Investigating Physical Latency Attacks against Camera-based Perception

Raymond Muller
Purdue University
mullerr@purdue.edu

Ruoyu Song
Purdue University
song464@purdue.edu

Chenyi Wang
University of Arizona
chenyiw@arizona.edu

Yuxia Zhan
New York University
yuxia.zhan@nyu.edu

Jean-Phillipe Monteuuis
Qualcomm
jmonteuu@qualcomm.com

Yanmao Man
HERE Technologies
yman@arizona.edu

Ming Li
University of Arizona
lim@arizona.edu

Ryan Gerdes
Virginia Tech
rgerdes@vt.edu

Jonathan Petit
Qualcomm
petit@qti.qualcomm.com

Z. Berkay Celik
Purdue University
zcelik@purdue.edu

***Abstract*—Camera-based perception is a central component to the visual perception of autonomous systems. Recent works have investigated *latency attacks* against perception pipelines, which can lead to a Denial-of-Service against the autonomous system. Unfortunately, these attacks lack real-world applicability, either relying on digital perturbations or requiring large, unscalable, and highly visible patches that cover up the victim's view. In this paper, we propose DETSTORM, a novel physically realizable latency attack against camera-based perception. DETSTORM uses projector perturbations to cause delays in perception by creating a large number of adversarial objects. These objects are optimized on four objectives to evade filtering by multiple Non-Maximum Suppression (NMS) approaches. To maximize the number of created objects in a dynamic physical environment, DETSTORM takes a unique greedy approach, segmenting the environment into "zones" containing distinct object classes and maximizing the number of created objects per zone. DETSTORM adapts to changes in the environment in real time, recombining perturbation patterns via our *zone stitching* process into a contiguous, physically projectable image. Evaluations in both simulated and real-world experiments show that DETSTORM causes a 506% increase in detected objects on average, delaying perception results by up to 8.1 seconds, and capable of causing physical consequences on real-world autonomous driving systems.**

1. Introduction

Object detection and tracking (ODT) are essential to perception in autonomous systems (AS), including for autonomous driving [2], [13], autonomous surveillance [21], [34], and unmanned aerial vehicles [37]. ODT allows autonomous systems to identify and track movement of objects within the environment, such that a correct planning decision can be made. For example, in autonomous vehicles, ODT may warn the system about an obstacle on the road, and the autonomous vehicle may, in return, stop or go around it.

Previous work has extensively explored modifying the results of ODT through *perception attacks*. For example, *misclassification attacks* [17] alter an object's class information to influence a system's decisions; object creation [36] creates fake objects that are perceived as real by ODT; object deletion [9] causes objects to be overlooked by ODT; and tracker-hijacking [35] alters the movement of objects over time to trigger a targeted action by a system. All of these attacks are against the *integrity* of an autonomous system, changing the output of ODT to suit an attacker's goal.

However, recent work has proposed attacks against the *availability* of autonomous systems, through *latency attacks* [11], [30], [32], [44], [49]. These attacks seek Denial-of-Service against ODT, preventing it from appropriately yielding decision-making results. They used adversarial perturbations to create a large number of artificial objects, positioned and sized to evade non-maximum suppression (NMS). NMS, positioned between object detection (OD) and object tracking (OT), seeks to filter out duplicate bounding boxes by removing lower-confidence bounding boxes that overlap with high-confidence bounding boxes. Yet, by forcing an $O(n^2)$ number of comparisons with a mass of non-overlapping, non-removable fake objects, NMS is slowed down. With delayed perception results, the previously proposed integrity defenses that require perception results to operate are unable to mitigate these kinds of attacks (Section 9).

Unfortunately, previous latency attacks are not physically feasible, as they require perturbing unalterable areas like the sky or using large patches that obscure much of the camera view, limiting scalability. They are also sensitive to environmental conditions; for instance, attacks generated with dark-colored buildings do not transfer well to bright-colored ones (Section 3.1). Further, these attacks are impractical in real-time or dynamic environments, as generating a single adversarial example can take 22–39 minutes [49]. Finally, prior attacks target only one version of NMS, overlooking other popular implementations.

In this paper, we introduce DETSTORM, the first *physical* latency attack against camera-based perception. DETSTORM increases victim latency by creating a large number of objects on *all* physically perturbable surfaces in the environment, which causes a large number of inter-object comparisons for NMS and object tracking. Using a camera to map the surrounding environment, DETSTORM first leverages depth-wise separable convolutions to separate the input camera

frame into unique "zones", each containing a distinct class (e.g., pedestrians, cars, roads). DETSTORM takes a greedy approach and maximizes the amount of created objects for each zone, as opposed to previous approaches that increased the number of objects in the entire image space. Next, DETSTORM uses a pre-generated *perturbation dictionary* which contains universal adversarial patches capable of attacking each type of perturbable zone. We create the perturbation dictionary using a new loss function that optimizes four separate objectives to allow our attack to generalize well to several NMS approaches. Finally, we create a zone stitching algorithm to adapt our attack to environmental changes in real time, combining multiple perturbations from our dictionary into a unified pattern that can be projected to launch the attack.

We evaluate our attack through simulated physical experiments on the BDD100K dataset [55] and through real-world autonomous driving experiments, measuring latency increases and object counts across five GPU models, including visual computing platforms, embedded boards, and an autonomous vehicle's hardware stack. We assess attack transferability across various object detection, NMS, and tracking methods and examine the impact of victim system settings (e.g., NMS confidence threshold), available attack surfaces, and camera-surface angles. Finally, a case study on Autoware [27] demonstrates the potential physical impact of DETSTORM on a fully autonomous driving system.

Our evaluation shows that in simulated experiments, DETSTORM increases object counts by 506%, with latencies 170% higher than prior work [44]. In real-world experiments, DETSTORM achieves latencies up to 8.1 seconds without printed patches, $259.62\times$ the maximum achieved by previous attacks [49]. We reach 100% transferability to DIoU [59] and Confluence [45] NMS, and retain 100% of created objects across single- and two-stage tracking methods. DETSTORM is effective regardless of victim-tuned parameters (Section 6.4) and evades existing defenses (Section 9). Finally, DETSTORM induces collisions in simulation experiments against Autoware [27] by introducing latencies of $0.27+$ seconds.

Our contributions can be summarized as follows:

- We introduce DETSTORM[1], the first end-to-end physical latency attack against camera-based perception.
- We formulate a novel greedy approach for latency effects in dynamic physical environments, leveraging depthwise separable convolutions to maximize object creation on all perturbable surfaces.
- We conduct the most comprehensive NMS attack evaluation to date, across a digital dataset, real-world experiments, and an autonomous driving simulator, measuring effectiveness on seven object detection models, four tracking models, and three NMS algorithms.

2. Background

As depicted in Figure 1, camera-based perception pipelines commonly include object detection, non-maximum suppression, and object tracking. First, given video input, *object detection* quickly identifies objects in each frame, creating redundant bounding boxes in the process. *NMS* then filters these to retain the highest-confidence box per object. Next *object tracking* links these bounding boxes across frames, ensuring spatio-temporal consistency and providing object tracks with velocity and unique IDs. This data, along with object classes, feeds into global planning and sensor fusion, which pass information to the control module for system actuation (e.g., throttle, brake, and steer for vehicles, alerts for surveillance).

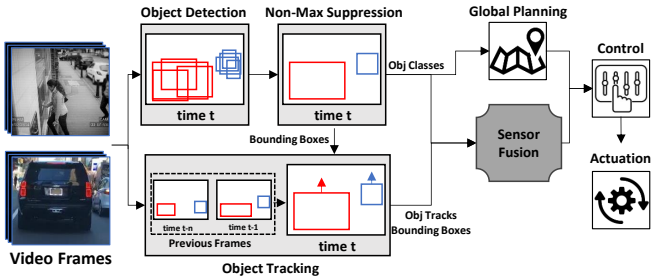


Figure 1: An overview of a camera-based perception pipeline.

Object Detection. Autonomous systems begin perception with *object detection*. Although there are many types of architectures, including the growing interest in transformer-based approaches [14], CNNs remain more commonly deployed for object localization and classification. Object detection involves three key steps. First, during *pre-processing*, resizing, normalization, and data augmentation prepare the input image. Second, during *inference*, a trained model detects and classifies objects, producing bounding boxes and class probabilities. Finally, during *post-processing*, techniques like non-maximum suppression (NMS) refine the output and remove redundant boxes, resulting in the final detected objects.

Non-Maximum Suppression (NMS). NMS is essential in object detection, with a recent survey reporting that 11 of 12 OD models rely on it to refine results [29]. NMS filters duplicate bounding boxes from OD. In the most common approach, *vanilla NMS*, this is done in three steps. First, objects are grouped together in their own coordinate spaces by class, to prevent objects of different classes from being considered as the same object. Next, bounding boxes are ordered based on their confidence scores. To save computations, objects with confidence scores below confidence threshold T_{conf} are discarded. Additionally, if the number of objects exceeds the proposal number threshold T_{prop}, only the first T_{prop} detections ordered by confidence are kept. Finally, the highest-ranked bounding box is selected, and its *Intersection over Union* (IoU) with remaining boxes is calculated.

If no boxes are removed, NMS requires up to $O(n^2)$ computations, where n is the total bounding box count. Lastly, boxes with IoU above a similarity threshold T_{Sim} (e.g., 0.5) are discarded. This process repeats until vanilla NMS outputs the highest-confidence boxes per object.

Two popular alternative NMS metrics are DIoU [59] and Confluence [45]. DIoU measures bounding box similarity via centroid distance, with box pairs under an L2 distance

1. Available at https://github.com/purseclab/DetStorm

of T_{Sim} considered the same object. It outperforms vanilla NMS by 1.22% to 3.14% in crowded scenarios. Similarly, Confluence uses a confidence-weighted Manhattan Distance to measure similarity, reducing false positives and achieving 2.5% to 3.6% higher average precision than vanilla NMS.

Object Tracking. After objects are classified, object tracking assigns unique identities to objects between frames, estimating their trajectories. Broadly, there are two types of object tracking approaches: two-stage trackers like SORT [5] use detection results for association and motion estimation, while one-stage trackers like Siamese tracking [4] track objects based on appearance. Object tracking is split into three steps. First, in *detection*, a Kalman Filter predicts object movement between frames, creating "trackers". Next, in *association*, OD boxes are matched with trackers using the Hungarian method. Finally, in *post-processing*, trackers are adjusted or removed, and velocities are assigned. Because each detected object must be associated with a tracker, object tracking, similar to NMS, requires $O(nm^2)$ comparisons, where n is the number of detected objects and m is the number of generated trackers.

3. Previous Attacks and Challenges

3.1. Latency Attacks

Previous work on latency attacks exploits the observation that generating numerous bounding boxes resistant to NMS elimination (e.g., low IoU in vanilla NMS) can slow perception processing by increasing comparisons in NMS. Daedalus [49] (DS) pioneered latency attacks, using adversarial examples against vanilla NMS to achieve extensive image-wide perturbation. Unfortunately, DS's physical attacks (attempted via printed patches) are not feasible: DS patches are non-transferable between images and require 22–39 minutes to generate, making them impractical for dynamic environments. Thus, if the environment changes in any way while the attack is being conducted (as is common in real world settings), the generated patch is no longer applicable. To combat this, DS attempts to create a large patch (e.g., 21.5 cm x 21.5 cm) positioned within 1.8 meters of the camera. However, these patches are highly visible and unsuitable for use against moving vehicles. Moreover, DS's reported maximum physical latency of 0.03 second falls well within the acceptable end-to-end latency of 0.1 second for autonomous vehicles [28].

