ALLUDE: A Unified Evaluation System for Configurable Attacks in Differentiable Environments

Mansi Phute[1] Alexander Greenhalgh[1] Matthew Hull[1] Haoran Wang[1]
Alec Helbling[1] ShengYun Peng[1] Elliott Faa[1] Willian Lunardi[2] Martin Andreoni[2]
Wenke Lee[1] Duen Horng Chau[1]

[1] Georgia Institute of Technology, USA
[2] Technological Innovation Institute, UAE

Figure 1. ALLUDE closes long-standing gaps in adversarial attack evaluation, uniting Unreal Engine photorealism with end-to-end differentiable rendering in a single cross-platform system. **(A)** fully configurable scenes, objects, attacks, lighting, and weather; **(B)** a rich set of built-in camera trajectories, from tight orbits to sweeping flyovers; and **(C)** a modernized library of CARLA maps and models, freed from legacy Unreal Engine versions for immediate reuse in current and future research.

Abstract

Adversarial attacks against vision models like object detectors are often evaluated under limited conditions, leaving their performance under-characterized. Bridging simulation and differentiable rendering enables more robust, end-to-end evaluation of these adversarial attacks, yet there is no easy-to-use, unified system that offers a rich set of customizable configurations for adversarial attacks across multiple scenes, objects, environmental and lighting conditions, and camera trajectories. We present ALLUDE*, which addresses these gaps, offering first-of-its-kind evaluation capabilities across Linux and Windows. We comprehensively demonstrate* ALLUDE*'s evaluation breadth through a two-pronged strategy: (1) using Latin Hypercube Sampling, we draw a representative subset from 5,400 configurations spanning 10 scene–object pairs, 9 weather conditions, 4 optimizers, 5 camera trajectories, and 3 detection models; (2) we stress-test existing attacks (CAMOU, RAUCA, FCA) under diverse weather conditions and continuous camera trajectories, revealing degradation of attack success across every attack, exposing evaluation gaps in prior work. Through* ALLUDE*'s end-to-end differentiable rendering, adversarial attacks can be optimized against shifting real-world deployment conditions. Our cross-platform open source code is available at* https://anonymous.4open.

Figure 2. Existing adversarial attack evaluation "superimposes" the adversarial pattern onto the pre-determined placeholder region (here: green patch). Any overlapping objects (here: cars, shadows), are identified and replaced using their metadata. While this appears as true pattern replacement, the adversarial pattern is not rendered in the environment, thus preventing accurate evaluation.

science/r/ALLUDE-A3F6.

1. Introduction

Robustness of vision models is an important area of research, providing the backbone to many safety-critical applications that depend on reliable and consistent visual predictions from modern detectors. Across adversarial attack literature, targeted perturbations in images on both pixel-space and 3D textures result in misclassification and detector-suppression attacks on a diverse set of objects, including vehicles [8, 25–27, 35, 36], traffic signs [2], and pedestrians [29, 32]. Adversarial patterns, or "patches" are typically generated by iterative optimization, where noise is added to an image and iteratively optimized against a model's object-class predictions to induce classification or detection failure. Researchers often use photorealistic simulations to better understand how these patches perform in physical settings and in varying environments and weather conditions that would not be possible otherwise. Currently, patches are not inserted into the simulation but rather added onto the rendered scenes during postprocessing [29]. During scene rendering, the area adversarial patches will occupy is filled with a colored placeholder, like a "green screen" in Fig. 2 that is visually distinct from its environment. The corners of this plain patch are then calculated in postprocessing with the optimized adversarial patch superimposed on the image using PyTorch transformations [29].

This postprocessing approach leads to two key issues: (1) The dynamic coordinate detection required for depositing patches onto the placeholder texture needs to be consistent across frames in the same scene; in practice, this is rarely the case. An error of a couple pixels can lead to the inserted adversarial patch appearance differing drastically across frames (Fig. 2). Under the intended threat model, where an adversarial patch is deployed in the physical world, the patch's appearance should be consistent across views in the same scene. Frame-to-frame inconsistency violates this assumption, and causes evaluation approaches to overestimate attack strength and underestimate defense robustness. (2) The optimized pattern does not respond properly to environmental constraints, such as correct lighting or physics, which can cause the adversarial patch to fail when inserted into photorealistic simulation [29].

As summarized in Table 1, existing attacks often couple a specific texture parameterization, differentiable renderer, target detector, and simulator into a fixed attack pipeline; attacks are typically demonstrated on a single object domain in isolation, rendered through standalone neural renderers, optimized against legacy detectors, and run only on the simulator's native platform. No existing system offers engine-faithful rendering, configurable weather and camera trajectories, or cross-platform support within one generalized framework. This fragmentation makes attacks difficult to compare or evaluate under diverse, deployment-relevant conditions that matter for safety-critical vision applications.

To close these long-standing gaps in adversarial attack evaluation, we introduce ALLUDE (Fig. 1), a system that allows easy cross-platform development on Linux along with Windows systems, making adversarial research more accessible and easier to integrate with existing research pipelines (Table 1, bottom row). ALLUDE enables easy customization of environments and provides environment settings for various weather and lighting conditions. In addition, we modernize a broad set of CARLA [4] assets by uncoupling them from their original UE version. These CARLA environments and assets are often used in adversarial attacks and data generation. However, they are tied to the UE version CARLA uses, preventing researchers from using these assets in newer UE versions. By separating CARLA objects from their UE versions, we allow greater customization and improve usability for the adversarial machine learning research community. The main contributions of ALLUDE are:

