Investigating Stealthy Latency Attacks Against Camera-Based Object Detection Systems

Lichen Xia, Xing Gao, Weisong Shi
Department of Computer and Information Sciences
University of Delaware, Newark, DE 19716, USA
{lxia, xgao, weisong}@udel.edu

Abstract

Object detection (OD) systems are fundamental to safety-critical applications such as autonomous driving and indoor mobile robots, yet remain vulnerable to latency-based attacks that degrade real-time performance. Existing attacks suffer from computational inefficiency and poor stealthiness, as phantom object generation disrupts original detections, making attacks easily identifiable. We propose Groundswell, a novel latency attack employing Regional Perturbation Balance (RPB) to inject adversarial phantom objects while preserving original detections. By leveraging spatial object distribution patterns, RPB generates masks that guide phantom placement into regions with minimal interference, enabling efficient optimization via Projected Gradient Descent. Evaluations across multiple OD architectures demonstrate that Groundswell reduces resource utilization by 50% and training time by 80% over existing methods, while successfully inducing latency delays that exceed real-time thresholds for safety-critical systems.

1 Introduction

Deep learning-based object detection (OD) systems, including the YOLO series [11], have achieved outstanding detection precision and real-time efficiency, making them essential in safety-critical domains such as autonomous vehicles (AVs) where real-time perception is crucial. Modern OD architectures including Faster R-CNN [21], SSD [16], and YOLO [9] rely on Non-Maximum Suppression (NMS) as a post-processing step to remove redundant predictions.

Adversarial attacks deceive machine learning systems through carefully crafted input perturbations, and their security implications have been extensively studied [4, 23, 27]. However, these traditional attacks primarily target model integrity by causing misclassifications or missed detections. Recent work has proposed latency-based attacks against OD system availability [17,19,24,28], focusing on increasing computational execution time rather than compromising model integrity. Current methods [3, 15, 19, 22] amplify detected object quantities through phantom object generation, exploiting the positive correlation between NMS processing time and detection count. Such attacks pose considerable dangers to time-critical applications, particularly Level 4 and Level 5 autonomous driving systems that require maintaining perception output generation under 30 milliseconds [5].

Unfortunately, existing latency attacks suffer from critical limitations hindering practical deployment. First, phantom objects can disrupt original detections, making the attack easily noticeable. Second, approaches that attempt to preserve original detections introduce additional loss functions requiring pairwise computation between detected objects, incurring significant computational overhead.

We introduce Groundswell, a latency attack employing a novel framework called Regional Perturbation Balance (RPB) to preserve original detection performance while maximizing latency impact. Exploiting the observation that objects tend to concentrate in specific spatial regions across sensor inputs, Groundswell crafts a universal adversarial perturbation (UAP) via Projected Gradient Descent (PGD). RPB learns these spatial distribution patterns to generate a mask identifying suitable regions for phantom object insertion, maximizing NMS computational overhead while minimizing interference with legitimate detections.

We evaluate Groundswell across varying configurations and diverse OD architectures. The results demonstrate a strong balance between effectiveness and stealthiness, with 50% less resource utilization during training and 80% faster optimization than existing work, making it significantly more practical for real-world deployment.

Our main contributions are outlined as follows:

- We propose a novel latency attack with RPB, which directs phantom object placement into image regions with minimal impact on original detections, achieving a favorable trade-off between attack effectiveness and stealthiness while improving optimization efficiency.
- We perform extensive experiments across diverse config-

urations and target object detection architectures, demonstrating our methodology substantially improves optimization efficiency compared to existing approaches.

- We establish the effectiveness of our approach across various OD architectures, highlighting its potential as a significant threat.

2 Background

2.1 Perception Pipeline

A typical perception pipeline consists of object detection and Non-Maximum Suppression (NMS). Object detection extends beyond image classification by simultaneously localizing and recognizing multiple objects, with deep learning approaches falling into two categories: two-stage detectors (e.g., Faster R-CNN [21], Mask R-CNN [7]) that separate region proposal and classification, and one-stage detectors (e.g., SSD [16], YOLO [9]) that unify both tasks in a single network. The pipeline operates in three stages: (1) preprocessing, where sensor inputs are resized, normalized, and augmented; (2) inference, where the CNN generates bounding boxes with class probabilities; and (3) post-processing, where NMS eliminates redundant overlapping boxes, retaining only the highest-confidence detection per object.

2.2 Non-maximum Suppression

NMS is a critical post-processing component in modern OD systems. A recent survey shows that 11 out of 12 OD systems employ NMS for refining their results [14]. Since detection models generate numerous candidate bounding boxes, each containing position, dimensions, objectness confidence, and class probabilities, NMS consolidates overlapping predictions corresponding to the same physical object.

The execution of the most common NMS algorithm (i.e., vanilla NMS [10]) can be decomposed into three distinct stages. During the initial stage, NMS processes the candidate set C, which is the output of the second phase of OD models, and eliminates entries with confidence scores falling below the threshold τ_{conf}:

$$C_1 = \{c_{obj} > \tau_{conf} | c \in C\} \tag{1}$$

where $c \in C$ is a single candidate included in the candidate set C, and the objectness score c_{obj} represents the model's confidence in the presence of an object.

Subsequently, candidates with insufficient class confidence are also eliminated:

$$C_2 = \{c_{obj} \cdot \max\{c_{cls,i}\}_{i=0}^{N} > \tau_{conf} | c \in C_1\} \tag{2}$$

where the class confidence score $c_{cls,i}$ represents the classification certainty for each class i from 0 to N. The class with the highest class confidence score is assigned as the class label for the candidate c.

In the second phase, filtered candidates C_2 are sorted by confidence score c_{obj}, then offset by class to group detections into separate coordinate spaces. This prevents cross-class detections from being incorrectly merged.

Finally, starting from the highest-confidence detection, NMS iteratively computes the Intersection over Union (IoU) between each box and all remaining candidates. Boxes of the same class exceeding a specified IoU threshold (e.g., $\tau_{IoU} = 0.5$) are suppressed as redundant, retaining only the highest-confidence detection per object. This continues until no redundant detections remain. In the worst case where no candidates are removed in phase one, the time complexity of the NMS reaches $O(|C|^2)$.