The next attack, Overload [11] (OL), improved on DS by proposing a *spatial attention* metric, that prioritizes bounding box creation in less occupied areas to reduce overlap. Although OL generates over $24,000$ objects against YOLOv5s, it requires 3 minutes to generate a single attack sample and is only demonstrated digitally with perturbations across the entire image. Meanwhile, Phantom Sponges [44] (PS), advances latency attacks using *universal adversarial perturbations* (UAP) to apply a single pattern across multiple images, removing generation time. Given *attack source* images, PS creates a pattern that induces numerous bounding boxes in similar images, achieving a 45% increase in detection time on vanilla NMS. Unfortunately, our experiments reveal PS's effectiveness depends heavily on the source images used. As shown in Appendix Figure 13, UAPs generated from dark images are limited to dark surfaces (e.g., roads, shadows), indicating that they are not strictly universal and require adaptation to the target environment.

Table 1: Comparing DETSTORM with other latency attacks.

	DS [49]	PS [44]	OL [11]	SL [30]	ST [32]	DETSTORM
Full AS Pipeline	✗	✗	✗	✗	✓	✓
Physically Realized	✗†	✗	✗	✗	✗	✓
OT Transferability	✗	✗	✗	✗	✓	✓
NMS Transferability	✗	✗	✗	✗	✗	✓
Real-time Adaptability	✗	◐‡	✗	✗	✗	✓

† Daedalus's physical attack is not successful in creating significant latencies [28].
‡ PS's adaptability is limited by the appearance of the samples in UAP generation (Sec 3.1).

Other works include SlowLidar [30] (SL), which extends digital latency attacks to 3D LiDAR detection, and SlowTrack [32] (ST), which applies PS concepts to object tracking. ST is the first to show that in a full ODT pipeline, generating many objects to slow NMS also hinders tracking, as OT requires comparable or more comparisons than NMS. However, ST's approach requires perturbing the entire image, impractical for physical attacks where some regions (e.g., sky) cannot be perturbed. Additionally, ST generates attacks for each video frame (except the first), adding significant overhead in real-time dynamic environments.

We compare DETSTORM with previous latency attacks in Table 1. First, previous attacks are either digital or fail in real-world patch-based scenarios. DETSTORM is the first work to use projector-based physical perturbations for latency attacks. Next, unlike previous attacks, which target only vanilla NMS, DETSTORM transfers across multiple NMS algorithms, including DIoU and Confluence, without performance loss. Finally, DETSTORM is the first to adapt its perturbations in real time to changing environments.

3.2. Design Challenges

To perform *physical* latency attacks on ODT, we address four unique challenges not considered in previous works.

(C_1) **Constraints on Attack Surfaces.** Unlike digital attacks, not all surfaces in the real world can be perturbed, e.g., the sky. One approach to address this challenge is to create one's own attack surfaces, e.g., by using a monitor or a paper printout as a patch that can be displayed where the attacker wants. However, previously proposed approaches required the patch to cover most of the camera's field of view and obscure other objects [49]. This is a very strong attack assumption, as an attacker capable of obscuring the victim's field of view would be able to arbitrarily cause a desired effect (e.g., simply covering the camera to remove perception results). Additionally, such an attack would be highly visible, since it requires a patch to be either extremely large or extremely close to the victim's camera. Instead, a physical attack must be able to utilize as much of the available environment as possible, without relying on an attacker-controlled surface.

Another approach explored in previous work [35] is to map the perturbable surfaces in the environment with a LiDAR point cloud. However, this method does not capture information about the *appearance* of the mapped

surfaces, which is an important factor in the success of latency attacks. For example, dark-colored objects like roads respond differently to latency-inducing perturbations than bright-colored objects like street signs (Section 3.1). Thus, an attacker must be able to reason on not only the topology of the surfaces in the environment, but also their semblance.

(C_2) **Maximizing Latency Effects.** In the physical domain, attack surfaces are limited. Therefore, maximizing object creation on available surfaces is crucial. Unlike digital attacks such as ST and OL, which spread created objects across the entire image, physical attacks must create flexible perturbations to avoid limiting the effect to specific surfaces.

(C_3) **Competing Light Sources.** Traditionally, perturbations in the digital domain are made to be as small in magnitude as possible, to be less perceptible to a casual observer. This applies doubly in the physical domain, where realizing a greater perturbation magnitude requires a more powerful and expensive projector [33]. However, unlike in the digital domain, physical projector perturbations must compete with surrounding light sources, which means that perturbations must not be so subtle that they can be easily overwhelmed by changing lighting conditions. These two opposing factors must be balanced against each other for a successful physical attack.

(C_4) **Real-time Adaptation.** Physical environments change quickly. For example, a vehicle, pedestrian, or other object may suddenly move across the camera's field of view, altering both available attack surfaces and the appearance of the scene. Previous attacks focused on generating perturbation patterns separately as each image frame is received, a process which can take a few minutes [11] or up to half an hour [49]. Other works [32] propose the use of Expectation over Transformation [1] to convert their digital attacks to physical patch attacks, but this requires optimizing over a distribution of transformations and is computationally expensive. With physical environments that can change within the span of a second (e.g., those encountered in autonomous driving), a physical attack must be able to update its perturbation pattern to match environmental changes within these small spans of time.

4. Threat Model

We consider an adversary launching a *latency attack* against camera-based perception in autonomous systems, aiming to overload ODT algorithms with numerous objects to slow processing. The attacker has access to their own camera for real-time environmental information and a projector for perturbations. We primarily evaluate attackers with white-box knowledge of the victim's *object detection* model (e.g., determined via the victim's technical specifications), examining black-box knowledge of OD in Section 6.3. However, we assume black-box knowledge of OT or NMS algorithms, as DETSTORM 's objective function generalizes across different approaches for both.

We physically conduct attacks with projector-based perturbations, as detailed in Section 6. These attacks must comply with *physical constraints*, such as limited perturbable surfaces, projector brightness, and competing light sources. Physical access to the victim's hardware is not considered, as this would trivialize the attack [19].

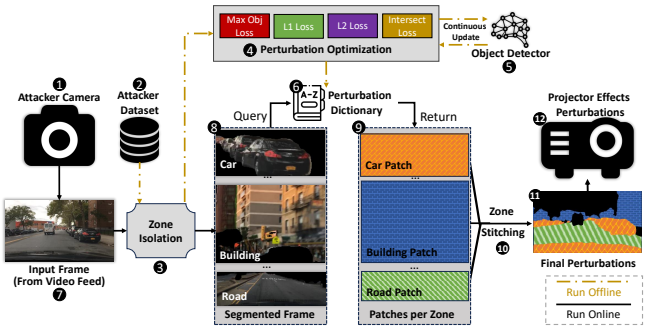


Figure 2: Illustration of DETSTORM's stages.

5. Physical Latency Attacks

We present DETSTORM, a physically launchable latency attack against ODT. Figure 2 presents the stages of DETSTORM. To begin, the attacker requires a camera (❶) for environmental mapping and an attacker dataset (❷) to generate perturbations. Next, the *perturbation dictionary* is created offline, before the attack is conducted. The attacker dataset, which can either be a public dataset (e.g., BDD100k [55]) or collected directly from the attack environment, is first passed through *zone isolation* (❸, Section 5.1), which uses depthwise separable convolutions to split images or videos into *zones*, with one zone for each perturbable class in a zone-isolated frame. Classes that are known not to be perturbable (e.g., sky, windows) are ignored. Therefore, the combination of all zones for an isolated frame represents the perturbable parts of the image, which addresses challenge C_1. Next, the zone-isolated dataset is passed into *perturbation optimization* (❹, Section 5.2.1), which maximizes the amount of objects in *each* perturbable area and addresses challenge C_2. During perturbation optimization, we continuously query an object detector (❺) to update our perturbation pattern over a set number of iterations. To adapt to a variety of environmental lighting conditions, we allow our noise to remain unbounded, but we minimize its intensity to preserve stealthiness. addressing challenge C_3.

When optimization is complete, we store the perturbation patterns for each zone in our perturbation dictionary (❻). During the attack runtime, DETSTORM first takes the input frame (❼) from the attacker camera, runs it through zone isolation, and returns the *segmented frame* (❽) containing each perturbable zone. Next, these zones are queried against the perturbation dictionary, which returns one perturbation pattern per zone (❾). Finally, in order to adapt the perturbations to changes in the environment, our *zone stitching* algorithm (❿, Section 5.2) unifies the perturbation patterns to match the layout of the environment in real time, addressing challenge C_4. The unified pattern is displayed via a projector to effect the attack (⓫-⓬).

5.1. Zone Isolation

Previous latency attacks seek to maximize the amount of created objects over an entire image. Unfortunately, this approach is not viable in a physical attack; some parts of the image may not be perturbable, and therefore objects cannot be created there. To address this, we split the image into perturbable zones, and maximize the amount of created objects for each. By induction, this also maximizes the total amount of objects for all perturbable areas in the image.

We define a "zone" as the region of an image frame that belongs to one of 119 classes that we reason can reflect projector perturbations (e.g., cars, pedestrians, buildings, roads, and animals). We detail all perturbable classes and their frequency within the BDD100K dataset in Appendix A.

To segment an image frame into zones, we train an Xception model [12] on ADE20K segmentation data, which contains a diverse amount of scene examples from various domains. Our aim is to achieve real-time performance (20-30 fps) with a complex model structure, prioritizing speed over maximizing accuracy, which is imperative for *zone stitching* (Section 5.2.2). Specifically, we employ *depthwise separable convolutions* of the form:

$$\text{Depthwise}(X, W_d)_{i,j,d} = \sum_{k=1}^{D_{\text{in}}} X_{i,j,k} * W_{d,k}, \qquad (1)$$

where D_{in} is the number of input channels, $X_{i,j,k}$ is the input activation at (i, j) for channel k, W_d is the depthwise convolutional filter for channel d, and $*$ is the convolution operation.The output of the depthwise convolution, with D_{out} channels, is then passed to a pointwise convolution, which performs a 1×1 convolution across all channels via:

$$\text{Pointwise}(Y, W_p)_{i,j,d'} = \sum_{d=1}^{D_{\text{out}}} Y_{i,j,d} * W_{p,d,d'}, \qquad (2)$$

where $Y_{i,j,d}$ is the output of depthwise convolution at (i, j) for channel d, and W_p is the pointwise convolutional filter.

Given an image frame I from our attacker camera, the model applies Equations 1 and 2 to output an array of zones $Z = [z_1, z_2, ..., z_n]$, with class information and a mask encoding which parts of the image the zone is located in. These are passed as output to the perturbation dictionary, which assigns relevant perturbation patterns to each zone. Every time a new frame is received, this process is repeated, ensuring that, as the attacker continuously maps the surroundings using their camera, the attack updates the zone and class information to match the environment.

5.2. Creating Latency

To create the perturbation dictionary, we generate universal adversarial perturbations for all 119 zone types. We first partition the dataset (e.g., BDD100K [55]) via zone isolation into zone-specific folders (e.g., "person" for pedestrian zones, "car" for car zones). Each folder's zones are batched according to system memory constraints (e.g., batch size of 100 for high-memory systems, 10 for low-memory systems), and these batches are passed to *perturbation optimization* for noise generation.

5.2.1. Perturbation Optimization. To optimize perturbations for a given batch, we use projected gradient descent (PGD), minimized by the $L2$ norm to reduce perturbation amount and enhance attack effectiveness (Section 6.4). Our loss function maximizes created bounding boxes while minimizing L1 distance, L2 distance, and intersection for each pair of boxes, ensuring evasion of various NMS approaches.