1. **ALLUDE: Easy-to-use, and cross-platform system with a rich set of customizable configurations for adversarial attacks, scenes, objects, environmental and lighting conditions, and camera trajectories** (Fig. 1; Sec. 3). ALLUDE readily integrates into existing research and evaluation pipelines (demonstrated in Sec. 4.2), offering the first-of-its-kind evaluation capabilities across Linux and Windows.
2. **Two-pronged evaluation strategy for comprehensive testing of ALLUDE:** We demonstrate ALLUDE's generalization through prototypical attack settings based on prior literature, sampling a representative subset of $N = 100$ configurations through *Latin Hypercube Sampling* from over 5,400 possible configurations, isolating the effects of detector model, environmental conditions and camera trajectories on attack performance.
3. **Open source implementation:** Our code, source data, and generated data are all open source at https://

Table 1. ALLUDE (bottom row) is the only adversarial attack framework with full marks in every object domain and system capability, uniquely combining arbitrary geometry, engine-faithful rendering, configurable environments, cross-platform support, and open-source release. The object-domain columns mark the classes each method is demonstrated on; "Other" denotes classes beyond the three often-studied types (vehicle, pedestrian, traffic sign); ALLUDE supports arbitrary simulation objects. **Capabilities**: ● full, ◐ partial, ○ none.

				Object domains				**Capabilities**				
Method	**Param.**	**Renderer**	**Sim.**	*Vehicle*	*Pedestrian*	*Traffic sign*	*Other*	Arbitrary geometry	Engine-faithful rendering	Config. environ.	Cross-platform	Open source
CAMOU [35]	Tiled 2D	Clone network	UE4	●	○	○	○	◐	◐	◐	○	●
DAS [27]	Partial 2D	Neural renderer	CARLA (UE4)	●	○	○	○	◐	◐	○	○	●
FCA [26]	Full UV	Neural renderer	CARLA (UE4)	●	○	○	○	◐	◐	◐	○	●
ACTIVE [25]	Triplanar	Neural renderer	CARLA (UE4)	●	○	○	○	●	◐	◐	○	○
RAUCA [36]	Full UV	Neural renderer	CARLA (UE4)	●	○	○	○	◐	●	◐	○	●
Adv. T-shirt [32]	2D patch	—	—	○	●	○	○	○	○	○	○	●
Invis. Cloak [31]	2D patch	—	—	○	●	○	○	○	○	○	○	●
ShapeShifter [2]	2D pert.	—	—	○	○	●	○	○	○	○	○	●
ALLUDE (Ours)	Full UV	**Mitsuba 3** [9]	**UE5**	●	●	●	●	●	●	●	●	●

anonymous.4open.science/r/ALLUDE-A3F6. We also make the CARLA assets uncoupled from CARLA's Unreal Engine (UE) version available at https://huggingface.co/datasets/allude-occluded/real-carla-assets-anon.

2. Related Work

Simulation in Adversarial Machine Learning. Simulation is an important aspect of optimizing adversarial attacks to test systems [20, 30, 34]. Unreal Engine is the backbone of many simulators including CARLA [4] and Airsim [24]. It is selected for its high fidelity and photorealistic rendering. However, Unreal Engine is non-differentiable, making direct optimization impossible and preventing it from seamlessly fitting into the adversarial optimization pipeline. This also persists in all simulation benchmarks derived from them [5, 12, 19, 21, 23, 28]. Adversarial approaches rely on approximations such as post-hoc superimposition of adversarial textures onto rendered images, which can leave errors (Fig. 2). Inserting a texture in simulation breaks attacks [33] and affects attack transferability [20].

Differentiable Rendering. Differentiable rendering allows us to merge traditional computer graphics and gradient-based optimization. It allows backpropagation of gradients to update geometry, lighting, and textures in a scene [11, 13, 16]. In adversarial machine learning differentiable rendering is used to optimize textures in varying environmental conditions [1, 8, 10, 14, 36]. However, a limitation of these methods, such as Mitsuba and PyTorch3D, is that they do not support simulation or complex weather and require specialized knowledge to use.

Bridging Simulation and Differentiable Rendering. Adversarial attacks span multiple object domains, such as vehicles [8, 25–27, 35, 36], traffic signs [2], and pedestrians [29, 32]. As shown in Table 1, each combines texture parameterization, differentiable renderer, and target detector into a method-specific attack pipeline. Vehicle camouflage methods utilize differentiable rendering over different 2D texture representations, tying their evaluation to specific renderers and legacy detection models for consistency with existing literature. Pedestrian and sign adversarial attacks avoid 3D differentiable rendering entirely, operating on only 2D patches that suffer from the same issues shown in Fig. 2. These discrepancies between individual attack pipelines limit cross-framework reproducibility, where a texture optimized through one method's renderer and UV mapping cannot be evaluated without manual reunification of their attack pipeline component. Recent work bridges differentiable rendering with photorealistic simulation, enabling end-to-end optimization of adversarial textures on arbitrary 3D objects within controllable environments [22]. This establishes that in-simulation attacks are *feasible*, but stops at demonstrating the capability — on Windows, a handful of objects and conditions, and a single detector — leaving open the empirical question this capability makes answerable, when do physical attacks actually succeed, and when does success measured under narrow conditions fail to generalize? ALLUDE turns this capability into the first systematic study of that question, contributing (i) the cross-platform infrastructure (headless Linux) needed to run it at scale, (ii) a standardized configuration space and quantitative characterization of attack behavior across it, and (iii) a released, version-decoupled CARLA asset library so the evaluation is reusable.

