Can't Slow Me Down: Learning Robust and Hardware-Adaptive Object Detectors against Latency Attacks for Edge Devices

Supplementary Material

A. More examples of NMS and Latency Attacks

During the training phase, object detectors such as YOLO [24, 33] and Faster-RCNN [26] usually apply a many-to-one matching strategy, that the prediction results contain multiple detection boxes for the same object with redundancy. The NMS module removes this redundancy by reducing the number of detection boxes to balance the precision and recall. As shown in Fig. 9, the model predicts a number of boxes to detect the object of cat. The box number is related to the hyperparameters such as the number of anchors. The initial confidence filtering removes the most irrelevant background bounding boxes.

The primary goal of latency attacks is to ensure that the confidence scores of most bounding boxes exceed the confidence threshold. Unlike another type of attack that solely targets model efficiency [4, 10], latency attacks on object detectors not only reduces the processing speed, but also the detection accuracy. Therefore, its defense carries greater practical value.

B. When DETR meets Latency Attacks

DETR (Detection Transformer) utilizes the Hungarian algorithm for one-to-one matching between predicted boxes and ground truth boxes, enforcing a strict limit on the number of detection boxes (e.g., 100 boxes) [2]. Intuitively, DETR should be agnostic to the number of objects. In terms of robustness to perturbations, the previous works also suggest that transformers such as ViT exhibit much higher robustness against gradient-based PGD attacks compared to CNN models [1].

These have collectively led to the following question: *whether the DETR families have the similar latency attack surface as the CNN-based object detectors?* To answer this question, we analyze the performance variations with the number of instances ranging from $[0, 100]$. We also investigate the latest advance from RT-DETR (Real-Time Detection Transformer) [46] under the pressure of latency attacks.

We select images with a single instance from three categories as the candidates, placing each image into an $N \times N$ grid to generate images with N^2 instances. We employ DETR and RT-DETR to perform inference on these images 100 times and examine the average execution time on Nvidia 4070Ti Super and Jetson Orin NX in Fig 10.

The preliminary experimental results show that execution time does not vary significantly with different number of instances. From an architectural design perspective, the stability is because DETR predicts all objects from end-to-end without an additional hand-craft NMS module for redundant box elimination. Therefore, it is tempting to conclude that, as long as the matching threshold/number of boxes has been set under the hardware capacity, DETRs do not expose the same vulnerability to latency attacks as their CNN counterparts.

C. More Details of Experimental Settings

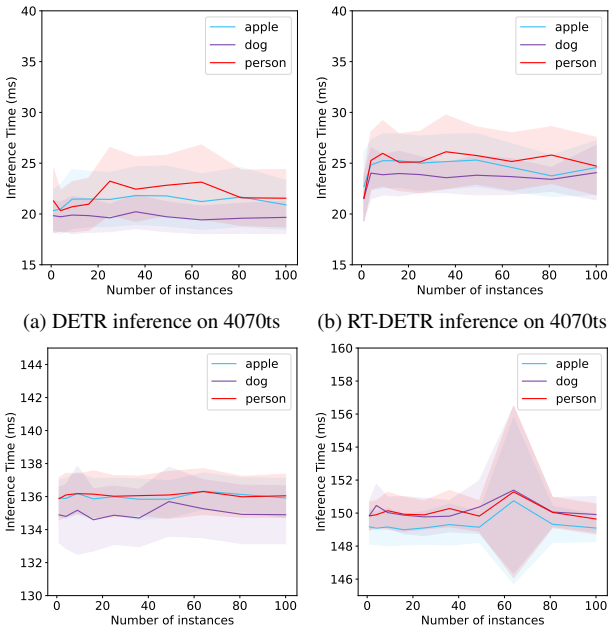
We perform all the AT on a workstation equipped with 8 Nvidia RTX 4090 GPUs, Intel Xeon Gold 6326 CPUs and 480 GB of RAM. The Python environment used for training and validation is configured with Python 3.9, PyTorch 2.0.0, Torchvision 0.15.0, and CUDA version 11.7. The edge device utilized Python 3.8, PyTorch 2.0.0, Torchvision 0.15.0, and CUDA version 11.4. The AT implementation for YOLOv3 and YOLOv5 uses the Ultralytics-YOLOv5 interface [33], while YOLOv8 adopts the Ultralytics interface [34]. Hyperparameter selection are described in the main text of the paper, and the experiments are repeated for 5 times.