Maximizing Created Objects. To cause latency effects, we create a large number of objects that must be processed in $O(n^2)$ to $O(n^3)$ time by NMS and OT. To avoid being filtered by NMS, the created objects must have a confidence score greater than the victim's confidence score threshold T_{conf}. Thus, the attacker first provides a target confidence threshold T'_{conf} that the created bounding boxes aim to achieve. Ideally, T'_{conf} should be set to T_{conf} if the value is known or can be estimated. However, the attack can remain effective even when $T'_{\text{conf}} \neq T_{\text{conf}}$ (Section 6.4).

Next, for each bounding box B created by the perturbation, we minimize the loss function:

$$\ell_{\text{conf}} = \max\left(T'_{\text{conf}} - B_{\text{conf}}, 0\right), \qquad (3)$$

where B_{conf} is the confidence score of the created bounding box. Equation 3 maximizes B_{conf} with respect to T'_{conf}, while no incentive is given to increase B_{conf} past T'_{conf}. With this, we ensure that, over multiple objects, the loss function favors increasing the confidence of objects below the threshold, rather than increasing the confidence of objects above it.

Let C_b be the set of candidate bounding boxes before applying the confidence threshold and C_a be the candidates after. We maximize the amount of created bounding boxes by minimizing the following loss function, over all objects:

$$\ell_{\text{max\_obj}} = \frac{1}{|C_b|} \sum_{B \in C_a} \ell_{\text{conf}}(B). \qquad (4)$$

We design Equation 4 so that the loss is minimized as the number of created objects where $B_{\text{conf}} \geq T'_{\text{conf}}$ increases.

Minimizing L1 Distance. The Confluence [45] approach filters out repeated bounding boxes based on an L1 distance metric. To combat this, we would like to minimize the L1 distance between our created bounding boxes. Let $d_1(i, j)$ be the L1 distance between boxes i and j given by:

$$d_1(i, j) = |x_i - x_j| + |y_i - y_j|, \qquad (5)$$

where (x_n, y_n) represents the centroid coordinate of the bounding box n. Let α and β be the coordinates of the upper left and bottom right corners of the current zone, respectively. We minimize L1 distances between all pairs of created boxes through the loss function:

$$\ell_{l1} = \sum_{\forall C_b} d_1(C_b(i), C_b(j)) \cdot \frac{1}{|C_b| \cdot d_1(\alpha, \beta)}, \qquad (6)$$

which reduces the average L1 distance calculated over all

created objects. We weight this value based on the size of the entire zone $d_1(\alpha, \beta)$. Intuitively, given the same number of created objects, a larger zone is more likely to have a higher average L1 distance with created objects compared to a smaller zone, thus we adjust the loss function accordingly.

Minimizing L2 Distance. DIoU computes the NMS overlap based on the L2 distance between the bounding boxes, reasoning that objects whose centroids are farther apart are more likely to be different objects [59]. Although reducing L1 distance via Equation 6 indirectly reduces the L2 distance as well, it does not account for potential outliers. Therefore, given the function

$$d_2(i,j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}, \tag{7}$$

we directly minimize L2 distances between created bounding boxes with:

$$\ell_{l2} = \sum_{\forall C_b} d_2(C_b(i), C_b(j)) \cdot \frac{1}{|C_b| \cdot d_2(\alpha, \beta)} \tag{8}$$

Similar to L1 distances, L2 distances are minimized as an average over all objects, weighted by the size of the zone to account for larger zones corresponding to larger average L2 distances among bounding boxes.

Minimizing Intersection. Independent of *any* IoU methodology, it is intuitive that objects with a low intersection are less likely to be the same object since they do not occupy the same space. This principle is applied either directly or indirectly by vanilla NMS, DIoU, and Confluence. Therefore, we seek to minimize the intersection between the bounding boxes created in our loss function.

Given two bounding boxes B_i and B_j, with coordinates in the form $B = (X_1, X_2, Y_1, Y_2)$, where (X_n, Y_n) encodes the coordinate for either top left point $(n = 1)$ or the bottom right point $(n = 2)$, we first compute the maximum extents on each axis:

$$\begin{aligned} X_{(a,b)} &= \max\left(X_{(1,2)}(B_i), X_{(1,2)}(B_j)\right), \\ Y_{(a,b)} &= \max\left(Y_{(1,2)}(B_i), Y_{(1,2)}(B_j)\right), \end{aligned} \tag{9}$$

With these, we compute the intersection of B_i and B_j with:

$$\cap(B_i, B_j) = \left|\max(X_b - X_a, 0) \cdot \max(Y_b - Y_a, 0)\right|. \tag{10}$$

Using Equation 10, we formulate a loss function that minimizes the intersection on all created objects:

$$\ell_{\text{inter}} = \frac{\max\left(\cap(B_i, B_j)\ \forall B_{i,j} \in C_b\right)}{\alpha \cdot \beta}, \tag{11}$$

which reduces the *maximum* intersection between any two created boxes, normalized according to the size of the zone. This design ensures that instead of seeking small reductions in intersection over multiple objects, the loss function attempts to target the two bounding boxes with highest intersection and reduce their overlap. This helps the loss function ignore bounding boxes with already minimized intersections and focus on the objects most at risk of elimination.

Final Loss Function. To create the final perturbation pattern P for each zone, we minimize a loss function by combining our four loss functions into a single equation:

$$\ell = \min_{P}\left[\lambda_1 \cdot \ell_{\text{max\_obj}} + \lambda_2 \cdot \frac{\ell_{l1} + \ell_{l2} + \ell_{\text{inter}}}{3}\right], \tag{12}$$

where λ_i is a weighting factor set by hyperparameter tuning (See Section 6.1). Over an attacker-specified number of iterations, we compute a gradient over the object detection model and update P via back-propagation. When completed, we store the optimized pattern P in the perturbation dictionary for use against future examples on the same class of zone.

Algorithm 1 Creating a final perturbation pattern from zones.

Input: Set of input zones Z = [class, mask] $\forall$ zones, Set of perturbations $\rho\ \forall\ Z$, Image width and height (I_w, I_h).
Output: Final perturbation pattern to project.

```
1: function STITCH_ZONES(Z, ρ, (I_w, I_h))
2:     return_image = zeroes(I_w, I_h, 3)
3:     for class, mask in Z do
4:         current_pert = ρ(class)
5:         cubic_interpolate(current_pert, (I_w, I_h))
6:         for rgb_channel in current_pert do
7:             current_pert(rgb_channel) *= mask
8:         end for
9:         return_image += current_pert
10:    end for
11:    return return_image
12: end function
```

5.2.2. Zone Stitching. To adapt to changes in the environment in real time, perturbations from our perturbation dictionary need to be selected and stitched together into a single, contiguous region that matches the environment. This also handles the conversion of digital perturbations to physical, projectable patches that can be displayed by our projector. Algorithm 1 outlines our zone stitching process. First, it leverages zone isolation (Section 5.1) to distill the layout of the environment into a set of regions Z, encoding the class and area of each zone in the image. Next, it takes the set of corresponding perturbations ρ from the perturbation dictionary generated during perturbation optimization (Section 5.2.1). Lastly, it takes the attacker camera's width and height to size the final perturbation properly.

The algorithm starts with a completely blank `return_image`, represented by an array of zeroes encompassing the image's width/height along with all three RGB channels (Line 1). Next, for each class and mask in the input zones (Line 3), we take the corresponding perturbation pattern `current_pert` from ρ and resize it to the full image size via cubic interpolation (Lines 4-5). To constrain the perturbations to the zone's physical bounds, we multiply each channel in `current_pert` by the attack mask and add it to the `return_image` (Lines 6-9). This populates `return_image` with the perturbation pattern of each zone over each iteration, until each zone's perturbation pattern has been layered onto the blank image, encompassing all perturbable regions. This final pattern is returned (Line 11) and projected directly onto the environment, snapping to the correct surfaces and effecting the attack.

6. Evaluation

We evaluate DETSTORM against a full ODT stack including OD, NMS, and OT. For OD, we select YOLOv5 [26], which remains both the most widely used OD algorithm [20] and one of the fastest in terms of latency [10]. Additionally, we evaluate DETSTORM against four different OT algorithms (SORT [5], StrongSORT [16], OC-SORT [8], and SiamMOT [47]) along with three different NMS algorithms (vanilla, DIoU [59], and Confluence [45]). Using these models, we perform (1) simulated physical attacks enforcing real-world constraints against the BDD100K autonomous driving dataset [55], and (2) real-world physical attack experiments with a camera, projector, and real-world vehicles. We compare the success of our attacks with previous work (Section 6.2), the transferability of the attack to other OD models (Section 6.3), and the effect of victim-configured or environmental parameters on the attack (Section 6.4).

6.1. Attack Scenarios and Setup

Domain. For our experiments, we focus on the *autonomous vehicle* domain, where latency effects are the most safety critical and can result in collisions (Section 7). However, our attack is equally applicable to other domains, including autonomous surveillance (where latency can cause frames to be ignored for the duration of the attack, potentially disguising unauthorized entry [50]) and mobile robot navigation (where latency for delivery robots [43] may cause service delays, or cause them to ignore objects on the road).

Datasets and Models. We evaluate our attacks against 1000 randomly selected examples from the BDD100K dataset [55], covering diverse driving scenarios. Additionally, we use image frames from BDD100K to create the perturbation dictionary (Section 5.2). For ODT, our focus is on YOLOv5 and SORT, as they are the most widely used model combination to date [20]. For the same reason, we evaluate attack effectiveness with vanilla NMS (Section 6.2). However, in Section 6.3, we assess DETSTORM on other OD/OT/NMS approaches. We evaluate Vanilla NMS using Ultralytics' default YOLOv5s implementation [26]. To evaluate DIoU, we repurpose the official DarkNet implementation of the metric [58] for our YOLOv5s model. These NMS implementations all run on the GPU via PyTorch. Confluence and the four object tracking algorithms are evaluated using their respective Python implementations [5], [8], [16], [46], [47].

Testing Hardware. We evaluate the effectiveness of our attacks on various hardware, since the observed latency of a given attack is primarily dependent on the GPU. We test latency effects on five different GPUs. The first three, NVIDIA RTX 2080 Ti, NVIDIA RTX 3070, and RTX 3080, are professional visual computing platforms, which are common in testing latency attacks against AD in previous work [30], [32]. The other two, NVIDIA Jetson Nano and NVIDIA Jetson TX2, are embedded edge-device GPUs commonly used in a wide variety of autonomous systems, including video surveillance, robotic vehicles [39]. We ran our experiments with 12th Generation Intel CPUs and 32GB of RAM, except for the Jetson devices which used their own CPU and 8GB RAM.

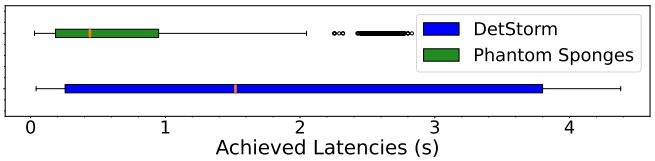


Figure 3: Latency distribution achieved by Phantom Sponges and DETSTORM across all hardware. DETSTORM is able to consistently achieve latencies beyond PS's top performance.

For real-world experiments, we used a real-world DataSpeed autonomous vehicle [24] (Appendix Figure 14), which is equipped with our RTX 2080 Ti GPU. The onboard camera recorded our attacks at 30 FPS in 1920×1080 resolution. To ensure safety, a human operator remained in control of the vehicle at all times, with the attack effect being measured via replayed data while the vehicle was stationary. This also allowed us to measure the effect of the same attacks under different GPUs. To conduct the attack, perturbations were generated on a laptop with an Intel i5-9300H CPU/NVIDIA GTX 1650 GPU/8GB RAM and effected via an Optoma LV130 projector [40]. On this setup, the runtime of the DETSTORM online perturbation loop averaged 28.796 fps, ranging between 22 and 30 fps, with zone isolation consuming 97%–97.8% of runtime, dictionary queries 1%, and stitching 1.2%–2%.