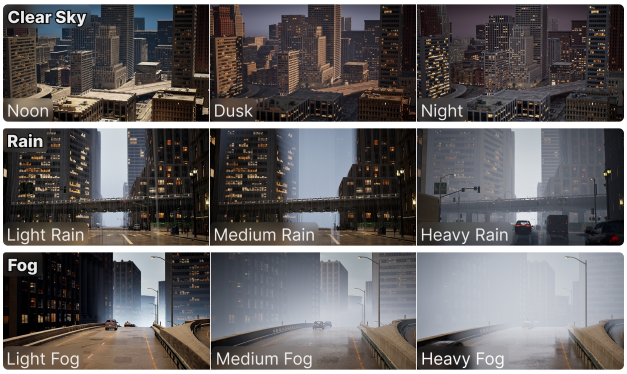

Figure 3. ALLUDE offers easy weather and lighting customization for stress-testing attacks. Users can control lighting to reflect time of the day, and add rain and fog with varying intensity levels.

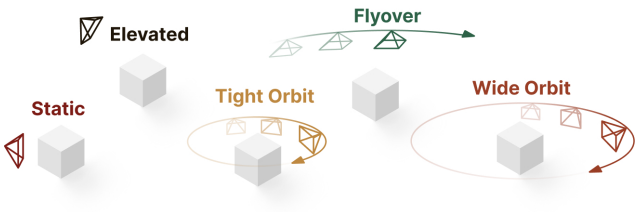

Figure 4. ALLUDE's 5 camera trajectories (static frontal, static elevated, flyover, tight orbit, wide orbit) optimize and evaluate adversarial textures across viewpoints that existing approaches miss.

3. System

ALLUDE supports a wide range of evaluation configurations, with presets for 10 objects, 10 scenes, 5 trajectories, 9 weather conditions, 4 optimizers, and 3 detectors (Sec. 3.1). ALLUDE enables optimizing adversarial perturbations of arbitrarily shaped objects on Linux and Windows systems without requiring a graphical user interface (GUI) (Sec. 3.2). This brings UE's photorealistic rendering to headless Linux systems such as HPC clusters and AWS servers, enabling large-scale, distributed evaluations of adversarial attacks in realistic simulation environments. It also allows integration of CARLA scenes and assets in experiments while eliminating their dependency on legacy UE versions and the CARLA stack (Sec. 3.3).

3.1. Easy to Use & Customize

3.1.1. Diverse Evaluation Configurations

Weather: A key advantage of using simulation in evaluating adversarial attacks is the wide variety of environments and weather conditions that can be replicated, allowing greater control over the experimental setup. ALLUDE takes advantage of Unreal Engine's suite of plugins and tools to include experiments with a variety of lighting and weather conditions. ALLUDE spans 9 weather conditions across three categories: clear skies at three times of day (*noon*, *dusk*, and *night*), three intensity levels each of rain and fog (*light*, *medium*, and *heavy*) (Fig. 3). Full Unreal Engine per-condition weather and lighting parameters are provided in Sec. S1.1.

Trajectories: ALLUDE provides 5 camera trajectories: *static frontal*, *static elevated*, *flyover*, *tight orbit*, and *wide orbit*, designed to be representative of how cameras capture objects in real-world deployment settings. These trajectories are illustrated in Fig. 4. Unlike prior adversarial simulation benchmarks [15] that sample from static viewpoints, ALLUDE supports creation of continuous camera trajectories. Detailed per-trajectory camera parameters and object-specific scale transformations are given in Sec. S1.3.

Optimizers: ALLUDE implements four gradient-based adversarial optimizers: FGSM [7], PGD [17], Auto-PGD, and AutoAttack [3]. While originally designed for ℓ_∞-bounded attacks on image classifiers (2D pixel space), in ALLUDE's differentiable-rendering setting they serve as the optimizer component of the 3D texture-space attack. Per-optimizer implementation details and hyperparameters are included in Sec. S1.2.

Detectors: ALLUDE provides built-in support for 3 popular detection models: YOLOv11, Faster R-CNN, and DETR, with the option to add in support for more. The selected models cover modern one-stage, two-stage detectors, and transformer-based architectures, enabling the framework's detector capabilities to be explored across many evaluation settings. For example, the Latin hypercube analysis (Sec. 4.1) easily varies the detector as one of the configurations.

3.1.2. Ease of Use

ALLUDE provides a comprehensive suite of options for configuring the various customizations described in Sec. 3.1.1. It runs as a single-line CLI command on both Linux and Windows, enabling quick adjustments to experiments. All parameters including attack hyperparameters, weather, and environment can be modified directly through the CLI.

3.2. First-Of-Its-Kind Cross-Platform Evaluation

ALLUDE provides researchers with cross-platform Linux and Windows support not previously available. Historically, Unreal Engine has been used primarily on Windows due to its more intuitive graphical user interface (GUI) and stronger user support. While a Linux distribution for Unreal Engine is available, it is not widely used. For data generation and attack evaluation in adversarial research, Linux compatibility is necessary to integrate evaluation capabilities into researchers' current research pipelines, which heavily favor headless Linux-based GPU servers. The headless setting of these GPU servers introduces further com-

plications in rendering photorealistic simulations: Windows applies sRGB gamma encoding through the display swapchain automatically, while headless Linux saves linear-light pixels without that conversion. However, it is possible to force sRGB output encoding in Linux, and we provide a docker configuration in our code-base for reproducibility.

3.2.1. Cross-platform Runtime Stability Engineering

When a Python callback receives a borrowed UStruct argument and releases the Global Interpreter Lock (GIL) for an extended period, which occurs in ALLUDE when a multi-second subprocess call is run, the garbage collector in Unreal Engine traverses the wrapper after its backing memory has been reclaimed. This results in a fatal Unreal Editor crash. We mitigate this with a three-part fix extending wrapper validity, pinning object references, and modifying garbage collection mechanisms. The complete patch description is provided in the supplementary Sec. S1.4.