D. Analysis of Training Process

As described before, the AT process initiates from the pretrained model available via the Ultralytics Github site[1]. We observe the training and validation losses of $\mathcal{L}_{\text{box}}, \mathcal{L}_{\text{obj}}, \mathcal{L}_{\text{cls}}$ with the mAP50/95 in Fig.11. We can see that the loss of the proposed Underload is less than the other two AT methods of MTD [43] and OOD [13] on $\mathcal{L}_{\text{box}}, \mathcal{L}_{\text{cls}}$ except $\mathcal{L}_{\text{obj}}$ because Underload employs objectness loss as an adversarial proxy. Injecting adversarial perturbations in the inner optimization against the objectness loss makes the outer minimization more difficult to learn with a larger training loss (but it is below the OOD loss). In the last two columns, the mean average precision of Underload is much higher than both of MTD and OOD from the beginning. The Precision/Recall curves in Fig. 12 reveal that the AT methods do not impact any specific category. However, because Underload considers the trade-off between clean and robust accuracy, the accuracy drop is mild and even negligible for some categories (e.g., the accuracy for the bicycle category only drops 0.001, compared to the drop of 0.082 and 0.095 for MTD and OOD).

[1] https://github.com/ultralytics/yolov5

Figure 9. Visualization of the entire NMS process: a) original image; 2) pre-processed results with all the prediction boxes; 3) box filtering with confidence threshold; 4) additional filtering with IoU threshold; 5) Final detection result.



(a) DETR inference on 4070ts (b) RT-DETR inference on 4070ts

(c) DETR inference on Orin NX (d) RT-DETR inference on Orin NX

Figure 10. DETR and RT-DETR inference time evaluation on different devices.

l_2-norm	Attack	Standard	MTD	OOD	**Underload***
10	Daedalus	**73.0**	57.6	55.8	68.7
	Phantom	**72.1**	57.7	55.7	68.7
	Overload	**71.8**	57.7	55.6	68.6
20	Daedalus	**70.9**	57.5	55.6	68.4
	Phantom	**70.2**	57.4	55.6	68.3
	Overload	66.1	57.2	55.5	**68.6**
30	Daedalus	66.6	57.2	55.3	**67.9**
	Phantom	64.2	57.4	55.8	**67.6**
	Overload	51.2	57.0	55.7	**68.4**
50	Daedalus	41.1	55.0	53.5	**62.0**
	Phantom	32.9	56.2	54.7	**65.4**
	Overload	17.1	54.0	54.8	**59.3**
70	Daedalus	12.8	50.1	48.4	**50.3**
	Phantom	7.5	54.7	53.3	**61.3**
	Overload	4.5	49.4	49.1	**53.3**

Table 2. Variation of the attack strengths in terms of l_2-norm for YOLOv5s. The **bold** l_2-norm is the default parameter used in the main text. The **Best** and Second Best values in each row are marked in **Red** and Blue.

E. Hyperparameter Tuning

We provide additional experiments of two key hyperparameters: *attack strength*, measured by the l_2-norm, and *balance between precision/recall*, measured by the IoU threshold Ω_{nms}.

E.1. Attack Strength

To assess attack strength, we select the maximum l_2-norm in latency attack [3, 28, 35] for Underload to ensure that our defense remains effective under challenging conditions. As shown in Table 2, the l_2-norm is set between $[10, 70]$. We find that at lower strengths when l_2-norm equals to 10 and 20, latency attacks have minimal impact on the accuracy of both standard (unprotected) and robust models (Underload AT), since the attack strength is insufficient to push a background region across the boundary margin to generate phantom objects.