The strong dependency of NMS's execution time on the number of detections makes it vulnerable for latency-based attacks against object detection systems.

3 Latency attacks and limitations

Sponge examples [24] first revealed that vision systems could be manipulated to consume excessive computational resources. Subsequent latency attacks on OD systems targeted preprocessing [28] and postprocessing [15, 17] stages. For NMS-targeted attacks, Daedalus [26] introduced phantom objects to increase execution latency, but produced non-transferable patches, limiting practicality. Shapira et al. [22] proposed universal adversarial patches that flood NMS with fake detections while minimizing overlap with original detections, but at high computational cost during optimization. Chen et al. [3] improved efficiency by inserting synthetic objects in sparse image regions, though still requiring per-image patch generation. Muller et al. [19] extended digital attacks to physical settings via zone stitching, but without stealthiness mechanisms. Our work addresses these limitations with a framework that efficiently optimizes attack effectiveness and stealthiness for practical real-world deployment.

4 Threat Model

Following prior work [3, 19, 22], we assume a white-box attacker with full knowledge of the target OD model. When the victim runs a publicly documented stack (e.g., Autoware with YOLOv5), the attacker can simply download the same public weights without any privileged access. The proliferation of model-sharing platforms further reinforces this assumption: Hugging Face alone hosts over two million public models, making it straightforward for any attacker to obtain the exact weights of widely deployed perception models [8]. As shown in Figure 1, the optimized UAP can be delivered physically by affixing it as a sticker on the camera lens [4] or by projecting it onto the attack surface in front of the victim vehicle using a

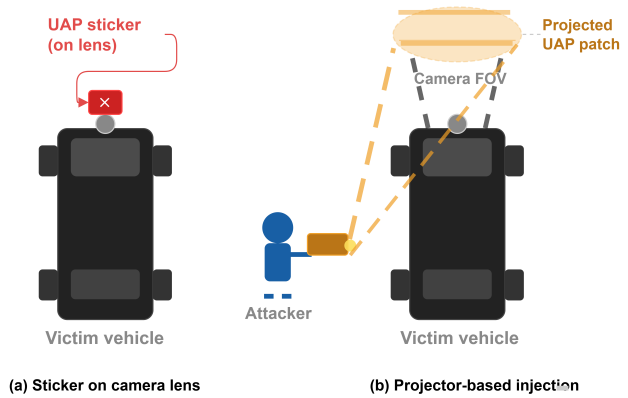


Figure 1: To effect perturbations, an attacker can either (a) affix a UAP sticker directly over the onboard camera lens [4], or (b) project the perturbation onto the surface in front of the victim vehicle using a portable projector [19].

portable projector [19]. Both methods inject the perturbation into raw frames before any in-vehicle processing occurs, requiring neither software access nor network interaction. The sticker approach is passive and persistent. The attacker only needs a single physical access to the vehicle and is not required to be present during the attack. The projector approach instead requires the attacker to maintain proximity to the victim, but allows real-time adjustment of the perturbation as driving conditions change.

The primary objective of the attacker is to identify a UAP δ that can be applied to input sequences (X) in real-time in order to maximize the total processing time (T_{OD}) of the target OD model. Additionally, to maintain attack stealthiness, the attacker aims to preserve the detection outputs from the original clean images. This dual objective can be formulated as the following optimization problem:

$$\max_{\|\delta\|_p \leq \varepsilon} \sum_{x \in X} T_{\text{OD}}(x+\delta) + \lambda_{stealth}(D(f(x+\delta), f(x)) \quad (3)$$

where the perturbation δ is bounded by an ε-ball under a given norm p. Larger ε values produce more visible perturbations, compromising attack stealthiness. Here, f represents the target OD model, while D measures the preservation of original detections. The parameter $\lambda_{stealth}$ controls the trade-off between maximizing latency and maintaining stealthiness of the attack.

5 Groundswell Attack

We propose the Groundswell attack that aims to maximize the attack effectiveness while preserving stealthiness by identifying image regions suitable for adversarial object insertion with minimal impact on original detections. The proposed RPB framework learns spatial distribution patterns of target sensors to generate masks that preserve original predictions. By combining with PGD [18], our method can generate an optimized UAP to enhance attack effectiveness while maintaining stealthiness.

5.1 Maximize the latency

As analyzed in Section 2.2, the NMS's third phase has the highest time complexity of $O(n^2)$, where n is the number of processed candidates. Thus, to induce latency, we attempt to generate a large number of phantom objects that require processing by the third phase of the NMS. However, many candidates may be discarded during the first phase of the NMS before reaching the third stage. To avoid being filtered out by the NMS's first phase, the injected objects must maintain confidence scores higher than the configured threshold τ_{conf} of the targeted OD models. Let C denote the candidate set output by the target OD model with unperturbed images, and C' denote the candidate set when inputs contain the UAP δ. For an individual proposal $c' \in C'$, the loss ℓ is defined as:

$$\ell(c', k) = max(0, \tau_{conf} - c'_{obj} \cdot c'_{cls,k}) \quad (4)$$

where k is the target class to which the adversary requires the proposal c' belongs. ℓ aims to minimize the gap between the confidence threshold τ_{conf} and the product of c'_{obj} and $c'_{cls,k}$. It encourages the proposal c' to be detected as a valid object with label k. If c' already qualifies as a valid object, the loss function applies no modification.

The overall loss function for maximizing the number of objects across an input image is formulated as:

$$\ell_{max\ obj} = \frac{1}{|C'|} \sum_{c' \in C'} \ell(c', k) \quad (5)$$

Bounding Box Area Loss. Minimizing the size of phantom object candidates can free up space for insertions, potentially enabling more qualified candidates to reach NMS processing. [22]. Thus, we further include a bounding box area loss:

$$\ell_{box\ area} = \frac{1}{|C'|} \sum_{c' \in C'} c'_w \cdot c'_h \quad (6)$$

where c'_w and c'_h are the width and height of the candidate c'. To craft the UAP, we minimize a loss function combining two components into a single equation:

$$\ell = \ell_{max\ obj} + \lambda \ell_{box\ area} \quad (7)$$

where λ is the weight of bounding box area loss $\ell_{box\ area}$.