3.2.2. Isolating independent C++ backbones

Running a real-time game engine with a differentiable renderer and a deep-learning framework in a single Python process faces three independent toolchain conflicts on Linux due to C++ conflicts. We therefore separate these components into two cooperating simultaneous processes. The engine drives scene capture in the parent process; a child process performs gradient computation and texture optimization. Additional details about this implementation can be found in the supplementary Sec. S1.5.

3.3. Decoupling CARLA assets from Legacy Unreal

CARLA is a self-driving simulation platform built on Unreal Engine, widely used in data generation pipelines for adversarial attacks. It inherits many of UE's simulation capabilities but is very tightly coupled to a specific UE version, making it difficult to update with newer releases of Unreal Engine. Using or modifying CARLA assets in Unreal also requires installing CARLA from source, which is time consuming and tedious. In all experiments in Sec. 4.2 and Sec. 4.1, we use CARLA assets and environments upgraded to Unreal Engine 5.7 and uncoupled from CARLA backend dependencies. We open-source these modernized assets to enable researchers to customize CARLA assets easily. Additional details on modernizing these assets are provided in Sec. S1.7, and the assets are available at https://huggingface.co/datasets/allude-occluded/real-carla-assets-anon.

4. Evaluation

We evaluate ALLUDE's capabilities through two complementary approaches, (1) demonstrating the broad attack capabilities through sampling a subset of the 5,400 evaluation configurations through Latin Hypercube Sampling (LHS) to characterize how targeted attack object, optimizer, detector, trajectory, and weather conditions influence attack success across the full configuration space (Sec. 4.1); and (2) stress-testing ALLUDE's optimized adversarial textures in the weather and trajectory conditions specified in Sec. 3.1.1 on the Audi E-Tron vehicle (Sec. 4.2).

Metric: Across both studies we report adversarial mAP@50 (adv-mAP@50) of the target class, where lower values indicate a stronger attack. Prior physical attack literature commonly reports attack success rate (ASR), calculated as the fraction of frames where a target is no longer detected. This metric carries the assumption of perfect benign detection, yet several of our weather and trajectory conditions degrade benign detection: benign mAP@50 falls to 0.89 under the flyover trajectory (Table 4), and 14 of the 100 sampled LHS configurations are *indeterminate*, with the object undetected even without an attack (Sec. 4.1). To avoid conflating attack effect with environmental detection failure, we use mAP@50, which is the mean average precision of the target-class detections at an intersection-over-union (IoU) threshold of 0.5:

$$\text{adv-mAP@50} = \frac{1}{|C|} \sum_{c \in C} \text{AP}_c^{\text{IoU}=0.5}, \tag{1}$$

With diverse environmental conditions and a broad range of object classes, mAP@50 provides a single comparable measure of attack strength that remains meaningful even when benign object detection is imperfect.

4.1. Evaluating Diverse Configurations via Latin Hypercube Sampling

To characterize attack behavior across the broader configuration space, we sample $N = 100$ configurations through balanced Latin Hypercube Sampling (LHS) over 5 categorical factors: optimizer (4 types), scene-object pairs (10 levels), trajectory (5 levels), weather (9 levels), and detector architecture (3 levels) with a deterministic seed. The full factor space contains $5,400$ cells, with the $N = 100$ sample covering 1.85% of the space (Fig. 5). The balanced marginals allow estimation of each factor's main effect on mAP@50; resolving interaction effects would require additional sampling fidelity within the configuration space. Every cell uses identical attack hyperparameters ($\epsilon = 128$, $\alpha = \epsilon/8$, and attack-specific iteration counts). We perform an untargeted evasion attack. This experimental design samples a subset of the parameter space that is representative of the broader configurations enabled by ALLUDE, demonstrating the framework's extensibility to a wide variety of adversarial attack conditions. Further details of the LHS approach, including renders of all $N = 100$ scenes used in this analysis are provided in Sec. S2.

Scenes: Scenes are drawn from CARLA town and city en-

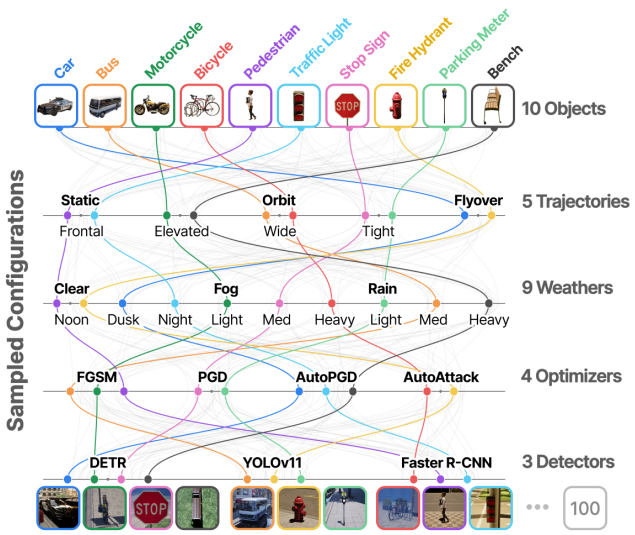

Figure 5. $N = 100$ configurations through balanced Latin Hypercube Sampling (LHS) over 4 optimizers, 10 scene-object pairs, 5 trajectories, 9 weather conditions, and 3 detectors.

vironments from the Unreal Engine Fab store. Of CARLA's 10 towns, Towns 8 and 9 are absent from the open-source release and are excluded. Towns 1 and 2 are near-duplicates: a small village and a small town, where a single scene is representative. Town 3 is an urban layout similar to Town 10, except with reduced asset fidelity. We select one scene from each of Towns 2, 4, 5, 6, and 7, and five scenes from Town 10. This selection balances two goals: (1) scene diversity, as Towns 4 and 6 have less urban layouts than the other open-source towns, and (2) scene quality, as Town 10 offers the most realistic urban environments in CARLA. Example scenes from each town are shown in Sec. S2.3.1.

