025\)](#),  
744 and adversarial inputs attack. However, the first two categories rely on strong assumptions. For  
745 backdoor attacks, those assumptions include having access to the training dataset (which is not  
746 publicly disclose) or being able to inject sufficient malicious data into the training dataset (which  
747 implies the attacker knows which data will be picked to train the model). For bit-flip attack, the  
748 attacker somehow has access to the weights of the model used in production and can tamper the  
749 weights. In general, model weights are not easily accessible because they are private and there are  
750 plenty of security mechanisms to monitor any change made to the model such as cryptographic  
751 signature, encryption for data obfuscation, access control for people accessing the model weights.  
752 With such privileges, an attacker can have better impact by using a malware attack or by leaking the  
753 model weights to the public or to competitors. Adversarial inputs attack, on the other hand, is based  
754 on more realistic assumptions such as tampering inputs at inference by probing a model in a product  
755 or by using a surrogate model that may be the same as the one in production.

756 For adversarial inputs attack, There are five attacks which focus on Camera Object Detection:  
757 Daedalus, Overload, Phantom Sponge, Beyond Phantom Sponge, and Detstorm.

758 Daedalus (D) [Wang et al. (2021)] pioneered latency attacks, using adversarial examples against NMS  
 759 to achieve image-wide perturbation.

760 Overload [11] (O) [Chen et al. (2024)], improved on DS by proposing a spatial attention mechanism,  
 761 that prioritizes bounding box creation in less occupied areas to reduce overlap.

762 Phantom Sponge (PS) [Shapira et al. (2023)], advances latency attacks using universal adversarial  
 763 perturbations (UAP) to apply a single pattern across multiple images, removing generation time.

764 Beyond Phantom Sponges (BPS) [Schoof et al. (2024)] is a spin-off of the original Phantom Sponge  
 765 Attack. BPS replaces the bounding box area loss from PS with a new loss (named *IoU loss*). This  
 766 new loss function decreases the IoU of a given bounding box and every other bounding boxes instead  
 767 of just decreasing the bounding box area.

768 DetStorm (DS) [Muller et al. (2025)] is another spin-off of the original Phantom Sponge Attack. DS  
 769 modifies the original PS attack to make it work as a physical adversarial example (real-world). Since  
 770 Detstorm is based on PS, it has the same flaws as Phantom Sponge.

## 771 C.2 Evaluation Settings

772 Each latency attack relies on a set of parameters and values that are common to all attacks but also  
 773 specific to an attack (see Table 11). Here is a list of all the parameters with their description.

774 **maximum iterations** is the maximum number of iterations allowed to optimize the adversarial noise  
 775 to be added to the image.

776 **initial constant** ( $c_0$ ) is the value of the constant ( $c$ ) at initialization when generating the Daedalus  
 777 attack. The constant ( $c$ ) helps to balance the distortion and the adversarial loss function related to the  
 778 attack.

779 **binary search steps** is the maximum number of search steps used during the binary search to find the  
 780 best value for the constant  $c$ .

781 **confidence** ( $\gamma$ ) is the minimal threshold value for the confidence of adversarial bounding boxes.

782 **grid size** signifies is the size of the grids used to divide the image for the spatial attention mecha-  
 783 nism [Chen et al. (2024)].

784 **epsilon** is the maximum size of the perturbation.

785 **lambda 1** is a weighting factor affecting the *maximum object loss* and the *maximum IoU loss* [Shapira  
 786 et al. (2023); Schoof et al. (2024)].

787 **lambda 2** is a weighting factor affecting the *bounding box area loss* [Shapira et al. (2023); Schoof  
 788 et al. (2024)].

Parameter	D	O	PS	BPS
max. iterations	1000			
initial constant ( $c_0$ )	2	NA	NA	NA
binary search steps	5	NA	NA	NA
confidence	0.3	NA	NA	NA
grid size	NA	10x10	NA	NA
$\epsilon$	NA	15	70	
$\lambda_1$	NA	NA	1	
$\lambda_2$	NA	NA	10	

Table 11: Attacks’ parameters and values used in our paper. “N/A” means “Not Applicable”.

## 789 D Generating box proposals for NMS

790 For the hardware evaluation, we use Algorithm 1 to generate non-overlapping bounding boxes for a  
 791 given image size ( $W \times H$ ). The motivation to create such a script is because existing NMS attacks  
 792 do not always For instance, if we use a  $640 \times 640$  images, then we will generate 409600 boxes which  
 793 is way above the maximum number of bounding box proposals that can be handled by Ultralytics  
 794 NMS. Therefore, we make sure to cap our total number of proposals with the parameter  $M$ . By

795 default,  $M$  is set to 30000 because it is the maximum number of proposals allowed to be processed  
 796 by Ultralytics’ NMS. Now, if we want to choose the number of proposals being processed by the  
 797 NMS function, then we are using the parameter  $N$ . For instance, if the NMS must process 5000  
 798 proposals, then  $N$  equals 5000. The remaining 25000 proposals will have their confidence value set  
 799 to 0 and thus, they will be filtered out as part of the NMS pre-processing.