5.2 Regional Perturbation Balance (RPB)

The secondary goal is attack stealthiness: the perturbed image x' should preserve as many original detections from clean

image x as possible. We find that images from specific sensors can exhibit certain patterns. For instance, for images collected by AV cameras, objects are more likely to accumulate in the central area of the images [2, 6]. The upper areas tend to be sky, while the bottom areas are typically roads. Consequently, perturbations applied to the image periphery have less impact on stealthiness than those in the central region, which we validate in Section 6.2.

Based on this observation, we propose RPB to efficiently balance attack effectiveness and stealthiness. An image is divided into three regions (upper, middle, bottom) by boundaries b_{upper} and b_{lower}, with perturbations applied only to the upper and bottom regions, where original detections (e.g., vehicles) are sparse.

The boundaries are dynamically adjusted via Algorithm 1 based on T_{ratio}, the target percentage of original objects the adversary wishes to preserve. At each optimization round, the ratio $r_{detection}$ of objects in the middle region to total detections in the clean image is computed (*line 5*). If $r_{detection} > T_{ratio}$, the middle region shrinks to allow more perturbation (*line 6-8*); if $r_{detection} < T_{ratio}$, it expands to preserve more original detections (*line 9-11*). The perturbation mask is then updated accordingly with the new boundaries (*line 14-16*).

Algorithm 1 Adaptive Boundary Update for Adversarial Patches

1: **procedure** SINGLE ROUND UPDATE
2: $T_{ratio} \leftarrow$ Target detection ratio
3: $C \leftarrow$ Set of candidates produced by the clean image
4: $s \leftarrow$ boundary adjustment step size
5: $r_{detection} \leftarrow detection\_ratio(C, b_{upper}, b_{lower})$
6: **if** $r_{detection} > T_{ratio}$ **then** ▷ Too many detections in middle region
7: $b_{upper} \leftarrow b_{upper} + s$
8: $b_{lower} \leftarrow b_{lower} - s$
9: **else** ▷ Too few detections in middle region
10: $b_{upper} \leftarrow b_{upper} - s$
11: $b_{lower} \leftarrow b_{lower} + s$
12: **end if**
13: $*, height, width \leftarrow image.shape$
14: $mask \leftarrow$ zeros_matrix$(image.shape)$
15: $mask[0 : b_{upper}, *] \leftarrow 1$ ▷ Upper region for perturbation
16: $mask[b_{lower} : height, *] \leftarrow 1$ ▷ Bottom region for perturbation
17: **return** $mask, b_{upper}, b_{lower}$
18: **end procedure**

6 Evaluation

To evaluate Groundswell attack, we conduct comprehensive experiments across multiple YOLO detector versions (YOLOv5 [12], YOLOv8 [25], YOLOv11 [11]) pretrained on MS-COCO [13]. Our evaluation enables direct comparison with existing work (Phantom Sponges [22], Overload [3], DETSTORM [19]) while assessing transferability to advanced architectures.

6.1 Experimental setup

Dataset and Models We utilize the Berkeley Deep Drive (BDD) dataset [29], comprising 100,000 autonomous driving images with diverse environmental conditions, including varied weather (clear, rainy), settings (city streets, residential areas), and lighting conditions (daytime, nighttime). YOLOv5 serves as our primary model for baseline comparisons, while YOLOv8 and YOLOv11 assess attack performance on more advanced architectures. All models process images at 640×640 resolution using small-sized variants (YOLOv5s, YOLOv8s, YOLOv11s) for computational efficiency.

Attack Parameters. Perturbations are bounded using L_2 norm with $\varepsilon = 70$ across all evaluated attacks. Detection thresholds τ_{conf} and τ_{IoU} are set to 0.25 and 0.45, respectively, matching YOLO detectors' default settings. The target class is fixed to 2 with $\lambda = 10$ in equation 7, following the best practices from Phantom Sponges [22].

Implementation Detail. UAP training employs PGD with a learning rate of 0.0005, conducted over 100 epochs using a batch size of 8. All experiments run on NVIDIA RTX 3090 hardware.

Metrics. Our attack targets end-to-end latency in object detection models while maintaining stealthiness, which we evaluate using three metrics consistent with previous works [3, 19, 22]. These include:

1. $M' = |C_2|$: representing the number of candidates fed into the third phase of NMS. The more candidates NMS processes, the longer it should take.
2. Time: inference timing measurements comprising total end-to-end time (T_{total}[ms]) and NMS execution time (T_{NMS}[ms]). This metric measures the latency introduced to the NMS and the overall OD model, providing a direct evaluation of the attack's effectiveness.
3. Recall [%]: measuring the ratio of original objects still detectable in perturbed images relative to clean images. A higher recall indicates a higher attack stealthiness.

6.2 Verifying the observation supporting RPB.

To validate the observation supporting RPB, we analyze per-image object distributions in the BDD dataset. Images are divided into three horizontal regions, with the middle region gradually expanded from 10% to 90% of the image area, and the ratio of middle-region objects to total objects recorded across 2,000 randomly sampled test images. As shown in Figure 2, when the middle region occupies just 30% of the image,

it already contains over 71% of all objects. This confirms that objects in AV camera inputs tend to concentrate in specific spatial regions, supporting the core assumption of RPB.

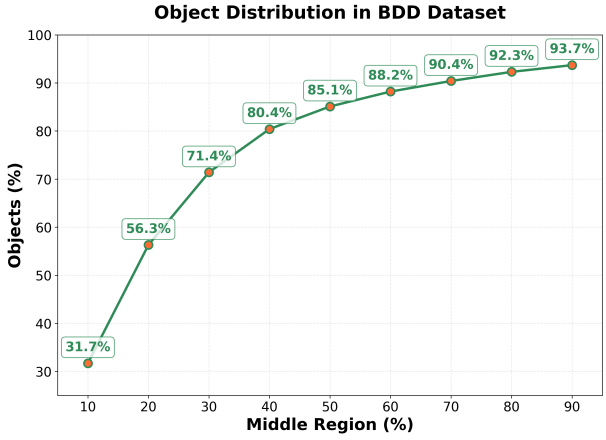


Figure 2: Object distributions of the images in the BDD dataset.