Objects: The 10 evaluation objects are sourced from CARLA and drawn from common COCO categories, matching the training distribution of the detection models: car, bench, traffic light, person, parking meter, bicycle, bus, fire hydrant, motorcycle, and stop sign. The literature comparison uses the Audi E-tron vehicle for consistent comparison with adversarial attack literature. Full scene and object details are described in Sec. S2.3.2.

Attack outcome and model. We define an attack as *successful* when the adversarial texture suppresses the target-class detection score such that adv-mAP@50 < 0.5. Of the 100 cells, 14 are *indeterminate*, where the detector fails to find the benign object (benign mAP@50 $= 0$) due to weather and trajectory specific detector degradation. We exclude these, leaving $N = 86$ scored cells. Across these, ALLUDE successfully attacks $40/86$ (46.5%) of the sampled configurations, and measurably reduces detection confidence (≥ 0.10 mAP@50 drop) in 72.1% of cells. We fit an L2-regularized logistic regression of mAP@50 across these 5 factors, and measure each factor's contributions by the corresponding drop in model fit (McFadden pseudo-R^2 [18]) with its removal, with a likelihood-ratio test for significance (Table 3).

Table 2. Per-object attack outcomes across the N=86 scored LHS cells. *Suppressed* cells have adv-mAP@50 < 0.5; *Degraded* cells have a measurable confidence reduction (≥ 0.10 mAP@50 drop). Rates span the full range, from universally attackable (traffic light) to universally robust (fire hydrant).

	Suppressed (< 0.5)		**Degraded** (≥ 0.10)	
Object	**Cells**	**Rate**	**Cells**	**Rate**
Traffic light	8 / 8	100%	8 / 8	100%
Bus	8 / 10	80%	10 / 10	100%
Bicycle	5 / 8	63%	7 / 8	88%
Motorcycle	5 / 8	63%	8 / 8	100%
Bench	2 / 4	50%	4 / 4	100%
Car	4 / 8	50%	8 / 8	100%
Parking meter	4 / 10	40%	6 / 10	60%
Stop sign	3 / 10	30%	8 / 10	80%
Pedestrian	1 / 10	10%	2 / 10	20%
Fire hydrant	0 / 10	0%	1 / 10	10%
Overall	**40 / 86**	**46.5%**	**62 / 86**	**72.1%**

Object class dominates attack success. Target object class is the primary predictor of attack effectiveness, explaining 22% of regression model fit ($p = 0.002$). Attack success by object spans the full range (Table 2): traffic lights are suppressed in every non-indeterminate attack $(8/8)$ and buses in most $(8/10)$, whereas pedestrian $(1/10)$ and fire hydrant classes $(0/10)$ are the most difficult to attack. This variation arises from two distinct sources. (1) Prominent, texture dominated objects such as buses, traffic lights, and stop signs provide extensive adversarial surface and weaker shape priors [6], while pedestrians and fire hydrants are detected largely from silhouette and contours that are inherently more difficult for a surface texture to disrupt [32]. This robustness is especially pronounced for pedestri-

Table 3. Configuration factor contribution to adversarial attack success (adv-mAP@50 < 0.5) for ALLUDE, across N=86 sampled configurations. "Variance explained" is the drop in model fit (McFadden pseudo-R^2) when the factor is removed; "Significant" denotes a likelihood-ratio test at $p < 0.05$ (object p=0.002, trajectory p=0.020, all others $p > 0.50$).

Factor	**Variance explained**	**Significant?**
Target object	**22%**	**Yes**
Camera trajectory	10%	Yes
Weather	4%	No
Attack algorithm	2%	No
Detector	<1%	No

ans, where detection models are trained on a wide distribution of poses. (2) Objects differ in how their 2D UV map textures are rendered onto the 3D objects in Unreal. Because of these differences, the effective perturbation budget that reaches the detector varies by object even at a fixed ϵ. We characterize these per-object texture-handling differences in Sec. S2.3.2.

Viewpoint diversity drives attack robustness. Camera trajectory is the second-strongest factor, explaining 10% of model fit ($p = 0.020$). Attack success across the sampled configurations decreases with trajectory complexity, where larger distributions of viewing angles and camera distances require the adversarial texture to remain effective across a wider range of conditions. Static trajectories, are the easiest to attack (static-elevated 78%, static-frontal 41%). Orbit trajectories circle the object through 360°, and their difficulty is consistent with viewing distance: the tight-orbit attacks succeed in 65% of cells while the more distant wide-orbit succeeds in 45%. The detailed flyover trajectory combines wide angular coverage with varying distances and framing of the object, and is the hardest at 20%. Because this dependence on camera motion is only observable under continuous trajectories, static-viewpoint benchmarks [12, 15] cannot expose it, yet it is among the strongest determinants of attack effectiveness.

Optimized textures remain effective across weather. In the configuration space analysis, Weather explains 4% of model fit and does not reach significance ($p = 0.84$). This demonstrates that optimizing an effective adversarial texture is achievable across all weather conditions as long as benign detection remains reliable. Adversarial textures optimized in the various weather conditions in ALLUDE reach the same rate of attack success across all nine weather conditions. Because each attack is optimized and scored under the same weather conditions, the optimizer always operates within its original training distribution. Attack success is shaped more by the target object and viewpoint than by atmospheric conditions, though we do not isolate weather directly in our experimental design.