---

**Algorithm 1** Generate non-overlapping box proposals

---

**Require:**  $W, H$ : List

```

1:  $\mathcal{B} \leftarrow \emptyset$  // Set of bounding boxes proposals
2:  $M = 30000$  // Max. proposals for PyTorch’s NMS
3:  $w = 1$  // Width of the box
4:  $h = 1$  // Height of the box
5:  $n = 0$  //number of proposals
6:  $l = 1$  //identifier of the object class for this proposal
7: while  $n < M$  do
8:   for  $h = 0 \rightarrow H$  do
9:     for  $w = 0 \rightarrow W$  do
10:       $n = n + 1$ 
11:       $x = h + 0.5$  //Box’s center (x) coordinate
12:       $y = w + 0.5$  //Box’s center (y) coordinate
13:      if  $n < N$  then
14:        // Wanted number of proposals for NMS
15:         $c = 1$ 
16:      else
17:         $c = 0$ 
18:      end if
19:       $b = [x, y, w, h, c, l]$  // Define proposal box
20:       $\mathcal{B} \leftarrow b$  //Append the  $b$  to  $\mathcal{B}$ 
21:    end for
22:  end for
23: end while
24: return  $\mathcal{B}$ 

```

---

## 800 E Additional Details regarding the evaluation

### 801 E.1 Hardware: measuring inference time

802 To measure the inference time (in milliseconds), we use the time computed in the Ultralytics NMS  
 803 code (see Appendix [A.3](#)). Because existing attacks couldn’t generate the maximum bounding box  
 804 proposals of 30,000 and we need to evaluate the worst case scenario, we created a python script to  
 805 generate a tensor of non-overlapping 30,000 bounding box proposals, where 30,000 is the maximum  
 806 number of proposals allowed by the NMS function in the Ultralytics library. Then, for each proposal,  
 807 we set its confidence score and its class probability to 1. For the NMS function, we use the default  
 808 value of each parameter (see Appendix [A.2](#) for details).

809 To measure the inference time of NMS for a specific number of bounding boxes ( $N$ ), we set the  
 810 confidence score to 1 for  $N$  proposals out of the 30,000 proposals. The remaining proposals (30,000  
 811 minus  $N$ ) have their confidence score set to 0. Therefore, thanks to the confidence threshold, there  
 812 will be  $N$  bounding boxes after NMS. In this study, the value of  $N$  ranges from 1 to 30,000. Also,  
 813 it is important to know that models may have a threshold in terms of number of proposals such as  
 814 YOLOv5 (25,000) and YOLOv8 (8,000). This factor may prevent a latency attack to succeed due to a  
 815 potentially insufficient number of bounding boxes proposals passed to the NMS.

### 816 E.2 Hardware: determining the success of an attack

817 We define a successful NMS attack as an attack capable to increase the processing time of the NMS  
 818 above a defined threshold. In this paper, we set this threshold to 100 ms, 500 ms, 1000 ms, and 2000  
 819 ms. The value of each threshold serves as an indicator of the performance of the attack for a specific

use case. For instance, a maximum latency above 100 milliseconds is unsafe in the autonomous driving domain [Lin et al. (2018)]. In a different context, a 100 ms delay will remain unnoticed by someone using an AI model for editing pictures on his phone or on his computer. However, a 2 seconds delay would be noticeable. Going back to Table 2, we observe a NMS attack will not work if the NMS processing is performed by a GPU instead of a CPU. Therefore, the evaluation shows that NMS latency attacks are harmless when NMS runs on GPU, which is becoming best practice.

Another factor impacting the success of the attack is the maximum number of proposal output by the AI model. As discussed in Appendix A.1 and shown in Table 2, YOLOv8 caps the number of proposals to 8,000. Hence, the attack can only generate sub-500 ms latency, considering it successful only for applications with latency threshold of 100-400 ms. For YOLOv5, which has a cap of 25,000 proposals, NMS attacks are capable to reach the 2 seconds threshold.

## F Prior Art on defenses

In this paper, *PDM-Pure* is our active defense. *PDM-Pure* is an off-the-shelf adversarial purifier based on a pixel diffusion model (PDM). *PDM-Pure* aims to effectively eliminate adversarial patterns generated by latent diffusion models, thereby maintaining the integrity of images.

Our passive defense is *Maximum detection threshold*, a defense advertised in the *Overload* paper [Chen et al. (2024)]. However, the impact of this defense on *Overload* has not been tested despite being already implemented as a parameter of the NMS component used by all YOLO models [Jocher et al. (2023)].

For our natural defense, we choose *JPEG*, a popular image compression algorithm, used in many use cases. For instance, an image may need to be compressed if the image needs to transit through a network with limited data bandwidth. This scenario can occur in embedded system such as within an autonomous driving car or in collective perception where a system shares images to an external party. During those use cases, compression is necessary.