6.3 Attack Evaluation

Attack Performance. Table 1 presents the experimental results when our attack is applied to YOLOv5s, YOLOv8s, and YOLOv11s (marked by the grey background). We also evaluate various configurations (T_{ratio}) of the RBP. On YOLOv5s, compared to clean images (images without perturbation), our attack results in a more than 300-fold increase in the number of candidates fed into the third phase of NMS (M') with the most aggressive settings ($T_{ratio} = 0$). The overall execution time of YOLOv5s is increased by approximately 250% accordingly. However, the recall drops from 100% to 18%, making the attack easy to identify. When the T_{ratio} of RBP is set to 0.4, the recall doubles with slight sacrifices in M' and the introduced latency. When configured with a conservative setting ($T_{ratio} = 0.8$), our attack manages to push the recall above 64% while still retaining most of the introduced phantom objects.

When applied to YOLOv8s and YOLOv11s, our attack with the RBP framework also demonstrates good performance, dramatically increasing the stealthiness of the attack (recall) without significantly compromising its effectiveness (M' and T_{total}).

RBP Evaluation. Figure 3 shows how the RBP influences attack effectiveness and stealthiness with various configurations. As T_{ratio} (from 0.0 to 0.8) increases, the size of the middle region (i.e., the image area that cannot be perturbed) grows gradually. We denote the proportion of the middle region to the entire image area as R_{middle}. Consequently, there is less area available for adding phantom objects, leading to a decrease in M' as well as T_{total}. Since fewer phantom objects are added to the images, more original detections are preserved from the clean images.

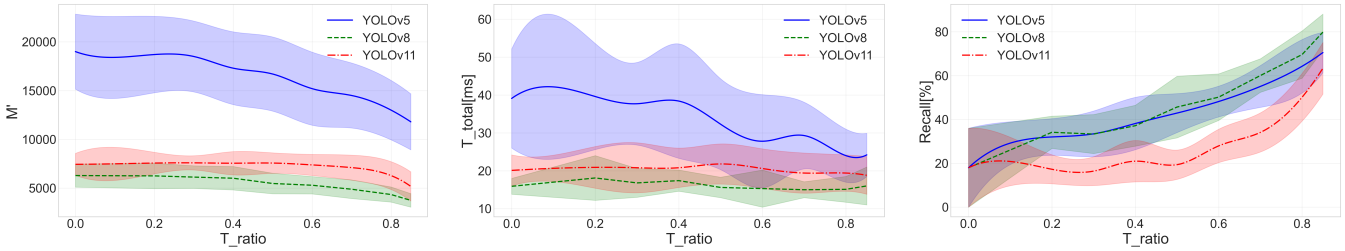
Table 1: Experimental results of the Groundswell attack compared to previous methods on YOLOv5s, YOLOv8s, and YOLOv11s. Gswell(T_{ratio}) refers to Groundswell attack with the T_{ratio} configuration shown in brackets. PSs($\ell_{max\ IoU}$) refers to the Phantom Sponge attack with the $\ell_{max\ IoU}$ setting shown in brackets.

YOLOv5s				
Attack	M'	T_{total}	T_{NMS}	Recall
Clean	101	15.2	1.7	100
Gswell(0)	19000	41.4	30.7	18.1
Gswell(0.4)	17300	38.4	23.8	38.2
Gswell(0.8)	13000	24.0	11.6	64.3
PSs(0)	18900	39.8	26.2	18
PSs(0.2)	15500	29.3	16.9	45.6
PSs(0.3)	13300	24.9	12.8	62.9
Overload	15400	31.0	19.1	4.5
DETSTORM	19800	41.1	28.5	2.2

YOLOv8s				
Attack	M'	T_{total}	T_{NMS}	Recall
Clean	52	12.8	1.5	100
Gswell(0.0)	6400	18.2	5.2	21.2
Gswell(0.4)	6100	17.7	4.9	37.2
Gswell(0.8)	4400	14.2	3.7	69.6
PSs(0)	6400	18.8	5.7	20.8
PSs(0.2)	5400	14.2	4.5	50.1
PSs(0.3)	4900	14.5	4.0	59.2
Overload	7200	18.3	6.7	4.5
DETSTORM	5700	16.8	4.5	4.0

YOLOv11s				
Attack	M'	T_{total}	T_{NMS}	Recall
Clean	50	15.3	1.6	100
Gswell(0.0)	7500	20.9	6.4	8.7
Gswell(0.4)	7500	20.8	6.0	21.0
Gswell(0.8)	6200	18.7	4.8	50.1
PSs(0)	7600	20.3	6.8	8.4
PSs(0.2)	6200	18.5	4.9	46.3
PSs(0.3)	5700	16.8	3.0	53.4
Overload	7600	20.1	6.5	2.1
DETSTORM	7800	20.8	6.9	0.9

Figure 4 shows how R_{middle}, guided by RBP, evolves during the training process. The R_{middle} of RPB with $T_{ratio} = 0.2$ converges to 0.1 within 25 iterations. Similarly, the R_{middle} of RPB with $T_{ratio} = 0.8$ converges to 0.4 with some fluctuation. The evolution of the RPB boundaries follows the observation in Figure 2: when the middle region occupies 10% of the whole image ($R_{middle} = 0.1$), it contains more than 20% of the original objects ($T_{ratio} = 0.2$). When the middle region



(a) M' vs T_{ratio} (b) T_{total} vs T_{ratio} (c) Recall vs T_{ratio}

Figure 3: Latency attack with different RPB configurations.