Attack success generalizes across optimizers and detectors. The attack algorithm and target detector are the two weakest factors in the design, explaining roughly 2% and under 1% of model fit and showing no significant differences across levels. The four optimizers (PGD, Auto PGD, AutoAttack and FGSM) perform at statistically indistinguishable rates ($41 - 57\%$; $p = 0.51$), with overlapping confidence intervals at this sample size. The three detectors also show similar attack success rates across Faster R-CNN (56%), DETR (48%), and YOLOv11 (38%), spanning one-stage, two-stage, and transformer architectures ($p = 0.75$). The numerical ordering loosely tracks model recency: the oldest detector (Faster R-CNN) is most affected and the most modern (YOLOv11) least, but this trend is not statistically significant and we draw no conclusion from it; characterizing whether these modern architectures are systematically more robust would require a dedicated experimental design. Prior physical attack work has largely developed custom attacks tuned to the idiosyncrasies of individual object classes and detectors. ALLUDE instead enables systematic evaluation across a broad range of objects, viewpoints, and conditions within one framework, which reveals where attack success generalizes and where it does not. For safety critical applications in autonomous driving, this distinction matters: robustness cannot be assessed against single objects, static viewpoints, or detector architectures in isolation, and conditions most relevant to deployment (continuous trajectories, degraded weather) are precisely those that prior static, single-object evaluations omit. By making such evaluation accessible across platforms, ALLUDE provides the ability to characterize these critical safety vulnerabilities in simulation before they affect model deployment.

4.2. Evaluating Published Attacks in ALLUDE

Published physical camouflage attacks couple their parameterization, rendering bridge, and loss into method-specific pipelines that do not transfer across frameworks. We therefore evaluate the published adversarial textures themselves across 13 configurations (9 weather conditions with fixed frontal trajectory, 5 trajectories at clear-noon, one shared cell). These attacks are selected based on compatibility with ALLUDE, as characterized by the per-method parameterization, renderer, and simulator in Table 1. Faithfully transferring the UV maps of published textures into our pipeline requires re-unifying the vehicle's UV layout; details on this approach are provided in Sec. S2.2. We evaluate six textures: (1) the **Benign**, **Random** baselines; (2) adversarial textures **CAMOU**, **RAUCA**, and **FCA**; and (3) our **ALLUDE** texture, optimized against DETR with a targeted suppression objective at ϵ=255 on the Audi E-Tron vehicle (Fig. 6). We optimize against DETR as a representative of ALLUDE's native optimization, not as an exhaustive or privileged choice. ALLUDE is detector-agnostic; we select DETR because it is widely used in the detection literature and readily available through a standardized HuggingFace interface, simplifying integration and reproduction. Because a manufactured physical texture cannot be re-optimized after deployment, we report each of the 13 conditions individually to characterize how each texture performs under environmental and viewpoint shift rather than inflating performance through condition-specific optimization.

ALLUDE achieves weather-invariant suppression. Across all nine weather conditions, the ALLUDE-optimized texture always suppresses mAP@50 to 0.00, achieving complete evasion with a single fixed texture. Among the published attacks, RAUCA matches this weather-invariant

Figure 6. Comparison of adversarial textures on the Audi E-Tron under various environmental conditions. Baselines (Benign, Random), published attacks (CAMOU, RAUCA, FCA), and the ALLUDE-optimized texture is rendered across weather conditions and camera trajectories. Textures that suppress detection at clear-noon static views (top) degrade as weather (right) and viewpoint (bottom) shift.

suppression, while FCA and CAMOU transfer far less reliably: FCA fails at clear-noon (1.00) and under light rain (0.98), and CAMOU fails at clear-noon and across all three fog conditions, leaving it the weakest of the published literature attacks. This ordering, RAUCA strongest and CAMOU weakest, is consistent with the results reported by [36], indicating that our rendering evaluation preserves the methods' established relative strengths.

Continuous trajectories remain challenging for all attacks. Under the moving camera trajectories (Table 4), every texture weakens substantially, including RAUCA, whose trajectory-mean rises to 0.46 on DETR from 0.00 under variations in weather. No attack, published or ours, achieves evasion across the moving-camera views. Notably, FCA was designed to provide strong multi-performance [26], yet rendered against DETR it does not retain that robustness, emphasizing the immense evaluation flexibility that ALLUDE offers in making it possible to test against any number of user-defined trajectories to reveal blind-spots. ALLUDE reaches a trajectory-mean result of 0.71 on DETR, with the largest reduction in the static-front view (0.33) while the flyover (0.78), tight-orbit (0.76), and wide-orbit (0.95) views remain largely detectable. This aligns with the ALLUDE configuration space analysis of Sec. 4.1, where camera trajectory is the second-strongest factor in determining attack success, and demonstrates that viewpoint robustness measured from static viewpoints can overstate the effectiveness of existing adversarial attacks.

5. Conclusion

ALLUDE is a first-of-its-kind system that is easy-to-use and offers a rich set of customizable configurations for adversarial attacks across multiple scenes, objects, environmental and lighting conditions, and camera trajectories, allowing cross-platform optimization on both Windows and Linux systems. We comprehensively demonstrate ALLUDE's evaluation breadth through a two-pronged strategy. We also make available CARLA assets such as objects and towns that can be imported and edited in the latest UE framework.

References

[1] Anish Athalye et al. Synthesizing robust adversarial examples, 2018. 3

[2] Shang-Tse Chen, Cory Cornelius, Jason Martin, and Duen Horng Chau. *ShapeShifter: Robust Physical Adversarial Attack on Faster R-CNN Object Detector*, page 52–68. Springer International Publishing, 2019. 2, 3

[3] Francesco Croce and Matthias Hein. Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks, 2020. 4, 11, 12

[4] Alexey Dosovitskiy et al. Carla: An open urban driving simulator, 2017. 2, 3

[5] Michaël Fonder. Mid-air: A multi-modal dataset for extremely low altitude drone flights. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 553–562, 2019. 3

[6] Robert Geirhos, Patricia Rubisch, Claudio Michaelis, Matthias Bethge, Felix A. Wichmann, and Wieland Brendel. Imagenet-trained CNNs are biased towards texture; increasing shape bias improves accuracy and robustness. In *International Conference on Learning Representations*, 2019. 6

Table 4. adv-mAP@50 of the target vehicle, **DETR**, on the Audi E-Tron literature comparison (lower = stronger attack). Weather varied at static-elevated; trajectories at clear-noon. ALLUDE is optimized natively against DETR (suppression objective, ϵ=255) and applied unchanged across all conditions; the published textures are evaluated as transfer. Ground truth is the per-frame benign detection.