Real-world Latency Attacks. We conduct 34 real-world attacks with our DataSpeed autonomous vehicle, in residential and commercial areas. 15 attacks were done in daytime. We prioritized safety during these experiments, choosing times and locations to minimize third-party traffic conditions and using personnel with experience in autonomous vehicle experiments. Our experiments were conducted with permission from the local police, and local patrol officers were notified of our activities.

Comparison Baselines. We compare our attack against the Phantom Sponges attack [44]. To our knowledge, it is the only work whose implementation is both publicly available and complete (i.e., the effects described in the original work are reproducible). We generate a Phantom Sponge UAP on the BDD100K images dataset, the same source we use for generating the perturbation dictionary. We use the settings and hyperparameters recommended by the authors.

Implementation Details. For our evaluation of attack success in Section 6.2, we adopt the default settings of the official implementations of each model. We evaluate the effect of changing these settings in Section 6.4. We set $T'_{\text{conf}} = 0.25$, a common value in NMS implementations. For our perturbation generation loss function, we set $\lambda_1 = 1$ and $\lambda_2 = 5$, where both values are tuned to optimal via grid search.

Evaluation Metrics. For simulated physical attacks where attacked examples can be compared against a ground truth baseline, we use two different metrics to measure the effectiveness of our attack:

(1) Rate of Increase in Objects (ROI-O) measures the relative increase in number of *final objects* processed by the

Table 2: Attack performance of DETSTORM during simulated physical experiments. DETSTORM can cause up to 92.5× increase in latency and achieves max latencies well above the 0.1 second safety threshold [28].

Hardware	ROI-O (Avg/Max)	ROI-L (Avg/Max)	Max Latency (s)
Jetson Nano	5.06 / 20.71	51.6 / 61.7	4.38
Jetson TX2		49.4 / 51.22	3.8
RTX 2080 Ti		53.33 / 75.88	1.52
RTX 3070		1.74 / 2.49	0.5
RTX 3080		21.33 / 92.5	0.4

perception pipeline when attacked, in the form:

$$\text{ROI-O} = \frac{\text{Object\_Num}(x^*) - \text{Object\_Num}(x)}{\text{Object\_Num}(x)} \quad (13)$$

(2) Rate of Increase in Latency (ROI-L) measures the relative increase in *total runtime* of the perception pipeline when attacked, in the form:

$$\text{ROI-L} = \frac{\text{ODT\_Latency}(x^*) - \text{ODT\_Latency}(x)}{\text{ODT\_Latency}(x)}, \quad (14)$$

Where, for both metrics, x is an input with no perturbations, and x^* is the same sample that has been attacked.

ROI-L is hardware-dependent and may change based on the victim's platform, while ROI-O is hardware-independent and can be used as a rough measure of expected latency based on the number of created objects. Thus, the most important metrics for an attacker to achieve are ROI-O (for hardware-independent effects), and *maximum latency*, which will lead to effects for the victim. The exact value at which a maximum latency will cause physical effects increases as speed and distance between vehicles decreases (Section 7). However, a max latency above *0.1 second* has been deemed unsafe in the autonomous driving domain [28].

6.2. Attack Effectiveness

Simulated Physical Attacks. To evaluate DETSTORM's effectiveness across diverse conditions and compare it with the digital Phantom Sponge baseline, we directly apply DETSTORM's perturbations and a Phantom Sponge UAP to BDD100K video data. To simulate physical constraints, both attacks are bound to the same physically perturbable regions. Additionally, the maximum perturbation amount ϵ was restricted to 70 for both attacks, although both attacks are capable of achieving a much lower perturbation amount during optimization.

Table 2 showcases the DETSTORM on YOLOv5s, with vanilla NMS and SORT tracking. For comparison, Appendix Table 7 illustrates the performance of Phantom Sponges on the same setup. We find that DETSTORM easily clears the 0.1 second safety threshold and outperforms the Phantom Sponges attack in all areas, having a 4% greater average ROI-O, and having between 55.36% and 410.29% higher ROI-L. This proves that, for the same environments with the same objects, DETSTORM is able to create more objects and longer latencies, regardless of victim-deployed hardware. Notably, although the maximum number of objects produced

(a) BDD100K. (b) Right turn (c) Drive in (parking)

Figure 4: DETSTORM's latency attack (a) against BDD100K and (b) in the real world as the victim turns right, and (c) in the real world as the victim pulls in to park.

Table 3: DETSTORM latencies in physical experiments.

Hardware	Avg Latency (s)	Max Latency (s)	Benign Latency (s)†
Jetson Nano	7.17 **(× 55.15)**	8.1 **(× 62.3)**	0.13
Jetson TX2	3.2 **(× 29.1)**	6.05 **(× 55)**	0.11
RTX 2080 Ti	1.79 **(× 28.96)**	3 **(× 48.54)**	0.0618
RTX 3070	0.1753 **(× 5.05)**	0.6 **(× 17.29)**	0.0347
RTX 3080	0.1749 **(× 13.25)**	0.56 **(× 42.42)**	0.0132

† Benign latency based on average performance on unaltered BDD100K dataset.

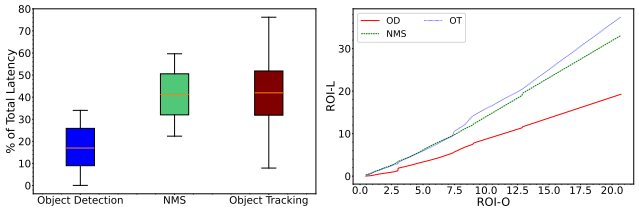
by DETSTORM is only 1.37% higher than PS, DETSTORM produces between 40% and 300% higher maximum observed latencies, demonstrating that DETSTORM's created objects are more efficient at creating latency. Finally, Figure 3 shows the distribution of latencies achieved by both Phantom Sponges and DETSTORM. DETSTORM has a large upper quartile skewed towards higher latencies, which far exceeds the maximum and outliers of Phantom Sponges. By being able to effect higher latencies, DETSTORM is able to cause physical consequences for a wider variety of conditions.

Among our RTX GPUs, the RTX 2080 Ti, mounted to our real vehicle, has the highest observed latency at 1.52 seconds, whereas our most powerful RTX GPU (RTX 3080) has the lowest observed latency (0.4 seconds). Meanwhile, on the embedded Jetson edge-devices, even higher latencies are achieved, up to 4.83 seconds for the Jetson Nano and 3.8 seconds for the Jetson TX2. The higher maximum latency and ROI-L in these devices demonstrates their difficulty to scale to high object densities, due to the processing constraints in both their lower GPU throughput and more limited RAM.

Yet, we note that the ROI-L is not necessarily correlated with the maximum latency. For example, the highest DETSTORM ROI-L is observed under the RTX 2080 Ti, with an average increase of 53.33 times the benign latency. However, it has a lower maximum latency than both the Jetson Nano and Jetson TX2. This occurs because some GPUs (e.g., RTX 2080 Ti) are extremely fast under low object density, but experience more extreme latency spikes under high object density. Thus, although the maximum observed latency roughly correlates with GPU throughput, the *relative* rate of increase in latency is more tied to memory access efficiency [38].

Figure 4-a illustrates the effect of a DETSTORM attack against BDD100K. Both perturbations and the attack's effect are limited to physically perturbable regions (e.g., walls, vehicles, and passing pedestrians). A large number of objects are created on these surfaces, creating the latency effect against perception.

Real-world Attacks. To evaluate the effect of DETSTORM on real-world automotive vision tasks, we use a projector to physically perturb a variety of outdoor environments. We



(a) (b)

Figure 5: The responsibility of each module for latency effects, measured via (a) percentage of total latency over all simulated physical/real-world experiments, and (b) averaged over digital experiments as a function of ROI-O.

choose both residential and commercial environments with parking lots, buildings, signs, and other vehicles. During experiments, we record the perspective of our victim vehicle as it performs usual actions (driving forward along the road, turning, parking, and reversing) and measure how the perception pipeline responds to the perturbation patterns.

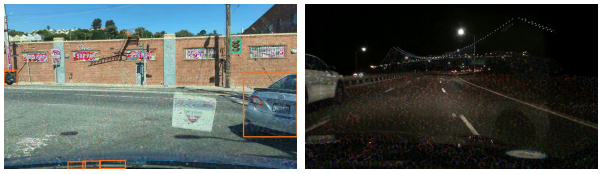
The attacker can use one of two strategies for projecting perturbations, illustrated in Appendix Figure 15. First, the attacker can carry the projector and manually point it at surfaces the victim can see (Figure 15-a). Otherwise, the attacker can mount the projector onto a vehicle to follow the victim, continuously perturbing the environment (Figure 15-b).

Table 3 compares the average/max latencies from real-world attacks with benign latencies averaged over the unmodified BDD100K dataset. While not a perfect baseline due to distribution differences, benign latencies offer a good figure to estimate relative latency increase due to attack. Similar to our digital attacks, we find that the greatest observed latency for RTX GPUs is on the RTX 2080 Ti, and the greatest overall latency is observed on the Jetson Nano. Interestingly, our real-world attacks achieve higher latencies than digital attacks, having between 20% (RTX 3070) and 97.37% (RTX 2080 Ti) higher maximum latencies depending on hardware. We note that higher the ROI-L experienced under digital DETSTORM attacks, the higher the increase in max observed latency was in real world experiments, once again highlighting the importance of memory access efficiency in mitigating latency.

Figures 4-b/c demonstrate two examples from our real world attacks. In Figure 4-b, the victim is taking a right turn into a driveway, while the attacker projects perturbation patterns on both the left and right sides of the wall, as well as on the sign. A large amount of objects appear on the right side of the wall, effecting the attack. The attacker aims to have perception freeze mid-turn, causing the victim to collide head on with the corner of the wall. In Figure 4-c, the attacker projects noise onto a wall as the victim comes in to park, creating a large amount of objects on the left and right garage doors. The attacker aims to have perception freeze as the victim pulls in, causing it to fail to come to a stop and collide with the wall. Although we do not evaluate the collisions in the real world for safety reasons, we discuss the requirements for such collisions to occur in Section 7.

Latency Per Perception Module. Although DETSTORM targets the entire camera-based perception pipeline, OD, NMS, and OT each respond differently to the latency attack. Figure 5-a breaks down the percentage of total latency (when attacked by DETSTORM in simulated physical and real-world experiments) that each module of perception represents. Overall, OT makes up the most significant portion of total latency time, representing 42% of total latency on average. NMS is also significant, representing 41% of the total latency time on average. Finally, object detection represents the smallest amount of latency on average, only 17%. This indicates that improvements in NMS and OT processing speeds are the most critical to mitigate latency effects compared to improvements in OD processing speeds.

Figure 5-b plots the increase in ROI-L for each perception module during our digital experiments, as a function of ROI-O. We see that for low ROI-O values, detection can have a higher latency than tracking and NMS. However, as the number of created objects increases, the latency increase for both OT and NMS increases rapidly due to their $O(n^2)$ to $O(n^3)$ runtime complexity. Meanwhile, the growth for OD is more linear.



(a) Object Creation [36] (b) Cooling-Shrinking [54]

Figure 6: Perturbations generated on YOLOv5s and applied to YOLOv5m showing different attack effects.

6.3. Transferability to other Models

In previous sections, we evaluated DETSTORM on YOLOv5s, vanilla NMS, and SORT tracking. In this section, we evaluate how the attack performs when the victim uses different object detectors, NMS algorithms, and trackers.