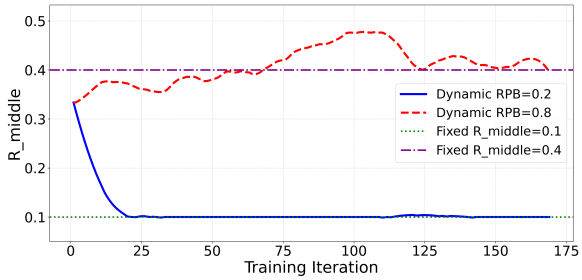


Figure 4: Evolution of the R_{middle} guided by RPB during the attack optimization.

occupies 40% of the whole image ($R_{middle} = 0.4$), more than 80% of the original objects are located within it ($T_{ratio} = 0.8$).

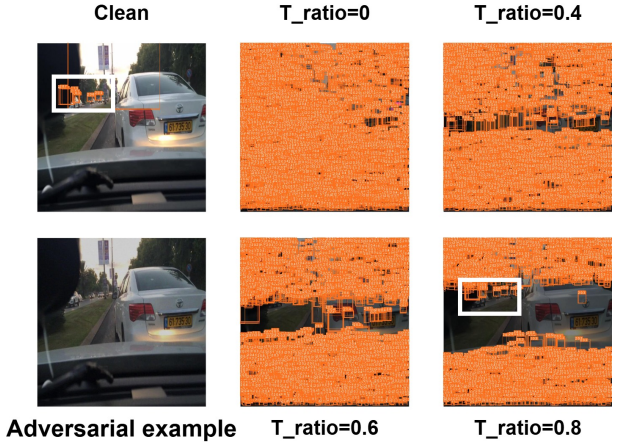


Figure 5: Visualization of Groundswell attack with different configurations of RPB when applied to a clean image.

Attack Visualization. Figure 5 visualizes how different configurations of RPB affect the attack. As shown, the crafted adversarial example, generated by applying the trained UAP and mask to a clean image, is virtually indistinguishable from the original image to the human eye. We further adjust target stealthiness level T_{ratio}. As T_{ratio} increases from 0 to 0.8, the area uncovered by the perturbation becomes larger, and progressively more original detections are revealed. When T_{ratio} is set to 0.8, most of the original detections (marked by white boxes) are preserved even when our attack is deployed.

Evaluation on different GPUs. We evaluate the Groundswell attack on different GPUs. The RPB framework is configured with T_{ratio} values of 0.2, 0.5, and 0.8, respectively. The evaluation results are presented in Table 2. As shown, on both the RTX 3080 and RTX 4080, our Groundswell attack introduces significant computational overhead to YOLOv5s. The results demonstrate that our method is effective across various GPU platforms.

6.4 Comparison to SOTA Approaches.

We compare our attack to three existing latency attacks (Phantom Sponges [22], Overload [3], and DETSTORM [19]) on YOLOv5s, YOLOv8s, and YOLOv11s.

The Phantom Sponges attack includes a third loss function $\ell_{max\ IoU}$ with an associated weight $\lambda_{max\ IoU}$ for adjusting attack stealthiness. Different configurations of the Phantom Sponges regarding $\ell_{max\ IoU}$ are presented for comprehensive comparison. Overload was initially designed for a different scenario. It generates distinct perturbations for individual images; we adapted its open-source code [20] to generate UAPs across multiple images and fine-tuned the training process for comprehensive comparison.

As shown in Table 1, Overload exhibits the worst performance in both attack effectiveness and stealthiness since it was not designed for training UAPs. DETSTORM has similar performance to the most aggressive configuration of our approach since it does not take attack stealthiness into consideration during attack optimization. Compared to Phantom Sponges, the biggest advantage of our approach lies in the efficiency of attack optimization. Table 3 compares our Groundswell attack with the RBP framework and the Phantom Sponges attack with $\ell_{max\ IoU}$ in terms of CPU utilization and execution time per epoch during optimization. Our approach reduces CPU utilization by half and optimization time by 80%. The loss function $\ell_{max\ IoU}$ employed by Phantom Sponges tracks the IoU between each pair of detected objects, which becomes costly when the number of detected objects is maximized.

Table 2: Performance evaluation of our Groundswell attack on different hardware with various configurations of RPB. T_{clean} and T_{attack} denote the end-to-end execution times of YOLOv5s for clean and adversarial images, respectively. Increase[%] shows the percentage increase in execution time from adversarial vs. clean images.

		$T_{ratio} = 0.2$			$T_{ratio} = 0.5$			$T_{ratio} = 0.8$		
GPU	T_{clean}	T_{attack}	Increase[%]	Recall	T_{attack}	Increase[%]	Recall	T_{attack}	Increase[%]	Recall
RTX3080	10.5	46.4	341.9	32	42.6	305.7	43	29.3	179.0	64
RTX4080	8.6	39.6	360.5	32	35.5	312.8	43	19.2	123.3	64

Table 3: UAP generation cost comparison between our Groundswell attack (Gswell) and the Phantom Sponges attack (PSs)

Method	CPU(%)	Time(s)
Gswell	48.0	∼50
PSs	96.0	∼250

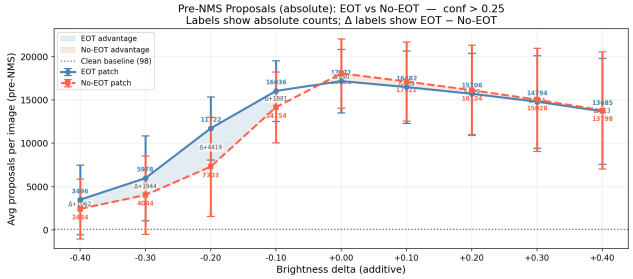


Figure 6: Pre-NMS proposal counts (M') per image at $\tau_{\text{conf}} = 0.25$ for the EOT and No-EOT patches across additive brightness deltas δ. Shaded region marks EOT advantage; Δ labels show EOT − No-EOT at each level; error bars denote standard deviation.

6.5 Robustness Under Environmental Factors

Real-world deployment of adversarial patches is subject to uncontrolled environmental variation (e.g., changing illumination, weather, and sensor noise). Expectation Over Transformation (EOT) [1] addresses this by averaging gradients over a distribution of transformations during training, producing a perturbation that is adversarially effective in expectation across the target distribution rather than at a single operating point. To demonstrate the impact of EOT on Groundswell, we use brightness variation as a representative environmental factor and evaluate how both patches hold up as ambient illumination shifts away from the neutral training condition.