	Weather (static-elevated)									**Trajectory** (clear-noon)			
Condition	Clear noon	Dusk	Night	Rain L	Rain M	Rain H	Fog L	Fog M	Fog H	Static front	Flyover	Tight orbit	Wide orbit
Benign	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.89	0.94	1.00
Random	1.00	0.10	0.00	0.53	0.10	0.03	1.00	1.00	0.00	0.96	0.64	0.70	0.95
CAMOU	1.00	0.00	0.00	0.43	0.02	0.00	1.00	1.00	1.00	1.00	0.76	0.70	0.97
RAUCA	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.20	0.55	0.59	0.49
FCA	1.00	0.00	0.00	0.98	0.12	0.00	0.06	0.00	0.00	0.53	0.82	0.76	0.86
ALLUDE	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.33	0.78	0.76	0.95

[7] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples, 2015. 4, 11

[8] Matthew Hull, Haoyang Yang, Pratham Mehta, Mansi Phute, Aeree Cho, Haorang Wang, Matthew Lau, Wenke Lee, Willian Lunardi, Martin Andreoni, et al. Complicitsplat: Downstream models are vulnerable to blackbox attacks by 3d gaussian splat camouflages. *arXiv preprint arXiv:2508.11854*, 2025. 2, 3

[9] Wenzel Jakob, Sébastien Speierer, Nicolas Roussel, and Delio Vicini. Dr. jit: A just-in-time compiler for differentiable rendering. *ACM Transactions on Graphics (TOG)*, 41(4):1–19, 2022. 3

[10] Ruiyang Jia et al. Cca: A novel camouflage coating attack on object detectors in remote sensing images. *IEEE Transactions on Geoscience and Remote Sensing*, 63:1–16, 2025. 3

[11] Hiroharu Kato et al. Neural 3d mesh renderer, 2017. 3

[12] Sahil Khose et al. Skyscenes: A synthetic dataset for aerial scene understanding. 2024. 3, 7

[13] Samuli Laine et al. Modular primitives for high-performance differentiable rendering, 2020. 3

[14] Yang Li et al. Flexible physical camouflage generation based on a differential approach. *arXiv preprint arXiv:2402.13575*, 2024. 3

[15] Jiawei Lian, Jianhong Pan, Lefan Wang, Yi Wang, Shaohui Mei, and Lap-Pui Chau. Padetbench: Towards benchmarking texture- and patch-based physical attacks against object detection. *Knowledge-Based Systems*, 329:114395, 2025. 4, 7

[16] Matthew M. Loper and Michael J. Black. Opendr: An approximate differentiable renderer. In *Computer Vision – ECCV 2014*, 2014. 3

[17] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations*, 2018. 4, 11

[18] Daniel McFadden. Conditional logit analysis of qualitative choice behavior. 1972. 6

[19] Johannes Meier, Luca Scalerandi, Oussema Dhaouadi, Jacques Kaiser, Nikita Araslanov, and Daniel Cremers. Carla drone: Monocular 3d object detection from a different perspective, 2024. 3

[20] Federico Nesti et al. Evaluating the robustness of semantic segmentation for autonomous driving against real-world adversarial patch attacks. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 2280–2289, 2022. 3

[21] Liqiang Linand others. Capturing, reconstructing, and simulating: the urbanscene3d dataset, 2022. 3

[22] Mansi Phute, Matthew Hull, Haoran Wang, Alec Helbling, ShengYun Peng, Willian Lunardi, Martin Andreoni, Wenke Lee, and Duen Horng Chau. Undream: Bridging differentiable rendering and photorealistic simulation for end-to-end adversarial attacks. *arXiv preprint arXiv:2510.16923*, 2025. 3

[23] Giulia Rizzoli et al. Syndrone–multi-modal uav dataset for urban scenarios. *arXiv preprint arXiv:2308.10491*, 2023. 3

[24] Shital Shah et al. Airsim: High-fidelity visual and physical simulation for autonomous vehicles, 2017. 3

[25] Naufal Suryanto, Yongsu Kim, Harashta Tatimma Larasati, Hyoeun Kang, Thi-Thu-Huong Le, Yoonyoung Hong, Hunmin Yang, Se-Yoon Oh, and Howon Kim. Active: Towards highly transferable 3d physical camouflage for universal and robust vehicle evasion. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 4305–4314, 2023. 2, 3

[26] Donghua Wang, Tingsong Jiang, Jialiang Sun, Weien Zhou, Zhiqiang Gong, Xiaoya Zhang, Wen Yao, and Xiaoqian Chen. Fca: Learning a 3d full-coverage vehicle camouflage for multi-view physical adversarial attack. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36 (2):2414–2422, 2022. 3, 8, 15

[27] Jiakai Wang, Xianglong Liu, Jin Hu, Donghua Wang, Siyang Wu, Tingsong Jiang, Yuanfang Guo, Aishan Liu, and Jiantao Zhou. Adversarial examples in the physical world: A survey, 2024. 2, 3

[28] Wenshan Wang et al. Tartanair: A dataset to push the limits of visual slam. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4909–4916, 2020. 3