However, when l_2-norm reaches 30, it starts to affect the accuracy of the standard model. When the l_2-norm exceeds 50, the accuracy of the unprotected model declines sharply to single digits when l_2-norm increases to 70. On the other hand, we observe that the robust accuracy maintains above the 60% mAP level for most of the attack strength and is still above 50% under the maximum l_2-norm of 70. In comparison to MTD and OOD, our approach achieves an accuracy of 0.2% to 10.7% improvements under different attack strengths.

E.2. Balance between Precision/Recall

The Intersection over Union (IoU) threshold is an important parameter that balances the precision and recall in object detection. A higher IoU threshold during the NMS process retains fewer candidate boxes, which reduces the occurrence of false positives and enhances the model's precision. However, setting the IoU threshold too high may inadvertently remove some true positives, thereby reducing the recall. Conversely, reducing the IoU threshold can enhance recall by keeping more candidate boxes, but it also leads to

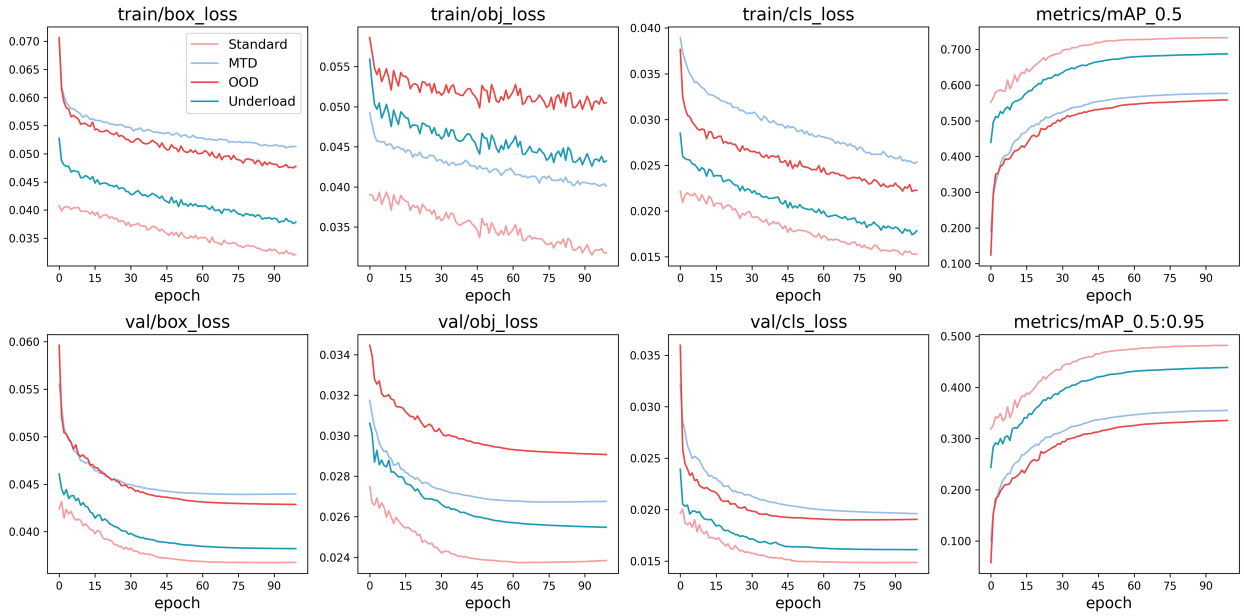


Figure 11. Visualization of the training process of YOLOv5s on the PASCAL-VOC dataset.