OD Algorithms. We tested generating/applying perturbation patterns on YOLOv4 and YOLOv3 to evaluate transferability between completely different OD models/weights. To evaluate transferability between similar OD models with different weights, we conducted the same evaluation on five different YOLOv5 variations (n, s, m, l, and x). We detail the differences in YOLO architecture in Appendix B. We found that *latency* effects do not transfer between models, i.e., perturbations generated via the YOLOv5s model will not create spurious objects for YOLOv5n, YOLOv3, or any other models, indicating that the creation effect is highly sensitive to the model architecture and weights. However, we found two consistent effects when applying attacks generated on one model to another. Firstly, 16% of cross-model perturbations cause *object creation*, where five or fewer objects are created somewhere in the image (Figure 6-a). These are artifacts of our mass creation process, and although it is not enough to cause a latency effect, it can cause other issues, e.g., vehicle stoppages, as evaluated in previous

Table 4: The percentage of created objects created by DETSTORM and Phantom Sponges that remain unfiltered under our three evaluated NMS methods.

	Vanilla	DIoU [59]	Confluence [45]
DETSTORM	100%	100%	100%
Phantom Sponges [44]	100%	79.26%	67.64%

Table 5: Performance of DETSTORM to other object trackers (relative to SORT [5])

	StrongSort++ [16]	OC-SORT [8]	SiamMOT [47]
ROI-O%	100%	100%	100%
ROI-L%†	937.42% ↑	77.16% ↓	424.28% ↑

† Based on performance on RTX 3080.

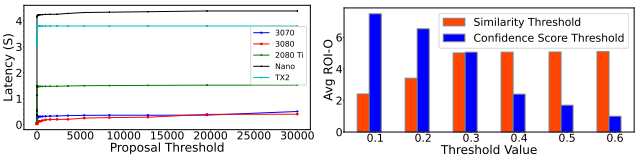
work [36]. Second, 63% of transferred perturbations cause *cooling shrinking* [54], where objects and their trackers are suppressed throughout the video feed (Figure 6-b). Finally, 21% of transferred perturbations cause object creation at extremely low confidence (< 0.1), and thus do not affect the perception pipeline unless T_{conf} is set sufficiently low.

Previous work [44], [49] has demonstrated that *ensemble training* can increase latency attack transferability across models. In ensemble training, instead of using a single object detection model during the propagation process in perturbation generation (Section 5.2.1), a different model is selected at random each time, which enforces loss function over each of the selected models. Though ensemble training cannot transfer perturbations to models that the attacker does not include in the training process, it is an option when multiple different known and accessible models may be used by the victim.

NMS Algorithms. In this paper, we consider the different types of NMS algorithms that can be deployed by the victim. We evaluate DETSTORM's transferability to these NMS algorithms by measuring the number of objects that are filtered out by the DIoU and Confluence NMS algorithms. Additionally, we evaluate the effectiveness of our loss function's design by comparing DETSTORM against Phantom Sponges, which is only designed for vanilla NMS.

Table 4 reports the percentage of created objects that remain unfiltered by each NMS method. As designed, DETSTORM does not suffer any loss of performance when evaluated under DIoU NMS or Confluence instead of Vanilla NMS. However, Phantom Sponges, designed only against vanilla NMS, has an average of 20.74% of its detections removed by DIoU and 32.36% of its detections removed by Confluence. Because distance metrics are not considered by the Phantom Sponges loss function, it tends to leave a larger gap between bounding boxes than DETSTORM, which makes it less transferable to DIoU and Confluence.

Object Tracking Algorithms. In addition to the common SORT tracker, we examine how DETSTORM performs under two different variations of SORT: (1) StrongSORT++ [16], one of the most accurate (by Multiple Object Tracking Accuracy [3]) tracking-by-association approaches, which improves upon SORT by using two unique algorithms to extract additional spatio-temporal information from objects, and (2) OC-SORT [8], one of the fastest two-stage MOT models which prevents tracking errors from accumulating by correcting untracked detections based on their previous behavior. Although they still include the $O(nm^2)$ Hungarian Matching phase of SORT, they can potentially reduce additional latency by eliminating erroneous trackers in m. Additionally, we evaluate DETSTORM under SiamMOT [47], a state-of-the-art multiple object tracker which uses *feature embeddings* in a Siamese neural network to associate objects between frames. By eliminating the presence of trackers entirely, complexity can drop between $O(n)$ and $O(n^2)$, depending on the input data [47].



(a) Proposal (b) Confidence/Similarity

Figure 7: DETSTORM attack effectiveness under different victim-set thresholds. The victim cannot tune their thresholds to evade latency effects without either reducing perception accuracy or creating other attack effects.

Table 5 compares the ROI-O and ROI-L of each evaluated object tracker with the results achieved on SORT. A value of 100% indicates that the same value as SORT is achieved, a value $> 100\%$ indicates a larger value than SORT, and $< 100\%$ indicates a smaller value is achieved. All three trackers achieve the exact same ROI-O as with SORT, indicating that none of our created objects are removed by any trackers.However, the total measured latency changes when different trackers are used. SiamMOT and StrongSORT++ both process bounding boxes much slower than SORT, resulting in a 4-9 times *increase* in latency, respectively, which means a degraded performance. Much of the latency increase here comes from the feature extraction phase of both approaches, rather than the association phase, showing that a decrease in time complexity for *association* at the expense of more *detection* time is not always a good tradeoff. Meanwhile, OC-SORT gets a slight improvement over SORT, decreasing overall latency by 22.84% due to faster processing speeds during tracking. This proves that improvements in object tracking speeds can mitigate latency effects for the entire pipeline.

6.4. Attack Parameters

Proposal Number Threshold. A safety-critical system must withstand worst case scenarios to prevent failures. The proposal threshold T_{prop} is typically set to $30,000$ [41], accommodating potentially large detections in various general detection scenarios. However, the "worst case scenario" varies by applications. For example, detectors for cars, pedestrians, and dynamic objects may favor a high T_{prop}. Conversely, detectors for low-frequency objects like traffic signs and lights may suffice with a lower threshold (e.g., 1000).

Figure 7-a charts the impact of different T_{prop} values on the observed latency (in seconds) across our hardware setups. Latency effects are stable between a T_{prop} of $5,000$ and $20,000$, before slowly increasing as T_{prop} approaches $30,000$. General detectors handling multiple object types may use

these values for T_{prop}, and they would be unable to tune T_{prop} to mitigate latency effects. However, latency effects begin to drop off rapidly at $T_{\text{prop}} = 3,000$ for all hardware. This implies that specialized detectors with few expected objects may not experience latency effects. However, by creating at least T_{prop} objects with a T'_{conf} greater than or equal to the highest confidence benign object, DETSTORM would be able to completely replace all of the detected objects with adversarially crafted ones, which could cause various effects (e.g., a failure to stop due to the original traffic light being removed and replaced with hundreds of speed limit signs). The smaller that T_{prop} is set to, the easier it is for an attacker to achieve this effect, making it infeasible to use T_{prop} tuning as a defense.

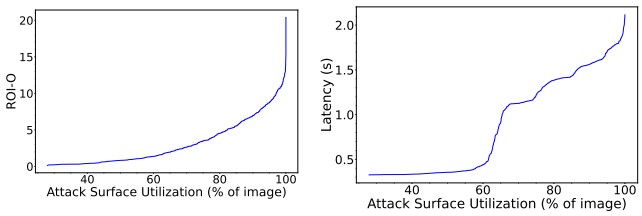
Confidence Score Threshold. The confidence score threshold T_{conf} is set based on the expected accuracy of an object detector during its usage. Setting T_{conf} too low increases the risk of false positives, while setting it too high filters out both false positives and benign detections.

Figure 7-b (blue) illustrates the average ROI-O of DETSTORM under different T_{conf} values. Lower thresholds result in higher ROI-O, as more lower-confidence objects survive filtering. However, at $T_{\text{conf}} = 0.4$, we begin to lose 33% of benign detections along with our adversarial ones. At $T_{\text{conf}} = 0.5$, 71% of benign detections are lost, and at $T_{\text{conf}} = 0.6$, both benign and adversarial detections are wiped out completely, dropping the ROI-O to 1. Nonetheless, setting the attacker-desired confidence T'_{conf} higher allows overcoming any threshold, rendering T_{conf} tuning insufficient to mitigate DETSTORM's latency attacks.

Similarity Threshold. The similarity threshold T_{Sim}, ranging from 0 to 1, exists for all considered NMS approaches, and governs whether two bounding boxes are treated as part of the same object during NMS. For vanilla NMS, T_{Sim} is the IoU threshold; for DIoU, the distance threshold; and for Confluence, the confidence-weighted Manhattan distance threshold. Setting T_{Sim} too low hurts benign performance, as the NMS will become more aggressive and attempt to combine separate objects into one bounding box. However, setting T_{Sim} too high can lead to more false positives, as duplicate objects normally filtered by NMS are no longer removed. Typically, T_{Sim} is set between 0.45 and 0.5 for general detection purposes. [41], [46], [59].

Figure 7-b (red) shows the ROI-O of DETSTORM under different values for T_{Sim}. Because we optimize our perturbations for all three NMS approaches, they perform the same for a given T_{Sim} value. We find that for $T_{\text{Sim}} = 0.1$ and 0.2, we still achieve between 2.41 and 3.41 ROI-O, although the NMS aggressively filters any of our created boxes that were not properly minimized. At $T_{\text{Sim}} = 0.3$, most of our created bounding boxes have already been minimized to be considered unique at this threshold. Therefore, for $T_{\text{Sim}} > 0.3$, performance gain is minimal, with a stable ROI-O achieved. Compared to confidence and proposal thresholds, T_{Sim} has a less pronounced effect on latency effects, and the victim cannot tune this parameter to evade the attack.

Attack Surface Utilization. The availability of attack surfaces profoundly influences DETSTORM's success as a physically realizable attack. Figure 8 plots how the utilization of attack surfaces affects success, with Figure 8-a showing how the ROI-O increases along with attack zone utilization during digital experiments, while Figure 8-b shows how observed latency changes with respect to attack surface utilization for all experiments, including the real world ones. The number of successfully created objects grows *exponentially* with a utilizable attack surface. At 0% surface utilization, no effect is achieved, whereas the number of created objects is maximized when 100% of the environment is usable (e.g. an entire wall covering the victim's view is available to perturb). However, the number of created objects only reaches 20% of its peak when 82% of the environment is available to perturb, and 50% when 97% of the environment is available. This exponential growth can be attributed to the topology of the attack surfaces: at lower attack surface utilizations, different patterns must be spread across smaller objects, whereas at utilizations closer to 100%, the attack is able to utilize a single pattern for a single, contiguous surface. A similar tend is highlighted by Figure 8-b, which shows that latency effects start to sharply increase at a 60% surface utilization rate. For comparison, projector-based tracker hijacking attacks against object tracking typically require only 30% of the environment to be available to perturb [35]. Therefore, to maximize latency effects, an attacker must choose an area visible to targeted victims that has the most available surface to project perturbations onto.



(a) ROI-O (b) Latency (s)

Figure 8: How the utilization of available perturbable surfaces affects DETSTORM's ROI-O (left) and Latency (averaged over all hardware, right).

Perturbation Amount. For each attack, we measure the *maximum perturbation*, representing the most significant modification to the environment necessary for the attack. While it does not affect our success metrics, it is important to evaluate as it relates to two essential factors. First, stealthiness: higher maximum perturbations make the attack more conspicuous to bystanders and the victim. Second, projector cost: brighter perturbations necessitate more powerful (and expensive) projectors, especially in well-lit environments).