Setup. We train two YOLOv5s patches under identical conditions (Section 6.1, $T_{\text{ratio}} = 0.3, \varepsilon = 70$, 100 epochs): a standard Groundswell patch trained at neutral brightness (No-EOT), and an EOT-augmented patch whose PGD gradient at each step is averaged over n=8 brightness samples drawn uniformly from $\delta \in (-0.3, +0.3)$ (EOT). Both patches are evaluated on the BDD100K validation split under a fixed sweep $\delta \in \{-0.40, \ldots, +0.40\}$, measuring M' at $\tau_{\text{conf}} = 0.25$.

Results. Figure 6 reveals three distinct conditions. In dark conditions ($\delta < -0.10$), EOT substantially outperforms the baseline. At $\delta = -0.20$, the EOT patch generates 9555 proposals per image versus 6132 for No-EOT (+55.8%). This confirms that EOT builds robustness to brightness suppression that a neutrally trained patch cannot acquire. At ($\delta \approx 0.00$), No-EOT holds a negligible edge (15505 vs. 14412, −7.0%). The baseline concentrates its entire gradient budget at this operating point, so the marginal advantage is expected and small. Above $\delta = +0.10$, both patches converge to near-identical counts as pixels saturate at the $[0, 1]$ boundary.

7 Discussion

While Groundswell demonstrates strong performance, two limitations warrant discussion. First, RPB relies on spatial object distribution patterns learned from BDD100K, so cameras mounted at significantly different heights or pitch angles could misalign the perturbation mask with actual sparse regions. Since mounting positions are fixed post-installation and follow consistent conventions across vehicle classes, a practical mitigation is to maintain a small offline dictionary of UAPs per mounting configuration. Second, while BDD100K's diverse weather and lighting conditions provide implicit environmental robustness, dedicated analysis remains future work; EOT [1] can be incorporated to optimize perturbations over distributions of noise, brightness, and contrast shifts. We conduct a preliminary evaluation, and the results confirm that EOT introduces extra robustness to the optimized UAP under various brightness conditions.

8 Conclusion

In this paper, we present Groundswell, a novel latency-based attack against deep learning-powered object detection systems that addresses critical limitations of existing attack methodologies. Our experimental results demonstrate that the Groundswell attack successfully reduces computational resource requirements by 50% and decreases optimization

time by 80%, while maintaining comparable or superior attack effectiveness. These efficiency gains make our approach particularly practical for real-world deployment scenarios.

References

[1] Anish Athalye, Logan Engstrom, Andrew Ilyas, and Kevin Kwok. Synthesizing robust adversarial examples. In International conference on machine learning, pages 284–293. PMLR, 2018.

[2] Manuel Carranza-García, Pedro Lara-Benítez, Jorge García-Gutiérrez, and José C Riquelme. Enhancing object detection for autonomous driving by optimizing anchor generation and addressing class imbalance. Neurocomputing, 449:229–244, 2021.

[3] Erh-Chung Chen, Pin-Yu Chen, I Chung, Che-Rung Lee, et al. Overload: Latency attacks on object detection for edge devices. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 24716–24725, 2024.

[4] Kevin Eykholt, Ivan Evtimov, Earlence Fernandes, Bo Li, Amir Rahmati, Chaowei Xiao, Atul Prakash, Tadayoshi Kohno, and Dawn Song. Robust physical-world attacks on deep learning visual classification. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 1625–1634, 2018.

[5] GSA Forum. The challenges to achieve level 4 & level 5 autonomous driving, 2023.

[6] Abhishek Gupta, Kandasamy Illanko, and Xavier Fernando. Object detection for connected and autonomous vehicles using cnn with attention mechanism. In 2022 IEEE 95th Vehicular Technology Conference:(VTC2022-Spring), pages 1–6. IEEE, 2022.

[7] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In Proceedings of the IEEE international conference on computer vision, pages 2961–2969, 2017.

[8] Hugging Face. State of open source on Hugging Face: Spring 2026. https://huggingface.co/blog/huggingface/state-of-os-hf-spring-2026, March 2026. Accessed: 2026-05-01.

[9] Glenn Jocher, Ayush Chaurasia, and Jing Qiu. Ultralytics: Yolo. https://github.com/ultralytics/ultralytics, 2023. Accessed: April 10, 2025.

[10] Glenn Jocher et al. Ultralytics yolo documentation. https://docs.ultralytics.com/, 2023. Accessed: May 13, 2025.

[11] Glenn Jocher, Jing Qiu, and Ayush Chaurasia. Ultralytics YOLO, January 2023.

[12] Glenn Jocher, Alex Stoken, Jirka Borovec, NanoCode012, ChristopherSTAN, Liu Changyu, Laughing, tkianai, Adam Hogan, lorenzomammana, yxNONG, AlexWang1900, Laurentiu Diaconu, Marc, wanghaoyang0106, ml5ah, Doug, Francisco Ingham, Frederik, Guilhen, Hatovix, Jake Poznanski, Jiacong Fang, Lijun Yu, changyu98, Mingyu Wang, Naman Gupta, Osama Akhtar, PetrDvoracek, and Prashant Rai. ultralytics/yolov5: v3.1 - bug fixes and performance improvements, October 2020.

[13] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence 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, pages 740–755. Springer, 2014.

[14] Chongwei Liu, Haojie Li, Shuchang Wang, Ming Zhu, Dong Wang, Xin Fan, and Zhihui Wang. A dataset and benchmark of underwater object detection for robot picking. In 2021 IEEE international conference on multimedia & expo workshops (ICMEW), pages 1–6. IEEE, 2021.

[15] Han Liu, Yuhao Wu, Zhiyuan Yu, Yevgeniy Vorobeychik, and Ning Zhang. Slowlidar: Increasing the latency of lidar-based detection using adversarial examples. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 5146–5155, 2023.

[16] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander 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, pages 21–37. Springer, 2016.