Ω_{nms}	Attack	Standard	MTD	OOD	**Underload***
0.30	Clean	**72.7**	58.1	57.4	68.8
	Daedalus	12.7	**50.4**	49.5	49.9
	Phantom	7.2	54.8	54.7	**61.2**
	Overload	4.1	49.1	50.3	**52.6**
0.45	Clean	**73.6**	58.7	57.8	69.5
	Daedalus	13.0	**50.9**	50.0	50.5
	Phantom	7.4	55.6	55.1	**61.9**
	Overload	4.3	49.7	50.6	**53.4**
0.60	Clean	**73.3**	57.7	55.9	68.9
	Daedalus	12.8	50.1	48.4	**50.3**
	Phantom	7.5	54.7	53.3	**61.3**
	Overload	4.5	49.4	49.1	**53.3**
0.75	Clean	**71.4**	53.7	50.7	65.7
	Daedalus	12.1	46.6	43.7	**47.5**
	Phantom	7.4	51.1	48.5	**58.6**
	Overload	4.4	46.6	45.1	**51.3**
0.9	Clean	**62.5**	39.6	35.1	52.7
	Daedalus	12.0	33.9	29.5	**36.9**
	Phantom	6.3	37.6	33.7	**46.5**
	Overload	3.7	35.0	31.6	**41.1**

Table 3. Variations of the IoU threshold Ω_{nms} in YOLOv5s. The **bolded** Ω_{nms} is the default parameter used in the main text. The **Best** and Second Best values in each row are marked in **red** and blue. The first row of "Clean" compares the clean accuracy drop with different AT methods under different Ω_{nms}.

an increase in overlapping detection, which ultimately decreases the precision.

Under latency attacks, the IoU threshold can be adjusted to simulate various attack scenarios. Setting the IoU threshold to 1.0 keeps all the candidate boxes (no box is removed), thereby retaining the artifacts produced by the latency attack that emulates the worst-case scenario. We evaluate five IoU thresholds ranging from 0.3 to 0.9, with an increment of 0.15, covering a wide range of IoU threshold values. We observe that both the clean and robust accuracy of the standard and robust models exhibit an initial increase followed by a decrease as the IoU threshold increases. Among the selected IoU thresholds, the highest accuracy occurs at an IoU threshold of 0.45. At this threshold, the reduction of Underload in clean accuracy is minimal, with a decrease of only 4.1%. In most cases, the robust accuracy of the Underload outperforms the other two AT methods. However, there are a few outliers in the low IoU cases (when the IoU threshold is 0.3 or 0.45), the robust accuracy of the MTD exceeds that of the Underload. We conjecture that it is due to the Daedalus method, which simultaneously optimizes the confidence and size of the phantom objects, may generate some high-confidence and large-area phantoms (compared with other latency attacks). When the IoU threshold is low, these phantoms can interfere with natural objects, leading to a reduction in the robust accuracy.

F. Framework and Hardware Optimization under Latency Attacks

In addition to PyTorch implementation, we also convert YOLOv5s and YOLOv8s to other implementations including ONNX and TensorRT for specialized acceleration. For

Model	Device	Attack	Standard		**Underload***	
			ONNX	TensorRT	ONNX	TensorRT
YOLOv5s	1650Ti Laptop	Clean	19.4	14.1	19.0	14.2
		`Overload`	69.2	64.9	18.7	14.0
	4070Ti Super	Clean	7.4	6.3	7.2	6.2
		`Overload`	31.1	28.6	7.0	6.3
	Jetson Orin NX	Clean	30.3	19.2	30.1	20.0
		`Overload`	100.8	87.7	29.8	19.9
	Jetson Xavier NX	Clean	57.7	30.8	57.6	30.0
		`Overload`	447.2	418.5	56.0	31.0
YOLOv8s	1650Ti Laptop	Clean	23.1	13.8	23.0	14.0
		`Overload`	25.3	16.0	23.1	13.9
	4070Ti Super	Clean	8.3	2.9	8.4	3.0
		`Overload`	9.4	4.0	8.3	2.8
	Jetson Orin NX	Clean	30.1	18.2	30.0	18.0
		`Overload`	33.2	21.7	30.3	18.1
	Jetson Xavier NX	Clean	58.1	42.2	58.0	40.0
		`Overload`	59.8	45.7	57.6	40.3

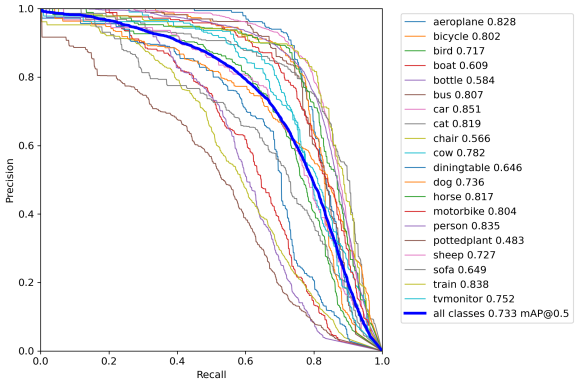
Table 4. Different frameworks of YOLOv5s and YOLOv8s model inference time (ms) in FP32.