DETSTORM, on average, requires a maximum perturbation $81.43\% \pm 5.62\%$ brighter than the surroundings, with the brightest perturbation 98.82% greater luminance. For reference, this is 13.9 times brighter than the maximum perturbations required by projector-based tracker-hijacking attacks [35], but 0.8 times the brightness required by projector-based misclassification attacks [33]. Maximum perturbation is also highly dependent on the environment, with darker environments requiring a lower value. For

Figure 9: Rate of increase in number of final objects for the same perturbation pattern at different camera distances and angles, relative to the perturbed surface.

example, for night-time cases from our real-world experiments, the maximum perturbation is 68.63%.

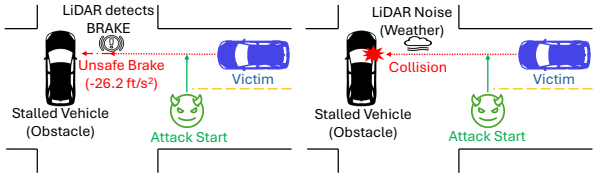
Camera Angle and Distance. We perform an ablation study to quantify the effect of camera distance and angles, relative to the perturbed surface, on our attack. First, using a flat wall in an empty area, we used a stationary projector to display a DETSTORM pattern on the wall from 3 meters away. Then, in camera distance increments of 1 meter and 22.5°on either side, we captured 3 second stationary videos centered on the wall. We repeated this with the projector turned off for baseline measurements. Camera settings were kept constant throughout the experiments, and the perturbation pattern was not transformed or altered in any way. Although we only take measurements up to 3 meters away due to environmental space constraints, we note that our real world experiments involve distances up to 8 meters.

With these data points, we construct a heatmap of the average ROI-O for each distance and angle setting, illustrated in Figure 9. We see that at close distances and tight angles closer to zero, our attack has no effect (i.e., ROI-O is 1), as the camera is close enough for imperfections in projector resolution to undermine the perturbation pattern. Additionally, at some angles, the camera is unable to capture the entire perturbed wall at once. At further distances and wider angles, the maximum effect of the attack is reached, as the camera is able to capture the entire perturbation pattern in full view with minimized influence from projector resolution. In general, an attacker can increase the effect of the attack by ensuring that the full projector pattern is visible to the victim.

7. Case Study

To safely examine the potential physical consequence of our attack on a real system, we perform a case study within the CARLA simulator [15]. Unlike our exhaustive evaluation in Section 6, this section is intended to simply show a few examples of how latency attacks may be employed against a full autonomous driving system stack. We assess our attacks using Autoware [27], an open-source software stack for self-driving vehicles, leveraging both camera and LiDAR-based perception. Although several autonomous driving systems, such as Openpilot [13], reason purely based on camera input, we select Autoware because it is stated to be fully autonomous (SAE Level 4).

Our case studies are illustrated in Figure 10, where a latency attack is performed when the vehicle crosses an



(a) (b)

Figure 10: When a latency effect of ≥ 0.27 secs is achieved, Autoware's camera-based perception pipeline can no longer respond to the environment, resulting either in unsafe deceleration when LiDAR detects vehicle (left), or collision when the LiDAR is unable to detect the object (right).

intersection blocked by a stopped vehicle. From the side of the road, an attacker projects DETSTORM perturbations onto the stopped vehicle. We experiment with several different DETSTORM attacks that yield different latency patterns.

For our specific scenario environment, we find physical consequences happen 100% of the time when a latency effect greater than or equal to 0.27 seconds is achieved, which is well within our achieved results in Section 6.2. Specifically, in all cases where this latency is achieved, camera-based perception freezes and is unable to percieve the obstacle in the intersection. In clear weather conditions, the LiDAR sensor detects the vehicle at the last moment, forcing a sudden, *unsafe brake* (Figure 10-a), decelerating the vehicle at $26.2 ft/s^2$, which is almost the double threshold for safe braking defined by the US Federal Highway Administration [18]. In addition to being an unsafe maneuver, such a brake would induce skids or poor traction on wet, icy, or dusty roads. Meanwhile, in rainy conditions (Figure 10-b), the LiDAR is unable to detect the obstacle at all due to sensor noise, leading to a *collision* 100% of the time. Similar LiDAR noise can be created by other conditions, including fog, snow, dust/sand, other LiDARs, and intense vibration. These examples confirm that latency attacks can induce physical consequences for AS.

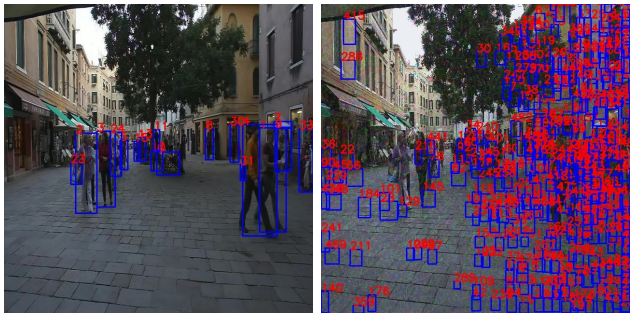
Beyond our examples, the actual latency effect required to cause a collision depends on two factors: *actuation delay* and *time to collision*. Actuation delay is entirely dependent on the autonomous driving system, and represents the normal time between when an obstacle is detected and when the system can respond to the obstacle. The higher a system's actuation delay is, the less latency is needed to cause a collision. Meanwhile, time to collision (TTC) represents the time remaining for a vehicle to brake before hitting an obstacle, in the form:

$$\text{TTC}(veh, obs) = \frac{\text{pos}_{veh} - \text{pos}_{obs}}{v(veh) - v(obs)} \tag{15}$$

where pos_{veh} and pos_{obs} are the positions of the vehicle and the obstacle, and v is velocity. Any induced latency $\geq TTC$ will create an unavoidable collision.

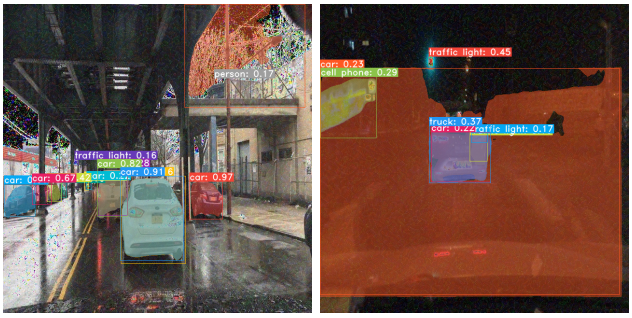
8. Limitations and Discussion

Mitigating Factors. Our evaluation highlights the importance of GPU throughput and memory access efficiency, showing that GPUs that are weaker in both areas will suffer much



(a) Normal DeepSORT (b) DeepSORT vs DETSTORM

Figure 11: DETSTORM perturbations create appearance features similar to those of real objects. Therefore, it implicitly evades appearance-based matching approaches, e.g., DeepSORT [51].



(a) (b)

Figure 12: Latency attacks against ODT implicitly affect YOLACT semantic segmentation [7], causing misclassifications and creating phantom segments.

greater latency effects. Unfortunately, current autonomous system hardware is typically geared towards embedded, low-memory solutions [39], which our evaluation shows is one of the most affected types of hardware. To fully mitigate latency attacks at the hardware level in resource-limited systems, further advances in hardware efficiency are essential.

Additionally, latency effects are dependent on the availability of the attack surface. Based on our experiments, at least 50% of the environment must be usable as an attack surface to create a significant increase in latency, with latency increasing exponentially from there. This means that attackers must carefully choose environments to maximize the amount of perturbable surfaces. Consequently, victims in urban areas with dense architecture may be more vulnerable to latency attacks compared to victims in rural areas with wide open spaces. Additionally, the consequence of malfunction in urban areas is generally more severe than in rural areas.

Unique Perception Pipelines. We design DETSTORM against the widespread camera-based perception pipeline, with distinct OD, NMS, and OT components. These pipelines currently exist in several domains, including autonomous driving, autonomous surveillance, and UAVs. Although we primarily demonstrate latency in the autonomous vehicle domain, our approach is domain-agnostic by deisgn, with our 119 perturbable classes existing in multiple domains. Porting DETSTORM to other domains requires minimal changes. The primary change would be prioritizing domain-specific prevalent classes in the attacker dataset.

Our evaluation shows that during an attack, the primary sources of observed latency come from NMS and OT components. However, recent work is exploring transformer-based *end-to-end autonomous driving* [23], [25], which performs all autonomous driving functions from perception to planning on a single neural network. Instead of distinct OD, NMS, and OT components, a continuous stream of encoders and decoders is used to accomplish tracking, mapping, and motion planning. This removes the need for the additional graph optimization or motion/appearance modeling required in NMS and OT. Given the recency of the work, the potential effect of latency attacks on unified autonomous driving is unclear, and we will explore this in future work.

9. Countermeasures

Naive Defenses. To increase the robustness of detection against our created perturbations, a victim may employ *adversarial training*, where the OD model is retrained on DETSTORM attacks to recognize and resist the perturbation patterns. However, this does not alter the model's fundamental vulnerabilities. Therefore, an attacker can easily evade an adversarially trained model by re-generating the perturbation dictionary on the new model and applying the new patterns. Yet, we find 3 denoising techniques (median/Gaussian blur and bilateral filter) that are able to remove $50\%+$ of adversarially created objects while maintaining $70\%+$ of benign detections (Appendix C). However, by applying these techniques to the perturbation pattern P in our loss function, we are able to evade the applied denoising methods 100% of the time, making it unsuitable to use as a standalone defense.

Appearance-based matching. A victim may attempt to employ an appearance-based matching approach such as DeepSORT [51] to filter out created objects. Appearance-based matching ensures that, for a given bounding box, the features inside that bounding box correspond to the reported class of the object. Unfortunately, as illustrated in Figure 11, even though no real object is enclosed by the DETSTORM-created bounding boxes, the extracted features of the perturbation pattern resemble the reported class for each object (e.g., for a created pedestrian bounding box, appearance-based matching will extract the features of a pedestrian from the perturbation pattern). Therefore, DETSTORM implicitly evades appearance-based detection. Additionally, appearance-based matching approaches carry heavy runtime overheads compared to other types of tracking [16], [51], which makes them counterproductive in mitigating latency effects.

Segmentation. A victim may attempt to filter out latency effects by constraining bounding boxes via segmentation. For example, a segmented building should not have a large number of vehicle/pedestrian bounding boxes present across its surface. However, we have found that latency attacks, including our approach and Phantom Sponges, implicitly affect segmentation approaches, including YolACT, causing instance

misclassifications and creating phantom segment areas (Figure 12). Additionally, previous work have shown that white box attacks can arbitrarily modify instance segmentation results [48], [57]. Due to our implicit black box attack effects and known white box attack effects, applying segmentation is not enough to defend against DETSTORM's latency effects.

Advanced Defenses. A victim may be interested in applying certified robustness to their models. Certified robustness defenses give a formal mathematical proof such that within a specified range of perturbations, the performance of the model will remain above a predefined threshold. However, these defenses are typically applied to pure image-based classifiers [52], as the complex mathematical operations involved make them difficult to scale to larger attack surfaces. Further, certified robustness can be evaded by increasing the intensity of perturbations to greater than the certified amount. Although this may decrease the attacker's stealthiness and increase the required cost of the projector, it will not mitigate latency effects. Yet, existing general OD defenses may be applied to protect the victim. To test the efficacy of this, we apply the PercepGuard [33] integrity defense to our case studies in Section 7. PercepGuard ensures that all bounding boxes in a frame are spatio-temporally consistent with their reported class. We do not optimize our created bounding boxes for spatio-temporal consistency; thus, PercepGuard is able to detect our attack. However, it is only able to run *after* the object detection pipeline finishes processing the 0.27 second latency. Thus, by the time PercepGuard detects the attack, it is already too late to avoid a collision. This vulnerability is shared by other spatiotemporal consistency based defenses, such as PhySense [56], as their threat models focus on integrity attacks rather than availability ones. Overall, traditional integrity defenses requiring perception results to operate are not sufficient to evade latency attacks.