[17] Chen Ma, Ningfei Wang, Qi Alfred Chen, and Chao Shen. Slowtrack: Increasing the latency of camera-based perception in autonomous driving using adversarial examples. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 38, pages 4062–4070, 2024.

[18] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. arXiv preprint arXiv:1706.06083, 2017.

[19] Raymond Muller, Ruoyu Song, Chenyi Wang, Yuxia Zhan, Jean-Phillipe Monteuuis, Yanmao Man, Ming Li, Ryan Gerdes, Jonathan Petit, and Z Berkay Celik. Investigating physical latency attacks against camera-based perception. In 2025 IEEE Symposium on Security and Privacy (SP), pages 4588–4605. IEEE, 2025.

[20] Maria-Irina Nicolae, Mathieu Sinn, Minh-Ngoc Tran, Beat Buesser, Ambrish Rawat, Martin Wistuba, Valentina Zantedeschi, Nathalie Baracaldo, Bryant Chen, Heiko Ludwig, Ian Molloy, and Ben Edwards. Adversarial robustness toolbox (art) - python library for machine learning security. https://github.com/Trusted-AI/adversarial-robustness-toolbox, 2018.

[21] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. Advances in neural information processing systems, 28, 2015.

[22] Avishag Shapira, Alon Zolfi, Luca Demetrio, Battista Biggio, and Asaf Shabtai. Phantom sponges: Exploiting non-maximum suppression to attack deep object detectors. In Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision, pages 4571–4580, 2023.

[23] Mahmood Sharif, Sruti Bhagavatula, Lujo Bauer, and Michael K Reiter. A general framework for adversarial examples with objectives. ACM Transactions on Privacy and Security (TOPS), 22(3):1–30, 2019.

[24] Ilia Shumailov, Yiren Zhao, Daniel Bates, Nicolas Papernot, Robert Mullins, and Ross Anderson. Sponge examples: Energy-latency attacks on neural networks. In 2021 IEEE European symposium on security and privacy (EuroS&P), pages 212–231. IEEE, 2021.

[25] Rejin Varghese and M Sambath. Yolov8: A novel object detection algorithm with enhanced performance and robustness. In 2024 International Conference on Advances in Data Engineering and Intelligent Computing Systems (ADICS), pages 1–6. IEEE, 2024.

[26] Derui Wang, Chaoran Li, Sheng Wen, Qing-Long Han, Surya Nepal, Xiangyu Zhang, and Yang Xiang. Daedalus: Breaking nonmaximum suppression in object detection via adversarial examples. IEEE Transactions on Cybernetics, 52(8):7427–7440, 2021.

[27] Kaidi Xu, Gaoyuan Zhang, Sijia Liu, Quanfu Fan, Mengshu Sun, Hongge Chen, Pin-Yu Chen, Yanzhi Wang, and Xue Lin. Adversarial t-shirt! evading person detectors in a physical world. In Computer vision–ECCV 2020: 16th European conference, glasgow, UK, August 23–28, 2020, proceedings, part v 16, pages 665–681. Springer, 2020.

[28] Oryan Yehezkel, Alon Zolfi, Amit Baras, Yuval Elovici, and Asaf Shabtai. Desparsify: Adversarial attack against token sparsification mechanisms. Advances in Neural Information Processing Systems, 37:127536–127560, 2024.

[29] Fisher Yu, Haofeng Chen, Xin Wang, Wenqi Xian, Yingying Chen, Fangchen Liu, Vashisht Madhavan, and Trevor Darrell. Bdd100k: A diverse driving dataset for heterogeneous multitask learning. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pages 2636–2645, 2020.

Artifact Appendix

Investigating Stealthy Latency Attacks Against Camera-Based Object Detection Systems

.1 Abstract

This artifact contains the implementation of Groundswell, a novel latency attack against deep learning-based object detection (OD) systems using the Regional Perturbation Balance (RPB) framework. The attack generates a Universal Adversarial Perturbation (UAP) via Projected Gradient Descent (PGD) that maximizes the number of phantom objects passing through Non-Maximum Suppression (NMS), while spatially constraining perturbations to image regions with fewer original detections to preserve stealthiness. The artifact includes: the Groundswell attack implementation (`uap_RPB.py`), local YOLO model copies (`local_yolos/`), dataset utilities (`datasets/`), and a main entry point (`run_attack.py`).

The artifact supports the following major claims:

- Groundswell reduces CPU resource utilization by ∼50% and optimization time by ∼80% compared to Phantom Sponges during UAP training.
- Groundswell increases the number of NMS candidates (M') by over 300× on YOLOv5s, increasing end-to-end latency by ∼250%.
- The RPB framework provides a configurable stealthiness-effectiveness trade-off via T_{ratio}, achieving over 64% recall at T_{ratio}=0.8 on YOLOv5s.
- Results transfer across YOLOv5s, YOLOv8s, and YOLOv11s architectures.

The artifact includes: the Groundswell attack implementation (`uap_RPB.py`), local YOLO model copies (`local_yolos/`), dataset utilities (`datasets/`), and a main entry point (`run_attack.py`). Output artifacts include the trained UAP patch, per-epoch loss logs, validation images, and saved patches.

.2 Description & Requirements

.2.1 Security, Privacy, and Ethical Concerns

The attack operates entirely in the digital domain. No physical hardware, real vehicles, or live camera systems are required. All attacked models (YOLOv5s, YOLOv8s, YOLOv11s) are publicly available pretrained models on MS-COCO. No undisclosed vulnerabilities or zero-day exploits are involved. The BDD100K dataset is a publicly available autonomous driving dataset. Generated UAP patches are research artifacts not optimized for physical-world deployment.

.2.2 How to Access

Zenodo DOI: https://doi.org/10.5281/zenodo.20397573

.2.3 Badges

- **Artifacts Available** — publicly hosted on Zenodo with permanent DOI.
- **Artifacts Functional** — installs and runs to produce UAP patches and per-epoch metrics.
- **Results Reproduced** — reproduces Tables 1, 2, and 3.