Package	Desktop Ver.	Edge Device Ver.
CUDA	11.7	11.4
Ultralytics	8.2.100	8.2.100
ONNX	1.14.0	1.17.0
ONNXRuntime	1.16.0	1.16.0
TensorRT	8.6.0	8.5.2

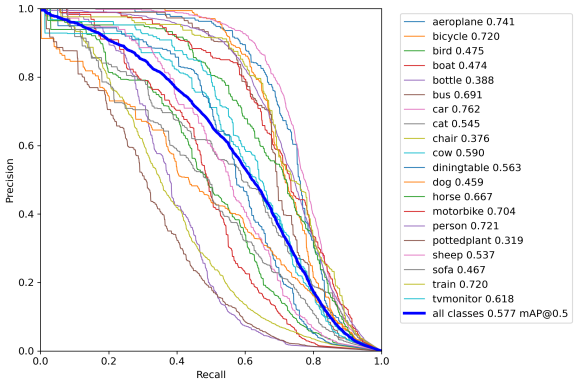
Table 5. ONNX and TensorRT models inference environments.

reproducing purposes, the environment setup is shown in the Table 5. We employ the same method to attack the implementations of ONNX and TensorRT. In details, we export ONNX and TensorRT models using the official implementation, which convert only the backbone without utilizing the `INMSLayer` or `EfficientNMSPlugin`. Other configurations remain the same as Sec. 5.

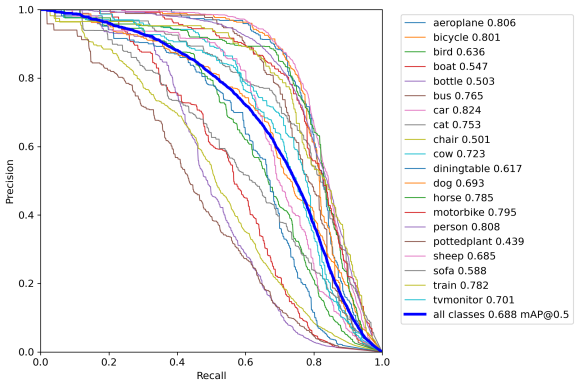
From Table 4, we find that latency attacks affect models across different implementations even for TensorRT. Despite of hardware-specific optimizations, the execution time still increases by $1.1 - 13.5\times$ under the `Overload` attack. In TensorRT, `EfficientNMSPlugin` is also vulnerable because as per to our evaluation, its execution time increases with the number of candidate boxes. Fortunately, the AT models exported from our `Underload` defense is able to defend against the corresponding latency attacks and portable to different frameworks and edge devices.



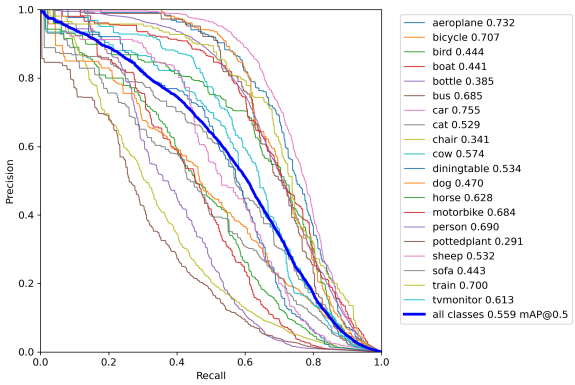
(a) Standard



(b) MTD



(c) Underload



(d) OOD

Figure 12. Precision/Recall curves for different AT methods.