Quantization and Acceleration. On supported hardware, a victim may implement their model in an INT8 format and use TensorRT to accelerate their inference times in object detection [22]. To test the effectiveness of this against DETSTORM, we rebuilt our YOLOV5s model using TensorRTx [53] and serialized it in INT8 format, integrating it with NMS and OT. With this model, we achieved a recall and precision within 0.5% of the original model, and decreased the total runtime of ODT by 7.6% in benign conditions. However, when attacked by DETSTORM, our accelerated ODT pipeline is 8.1% *slower* than the original pipeline. We note that an average of 83% of latency comes from NMS and OT (Section 6.2), which contain non-differentiable comparisons that cannot be quantized or accelerated. Additionally, the reduced precision of INT8 models can create a cumulatively increasing overlap in detection bounds, increasing the strain on the later parts of the perception pipeline.

Sensor Fusion/Other Sensors. We focus on camera-based perception. In certain domains (e.g., surveillance and certain autonomous driving software [13]), only camera sensors are used for perception, whereas in other domains, additional sensors such as LiDAR and radar may be used. However, our Autoware case study shows that when employing multiple sensors, the victim may still experience unsafe physical consequences (e.g., unsafe braking) as long as the camera-based perception is compromised. Latency attacks may still be conducted against other sensors (e.g., LiDAR [30]) in tandem with a DETSTORM attack against the camera, and ambient noise such as rain and fog (for LiDAR) or electromagnetic interference (for radar) can degrade the performance of these sensors significantly, deepening consequences for the attacked victim.

Robust Perception against Latency. Based on our evaluation, we believe that the best way to mitigate latency effects is to improve the time complexity of both OT and NMS. Current approaches for both require between $O(n^2)$ and $O(n^3)$, which is the root cause of latency. If the complexity for both could be optimized to $O(\log(n))$ or smaller, it is likely that latency effects will be reduced. Once latency effects are mitigated, existing integrity-based defenses can be used to handle the mass creation effect and avoid physical consequences.

10. Conclusions

We present DETSTORM, a new physically-launchable latency attack against camera-based perception pipelines. DETSTORM causes latency effects by creating a large number of objects on physically perturbable surfaces, slowing NMS and OT to delay final perception results. In order to maximize the number of objects created, DETSTORM takes a greedy approach and maximizes the amount of created objects for each "zone" in the image containing a single, distinct class. We generated the perturbations using a new loss function that optimizes 4 separate objectives, generalizing attacks to several NMS approaches with no loss in attack effectiveness. We evaluated the effectiveness of our attack on both BDD100K and real-world experiments, and measured transferability to other perception algorithms and efficacy under different attack parameters. DETSTORM is able to cause a 500% to 2000% increase to the number of objects processed by camera-based perception, delaying results by up to 8.1 seconds and causing physical consequences for systems like Autoware.

Acknowledgments

We thank our shepherd and the anonymous reviewers for their valuable suggestions. This work has been partially supported by the National Science Foundation (NSF) through grants CNS-2144645 and IIS-2229876, and the Army Research Office (ARO) under grant W911NF2110320. The views presented in this paper are those of the authors only. We would like to thank Dr. Yiheng Feng, Wangzhi (George) Li, and Tianheng Zhu for loaning us their DataSpeed autonomous vehicle and assisting us in running the real-world experiments.

References

[1] A. Athalye, L. Engstrom, A. Ilyas, and K. Kwok, "Synthesizing robust adversarial examples," in *International conference on machine learning*. PMLR, 2018, pp. 284–293.

[2] Baidu, "Apollo," https://github.com/ApolloAuto/apollo.

[3] K. Bernardin and R. Stiefelhagen, "Evaluating multiple object tracking performance: the clear mot metrics," *EURASIP Journal on Image and Video Processing*, 2008.

[4] L. Bertinetto, J. Valmadre, J. F. Henriques, A. Vedaldi, and P. H. Torr, "Fully-convolutional siamese networks for object tracking," in *European conference on computer vision*. Springer, 2016.

[5] A. Bewley, Z. Ge, L. Ott, F. Ramos, and B. Upcroft, "Simple online and realtime tracking," in *IEEE Intl. Conf. on Image Processing*, 2016.

[6] A. Bochkovskiy, C. Wang, and H. M. Liao, "Yolov4: Optimal speed and accuracy of object detection," *CoRR*, vol. abs/2004.10934, 2020. [Online]. Available: https://arxiv.org/abs/2004.10934

[7] D. Bolya, C. Zhou, F. Xiao, and Y. J. Lee, "Yolact: Real-time instance segmentation," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2019.

[8] J. Cao, J. Pang, X. Weng, R. Khirodkar, and K. Kitani, "Observation-centric sort: Rethinking sort for robust multi-object tracking," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 9686–9696.

[9] Y. Cao, N. Wang, C. Xiao, D. Yang, J. Fang, R. Yang, Q. Chen, M. Liu, and B. Li, "Invisible for both camera and lidar: Security of multi-sensor fusion based perception in autonomous driving under physical-world attacks," in *IEEE Symposium on Security and Privacy*, 2021.

[10] E. Casas, L. Ramos, E. Bendek, and F. Rivas-Echeverria, "Yolov5 vs. yolov8: Performance benchmarking in wildfire and smoke detection scenarios," *Journal of Image and Graphics*, vol. 12, no. 2, 2024.

[11] E.-C. Chen, P.-Y. Chen, I. Chung, C.-r. Lee *et al.*, "Overload: Latency attacks on object detection for edge devices," *arXiv preprint arXiv:2304.05370*, 2023.

[12] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017.

[13] commaai, "openpilot," https://github.com/commaai/openpilot, 2021.

[14] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.

[15] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "Carla: An open urban driving simulator," *1st Conference on Robot Learning*, 2017.

[16] Y. Du, Z. Zhao, Y. Song, Y. Zhao, F. Su, T. Gong, and H. Meng, "Strongsort: Make deepsort great again," *IEEE Transactions on Multimedia*, 2023.

[17] K. Eykholt, I. Evtimov, E. Fernandes, B. Li, A. Rahmati, F. Tramèr, A. Prakash, T. Kohno, and D. Song, "Physical adversarial examples for object detectors," in *USENIX Conf. on Offensive Technologies*, 2018.

[18] Federal Highway Administration, "Driver, vehicle, and roadway characteristics related to driving in wet weather," https://safety.fhwa.dot.gov/speedmgt/ref_mats/fhwasa12022/chap_2.cfm, U.S. Department of Transportation, 2012, accessed: 11-Nov-2024.

[19] M. Foruhandeh, Y. Man, R. Gerdes, M. Li, and T. Chantem, "Simple: Single-frame based physical layer identification for intrusion detection and prevention on in-vehicle networks," in *Proceedings of the 35th Annual Computer Security Applications Conference*, 2019.

[20] GitHub, "Github object detection," https://github.com/topics/object-detection, 2023, [Online; Accessed 29-November-2023].

[21] L. Guo, L. Li, Y. Zhao, and Z. Zhao, "Pedestrian tracking based on camshift with kalman prediction for autonomous vehicles," *International Journal of Advanced Robotic Systems*, 2016.

[22] S. Hirose, N. Wada, J. Katto, and H. Sun, "Research and examination on implementation of super-resolution models using deep learning with int8 precision," in *International Conference on Artificial Intelligence in Information and Communication (ICAIIC)*. IEEE, 2022.

[23] Y. Hu, J. Yang, L. Chen, K. Li, C. Sima, X. Zhu, S. Chai, S. Du, T. Lin, W. Wang *et al.*, "Planning-oriented autonomous driving," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 17 853–17 862.

[24] D. Inc., "Dataspeed home," https://www.dataspeedinc.com/, 2024, [Online; Accessed 29-October-2024].

[25] B. Jiang, S. Chen, Q. Xu, B. Liao, J. Chen, H. Zhou, Q. Zhang, W. Liu, C. Huang, and X. Wang, "Vad: Vectorized scene representation for efficient autonomous driving," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023.

[26] G. Jocher, A. Stoken, J. Borovec, L. Changyu, A. Hogan, L. Diaconu, J. Poznanski, L. Yu, P. Rai, R. Ferriday *et al.*, "ultralytics/yolov5: v3. 0," *Zenodo*, 2020.

[27] S. Kato, S. Tokunaga, Y. Maruyama, S. Maeda, M. Hirabayashi, Y. Kitsukawa, A. Monrroy, T. Ando, Y. Fujii, and T. Azumi, "Autoware on board: Enabling autonomous vehicles with embedded systems," in *ACM/IEEE International Conference on Cyber-Physical Systems (ICCPS)*, 2018.

[28] S.-C. Lin, Y. Zhang, C.-H. Hsu, M. Skach, M. E. Haque, L. Tang, and J. Mars, "The architectural implications of autonomous driving: Constraints and acceleration," *SIGPLAN Not.*, vol. 53, no. 2, p. 751–766, mar 2018.

[29] C. Liu, H. Li, S. Wang, M. Zhu, D. Wang, X. Fan, and Z. Wang, "A dataset and benchmark of underwater object detection for robot picking," in *2021 IEEE international conference on multimedia & expo workshops (ICMEW)*. IEEE, 2021, pp. 1–6.

[30] H. Liu, Y. Wu, Z. Yu, Y. Vorobeychik, and N. 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*, 2023.

[31] Y.-L. Liu, J. Wang, X. Chen, Y.-W. Guo, and Q.-S. Peng, "A robust and fast non-local means algorithm for image denoising," *Journal of computer science and technology*, 2008.

[32] C. Ma, N. Wang, Q. A. Chen, and C. Shen, "Slowtrack: Increasing the latency of camera-based perception in autonomous driving using adversarial examples," *arXiv preprint arXiv:2312.09520*, 2023.

[33] Y. Man, R. Muller, M. Li, Z. B. Celik, and R. Gerdes, "That Person Moves Like A Car: Misclassification Attack Detection for Autonomous Systems using Spatiotemporal Consistency," in *USENIX Security Symposium*, 2023.

[34] G. Mathur, D. Somwanshi, and M. M. Bundele, "Intelligent video surveillance based on object tracking," in *International Conference and Workshops on Recent Advances and Innovations in Engineering (ICRAIE)*, 2018.

[35] R. Muller, Y. Man, Z. B. Celik, M. Li, and R. Gerdes, "Physical Hijacking Attacks against Object Trackers," in *ACM Conference on Computer and Communications Security (CCS)*, 2022.

[36] B. Nassi, Y. Mirsky, D. Nassi, R. Ben-Netanel, O. Drokin, and Y. Elovici, "Phantom of the adas: Securing advanced driver-assistance systems from split-second phantom attacks," in *ACM Conference on Computer and Communications Security (CCS)*, 2020.

[37] A. Nussberger, H. Grabner, and L. Van Gool, "Aerial object tracking from an airborne platform," in *international conference on unmanned aircraft systems (ICUAS)*, 2014.

[38] NVIDIA, "Geforce rtx 3080 family," https://www.nvidia.com/en-us/geforce/graphics-cards/30-series/rtx-3080-3080ti/, 2021, [Online; Accessed 5-February-2024].

[39] ——, "Jetson community projects," https://developer.nvidia.com/embedded/community/jetson-projects, 2024, [Online; Accessed 29-October-2024].

[40] Optoma, "Optoma lv130 ultra-portable projector," https://www.projectorcentral.com/Optoma-LV130.html, 2021, [Online; Accessed 5-February-2024].