.2.4 Hardware Dependencies

- **GPU:** NVIDIA GPU with CUDA 11.3 support. Paper uses RTX 3090; RTX 3080 and RTX 4080 also validated (Table 2).
- **GPU Memory:** Minimum 10 GB VRAM (`batch_size=8` on 640×640 images).
- **RAM:** Minimum 16 GB recommended.
- **Storage:** ∼20 GB free (YOLO weights + BDD100K subset + outputs).

.2.5 Software Dependencies

OS: Ubuntu 18.04 LTS. Python 3.8 exactly. CUDA 11.3. PyTorch 1.10.1+cu113, TorchVision 0.11.2+cu113 (installed separately). TensorFlow 2.13.1. Other: `numpy==1.24.3`, `opencv-python-headless==4.10.0.84`, `scikit-learn==1.3.2`, `scikit-image==0.19.3`, `scipy==1.10.1`, `pandas==2.0.3`, `matplotlib==3.7.5`, `albumentations==1.0.3`, `Pillow==8.4.0`, `seaborn==0.13.2`, `tqdm==4.67.1`, `PyYAML==6.0.2`.

.2.6 Benchmarks

Dataset: Berkeley Deep Drive (BDD100K), available at https://www.kaggle.com/datasets/solesensei/solesensei_bdd100k. Use validation set (∼190 images suffice; full 10K set yields more stable statistics). Format: images in `./val/`, YOLO-format labels in `./det_20/`.
Models: YOLOv5s, YOLOv8s, YOLOv11s pretrained on MS-COCO — weights bundled in `local_yolos/*/weights/`.

.3 Set-up

.3.1 Installation

1. Verify Python 3.8: `python3.8 -version`
2. Create virtual environment: `cd RPBAttack/ && python3.8 -m venv artifact_env && source artifact_env/bin/activate`
3. Upgrade pip: `python3 -m pip install -upgrade pip setuptools wheel`
4. Install dependencies: `python3 -m pip install -r requirements.txt`
5. Install PyTorch: `python3 -m pip install torch==1.10.1+cu113`

`torchvision==0.11.2+cu113 -extra-index-url https://download.pytorch.org/whl/cu113 -no-deps`

6. Prepare BDD100K: place images in `RPBAttack/val/` and labels in `RPBAttack/det_20/`. Run `python3 convert.py` to convert to YOLO format. Sample images/labels are included for testing.

.3.2 Basic Test

Run the following to verify all imports and CUDA availability:

```
python3 -c "import torch; import cv2; import numpy; print('All imports OK')"
python3 -c "import torch; print('CUDA:', torch.cuda.is_available())"
```

.4 Evaluation Workflow

.4.1 Major Claims

(C1): Groundswell (T_{ratio}=0) increases NMS candidates (M') from 101 to ∼19,000 and total latency from 15.2 ms to ∼41.4 ms (∼250% increase) on YOLOv5s. Recall drops to ∼18% (Table 1).

(C2): Groundswell (T_{ratio}=0.8) maintains ∼64% recall while increasing M' to ∼13,000 and T_{total} to ∼24 ms on YOLOv5s (Table 1).

(C3): Groundswell uses ∼48% CPU at ∼50 s/epoch vs. Phantom Sponges at ∼96% CPU and ∼250 s/epoch (Table 3).

.4.2 Experiments

(E1): *Main Attack Training on YOLOv5s* [30 human-minutes + ∼1.5 compute-hours + 5 GB disk]: trains Groundswell UAP on YOLOv5s to reproduce YOLOv5s rows of Tables 1 and 3 for three T_{ratio} configurations (Claims C1, C2, C3).

Preparation: Edit `run_attack.py`: set `attack_type = "RPB"`, `models_vers = [5]`, `epoch = 100`, `epsilon = 70`, `iter_eps = 0.0008`, `lambda_1 = 1`, `lambda_2 = 10`, `batch_size = 8`, and set `BDD_IMG_DIR`/`BDD_LAB_DIR` to the dataset paths. Set `threshold_ratio = 0.0` for the first run.

Execution: Run `python3 run_attack.py`. Repeat with `threshold_ratio = 0.4` and `0.8` for all YOLOv5s rows. Monitor CPU usage in a separate terminal via `top`. To reproduce the Phantom Sponges baseline (Table 3), set `attack_type = "PS"`.

Results: Compare printed `average_proposal_nums` (M'), `T_total`, `T_NMS`, and `Recall` against Table 1. Output files are written to `experiments/<patch_name>/`: `final_results/final_patch.png`, `saved_patches/` (best checkpoint), `applied_patch/` (validation images with UAP), and `losses/` (pickled loss lists).

(E2): *Transferability to YOLOv8s and YOLOv11s* [20 human-minutes + ∼3 compute-hours]: reproduces YOLOv8s and YOLOv11s rows of Table 1 (Claim C1).

Preparation: Same configuration as E1.

Execution: For YOLOv8s set `models_vers = [8]` and run `python3 run_attack.py`. For YOLOv11s set `models_vers = [11]`. YOLOv8/11 outputs are auto-converted to YOLOv5 format via `convert_yolov11_to_yolov5_format()` before NMS.

Results: Compare printed metrics against the corresponding rows of Table 1. Numeric variance of ±5–10% across GPU hardware is expected due to floating-point non-determinism.

.5 Notes on Reusability

The artifact generalizes beyond the paper experiments.

Custom datasets: replace `./val/` and `./det_20/` with any dataset in YOLO format.

Custom YOLO targets: `models_vers` accepts any combination of `[5, 8, 11]`; multi-model training selects a random model each batch.

Attack aggressiveness: `epsilon` controls the L_2 perturbation budget (default 70); `iter_eps` controls PGD step size (default 0.0008).

Stealthiness: `threshold_ratio` (T_{ratio}, range 0.0–0.8) is the key parameter: 0.0 = maximum latency, 0.8 = maximum stealthiness. Docker deployment is strongly recommended for cross-platform reproducibility.

Known limitation: CPU-only execution is not supported; CUDA-capable GPU is mandatory.

.6 Version

Based on the LaTeX template for Artifact Evaluation V20260101. Submission, reviewing and badging methodology followed for the evaluation of this artifact can be found at https://secartifacts.github.io/vehiclesec2026/.