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

[42] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *CoRR*, vol. abs/1804.02767, 2018.

[43] A. Schneider, A. Robinson, C. Grimm, and N. T. Fitter, "How do starship robots affect everyday campus life? an exploratory posting board analysis and interview-based study," in *2024 33rd IEEE International Conference on Robot and Human Interactive Communication (ROMAN)*. IEEE, 2024.

[44] A. Shapira, A. Zolfi, L. Demetrio, B. Biggio, and A. 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*, 2023.

[45] A. J. Shepley, G. Falzon, P. Kwan, and L. Brankovic, "Confluence: A robust non-iou alternative to non-maxima suppression in object detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2023.

[46] A. Sheply, "Confluence," 2021. [Online]. Available: https://github.com/ashep29/confluence

[47] B. Shuai, A. Berneshawi, X. Li, D. Modolo, and J. Tighe, "Siammot: Siamese multi-object tracking," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2021.

[48] M. Treu, T.-N. Le, H. H. Nguyen, J. Yamagishi, and I. Echizen, "Fashion-guided adversarial attack on person segmentation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021.

[49] D. Wang, C. Li, S. Wen, Q.-L. Han, S. Nepal, X. Zhang, and Y. Xiang, "Daedalus: Breaking nonmaximum suppression in object detection via adversarial examples," *IEEE Transactions on Cybernetics*, 2022.

[50] T. Wang, Z. Meng, M. Xu, R. Han, and H. Liu, "Enabling high frame-rate uhd real-time communication with frame-skipping," in *Proceedings of the 3rd ACM Workshop on Hot Topics in Video Analytics and Intelligent Edges*, 2021, pp. 19–24.

[51] N. Wojke, A. Bewley, and D. Paulus, "Simple online and realtime tracking with a deep association metric," in *IEEE International Conference on Image Processing (ICIP)*. IEEE, 2017, pp. 3645–3649.

[52] C. Xiang, A. N. Bhagoji, V. Sehwag, and P. Mittal, "{PatchGuard}: A provably robust defense against adversarial patches via small receptive fields and masking," in *USENIX Security Symposium*, 2021.

[53] W. Xinyu, "Tensorrtx," https://github.com/wang-xinyu/tensorrtx, 2024, [Online; Accessed 04-November-2024].

[54] B. Yan, D. Wang, H. Lu, and X. Yang, "Cooling-shrinking attack: Blinding the tracker with imperceptible noises," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020.

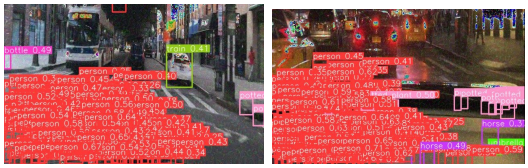
[55] F. Yu, W. Xian, Y. Chen, F. Liu, M. Liao, V. Madhavan, and T. Darrell, "Bdd100k: A diverse driving video database with scalable annotation tooling," *Computer Vision and Pattern Recognition Conference (CVPR)*, 2020.

[56] Z. Yu, A. Li, R. Wen, Y. Chen, and N. Zhang, "Physense: Defending physically realizable attacks for autonomous systems via consistency reasoning," in *ACM Conference on Computer and Communications Security*, 2024.

[57] Z. Zhang, S. Huang, X. Liu, B. Zhang, and D. Dong, "Adversarial attacks on yolact instance segmentation," *Computers & Security*, vol. 116, p. 102682, 2022.

[58] Z. Zheng, "Distance-iou loss in darknet," 2019. [Online]. Available: https://github.com/Zzh-tju/DIoU-darknet

[59] Z. Zheng, P. Wang, W. Liu, J. Li, R. Ye, and D. Ren, "Distance-iou loss: Faster and better learning for bounding box regression," in *Proceedings of the AAAI conference on artificial intelligence*, 2020.



(a) (b)

Figure 13: Phantom Sponge "universal" perturbations created on primarily dark-colored data will only affect dark surfaces.

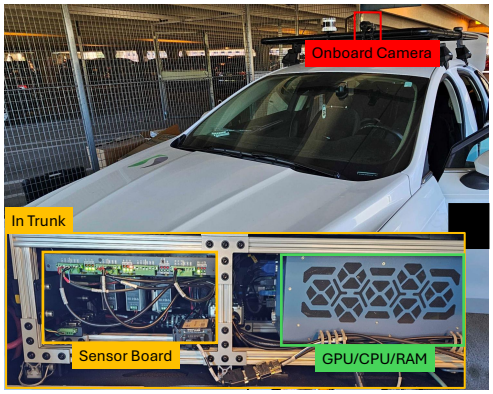


Figure 14: Our real-world DataSpeed autonomous vehicle used for experiments.

Appendix A. Zone Classes

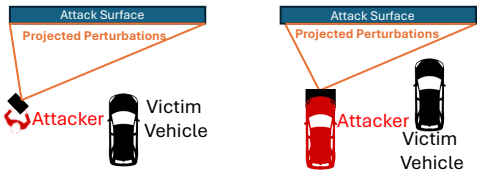
Most Frequent Zones. We used 119 perturbable classes extracted from the BDD100K dataset to compute individual patches on. The 10 most common perturbable zones, by percentage of total appearances, are represented in Table 6. As can be expected, objects like roads, cars, and buildings make up a large portion, together accounting for over a quarter of the total perturbable zones in BDD100K. Other traffic objects, such as signboards, walls, and fences, are also in the top 10. Pedestrians are the seventh most common class, representing just over 4% of total perturbable zones.

Table 6: Frequency of top 10 classes extracted from the BDD100K dataset.

Class	Frequency (%)
Road	9.7516%
Car	9.7506%
Building	9.4380%
Sidewalk	8.007%
Signboard	6.9074%
Wall	6.5573%
Person	4.147%
Fence	4.0415%
Ceiling	3.7387%
Pole	3.7071%

Table 7: Attack performance of Phantom Sponges [44] during simulated physical experiments. DETSTORM outperforms Phantom Sponges in all areas, causing between 55.36% and 410.29% more latency depending on hardware.

Hardware	ROI-O (Avg/Max)	ROI-L (Avg/Max)	Max Latency (s)
Jetson Nano		33.02 / 42.57	2.83
Jetson TX2		30.28 / 35	0.95
RTX 2080 Ti	4.87 / 20.43	11.13 / 14.87	0.44
RTX 3070		1.12 / 2.12	0.36
RTX 3080		5.08 / 50.65	0.18



(a) Human-carried (b) Vehicle-mounted

Figure 15: To effect perturbations, an attacker can either (a) carry the projector on foot, adjusting to match the victim's visible surfaces, or (b) mount a projector to a vehicle and follow alongside the victim.

Appendix B. YOLO Architectures

The YOLO (You Only Look Once) model [41] represents a pivotal advancement in the field of single-stage object detection, offering a unique approach that contrasts with traditional two-stage detectors. YOLO frames object detection as a single regression problem, directly mapping from image pixels to bounding box coordinates and class probabilities. This holistic approach allows YOLO to achieve remarkable speed and efficiency, making it highly suitable for real-time applications. Since its introduction, YOLO has undergone several iterations, each improving upon its predecessor in terms of accuracy, speed, and model complexity.

YOLOv3 [42] introduced several enhancements over its predecessors, including the use of three different scales to detect small, medium, and large objects. It also incorporated the use of logistic regression for 'objectness' prediction, making it more accurate in distinguishing foreground objects from the background. Additionally, YOLOv3 employed a deeper and more complex architecture based on Darknet-53, significantly improving its ability to recognize a wide variety of objects.

YOLOv4 [6] further refined the model's performance and efficiency, making it more accessible for use on a broader range of hardware. It introduced several new concepts, such as the use of the CSPDarknet53 backbone for reducing model complexity while maintaining high performance, and the integration of spatial pyramid pooling and PANet for more effective feature aggregation from different levels of the network. YOLOv4 also emphasized the importance of data augmentation techniques like CutMix and Mosaic, along with the use of advanced optimization methods such as CIOU loss, to enhance the model's accuracy and robustness.

YOLOv5 [26], developed and open-sourced by Ultralytics, marks a significant milestone in the YOLO evolution, focusing on simplicity, speed, and performance. It offers various model sizes (YOLOv5s, YOLOv5m, YOLOv5l, and YOLOv5x) to cater to different computational and accuracy needs. These variants differ primarily in their depth and width, affecting their speed and accuracy trade-offs.

Appendix C. Denoising Defense Parameters

We attempted 7 approaches to denoising: inverting pixel values, mean/median/gaussian blurring, contrast and brightness alterations, bilateral filtering, and Fast non-local means denoising [31]. To remove created objects from a non-adaptive attacker, we found the following methods to be most effective: (1) a median blur with a kernel size of 3, (2) gaussian blur with a kernel size of $(7, 3)$, and (3) a bilateral filter with a pixel diameter of 5, σ_{color} of 130 and σ_{space} of 90. However, if we apply these same values to the loss function of DETSTORM's perturbation generation, denoising is no longer effective.

Appendix D.
Meta-Review

The following meta-review was prepared by the program committee for the 2025 IEEE Symposium on Security and Privacy (S&P) as part of the review process as detailed in the call for papers.

D.1. Summary

The paper introduces a novel physical latency attack on camera-based object detection and tracking in autonomous systems. This attack exploits the high computational complexity of non-maximum suppression and object tracking algorithms by generating a large number of fake objects on perturbable surfaces, which significantly increases perception latency. It optimizes perturbations to maximize object generation and dynamically adapts to environmental changes. Evaluated through simulations and real-world experiments, the attack effectively delays perception and increases object counts across various object detection and tracking models and hardware platforms.

D.2. Scientific Contributions

- Provides a Valuable Step Forward in an Established Field

D.3. Reasons for Acceptance

1) The paper identifies a novel, physically realizable attack that can delay perception in camera-based object detection and tracking systems.
2) The paper addresses the challenges of real-time computation and the transferability of patch-based latency attacks while adapting to dynamic real-world environments.

D.4. Noteworthy Concerns

1) Reviewers questioned the extent to which the proposed attack applies to other object detection models, datasets, environments, and real-world conditions. While the authors defer evaluation on additional models to future work, they argue that their experiments on the BDD100K dataset and real-world scenarios demonstrate the attack's generalizability.
2) One reviewer raised concerns about evaluating the attack on low-power hardware platforms, questioning its relevance to high-end systems. The authors justify their choice by emphasizing that embedded platforms like the Jetson Nano/TX2 are widely used in low-cost autonomous vehicles and are crucial for assessing attacks on resource-constrained systems.

Appendix E.
Response to the Meta-Review

In the following, we detail our response to each of the noteworthy concerns.

1) In this paper, we focused on YOLOv5 due to its prevalence and superior real-time performance. Because latency effects primarily stem from the $O(n^2)$ to $O(n^3)$ bottlenecks in NMS and OT, OD architectures are less important to our approach. We defer evaluation on additional models to future work due to the extensive amount of time required to generate perturbation dictionaries on a new model; initial experiments in attacking Faster-RCNN show $3.32\times$ higher average ROI-L despite Faster-RCNN's higher baseline accuracy.
2) We split our hardware into two camps. Our embedded platforms like our Jetson devices are essential for examining the effect of latency in low-cost AVs or resource-constrained autonomous systems, while our more powerful hardware such as our RTX devices represent more powerful autonomous vehicle hardware, including the 2080 Ti mounted to our DataSpeed autonomous vehicle. Extremely advanced hardware (e.g., RTX 5090) or specialized AD hardware (e.g., Nvidia DRIVE) exist that may experience different ROI-L and max latencies. However, we consider these extreme high-performance devices to be outside the scope of our work, which is meant to be a general look at latency attacks performed in physical environments.