[29] Hui Wei et al. Physical adversarial attack meets computer vision: A decade survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 46(12):9797–9817, 2024. 2, 3

[30] Tong Wu, Xuefei Ning, Wenshuo Li, Ranran Huang, Huazhong Yang, and Yu Wang. Physical adversarial attack on vehicle detector in the carla simulator. *arXiv preprint arXiv:2007.16118*, 2020. 3

[31] Zuxuan Wu, Ser-Nam Lim, Larry S. Davis, and Tom Goldstein. Making an invisibility cloak: Real world adversarial attacks on object detectors. In *Computer Vision – ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part IV*, page 1–17, Berlin, Heidelberg, 2020. Springer-Verlag. 3

[32] Kaidi Xu, Gaoyuan Zhang, Sijia Liu, Quanfu Fan, Mengshu Sun, Hongge Chen, Pin-Yu Chen, Yanzhi Wang, and Xue Lin. Adversarial t-shirt! evading person detectors in a physical world, 2020. 2, 3, 6

[33] Weilin Xu et al. Imperceptible adversarial examples in the physical world. *arXiv preprint arXiv:2411.16622*, 2024. 3

[34] Tianyuan Zhang et al. Visual adversarial attack on vision-language models for autonomous driving. *arXiv preprint arXiv:2411.18275*, 2024. 3

[35] Yang Zhang, Hassan Foroosh, Philip David, and Boqing Gong. CAMOU: Learning physical vehicle camouflages to adversarially attack detectors in the wild. In *International Conference on Learning Representations*, 2019. 2, 3, 15

[36] Jiawei Zhou et al. Rauca: A novel physical adversarial attack on vehicle detectors via robust and accurate camouflage generation. *arXiv preprint arXiv:2402.15853*, 2024. 2, 3, 8, 15

Supplementary Material

S1. System Implementation Details

S1.1. Weather and Lighting Configurations

ALLUDE enables systematic evaluation of the attacks across nine distinct weather and lighting conditions, defined along three dimensions: (i) time of day (noon, dusk, and night), (ii) weather (rain, fog, and clear skies), and (iii) intensity (high, medium, and low).

Time of Day. The time of day is controlled primarily through the `DirectionalLight` object in Unreal Engine. By varying the light's angle, color, and intensity, we are able to reproduce the lighting characteristics corresponding to different times of day.

Weather: Weather effects are realized through a combination of Unreal Engine assets. (i) **Rain:** Rain is simulated using an asset constructed with Unreal Engine's Niagara particle system. Each such asset defines an emitter that spawns droplets within a volume above the scene, rendered using a combination of sprite and ribbon renderers, where the ribbon produces an elongated streak that replicates the effect of falling rain. Parameters such as the sprite spawn rate and velocity can be adjusted to control the appearance. (ii) **Fog:** Fog is simulated using Unreal Engine's `ExponentialHeightFog` asset, which exposes a range of parameters for controlling the fog's appearance. These include the primary and secondary fog layers, fog opacity, inscattering color, fog density, fog start and end distances, and fog falloff height.

Intensity: Adjusting the intensity of a given weather condition is not accomplished by varying a single parameter, but rather through the coordinated adjustment of several parameters. (i) **Rain:** The intensity of rain is increased by raising the spawn rate and velocity of droplets in the rain asset, increasing the fog density in `ExponentialHeightFog`, darkening the inscattering color slightly, and slightly reducing the light intensity of the `DirectionalLight` object. (ii) **Fog:** The intensity of fog is increased by raising the fog density, decreasing the fog height falloff, and increasing the fog opacity.

S1.2. Optimizer Configurations

ALLUDE integrates four optimizers into its adversarial attack pipeline: FGSM [7], PGD [17], Auto-PGD, and AutoAttack [3]. All four optimizers included with the initial release are ℓ_∞-bounded attacks that operate on the 2D UV texture of the target object. Every iteration, `torch_worker.py` (1) renders a batch of frames through Mitsuba with the current texture as the parameter key, (2) computes the detector's loss on the rendered RGB, (3) backpropagates through the differentiable render to get the per-texel gradients on the UV texture, and (4) applies the update rule of the optimizer, then projects the perturbed texture back onto the ℓ_∞ ϵ-ball around the baseline texture. Details on the shared optimizer hyperparameters are provided in Table S1, while optimizer specific hyperparameters are shown in Table S2.

Table S3 provides additional detail on the 4-stage ensemble used in our AutoAttack implementation. A tracker of the best iteration (`_aa_best_loss`) saves the highest-loss texture across all previous stages. When transitioning to the next stage in the ensemble, the algorithm restores this texture instead of utilizing a $2 \times \epsilon$ to prevent overstepping in the loss landscape. The final texture is the best loss texture across all four stages [3]. The loss that drives each optimizer is detector-specific; Table S4 summarizes the per-detector objective and task mode. DETR and YOLO (task=classification) are untargeted, while FR-CNN (task=object detection) is targeted against the benign bounding box.

Attack-region masks. ALLUDE can confine the optimized region to a binary UV mask, applying gradients only within the masked texels while the rest of the texture stays at its baseline value. Two objects use this (Table S11): the pedestrian, masked to clothing via `rp_mei_clothes_mask`, and the stop sign, masked to the sign face via `stop_sign_face_mask`. These masks are optional, and all other objects are perturbed over their full UV texture.

S1.3. Camera Trajectory Parameters

ALLUDE ships five camera trajectories that govern how the object is viewed during optimization and evaluation. Table S5 lists each trajectory with its fixed frame count and camera-path description. Coordinates are in Unreal units (UU; 1 UU $\approx$ 1 cm), rotations in degrees (UE Euler), and camera FOV defaults to UE's 90°.

S1.4. Runtime Stability Patch

While running ALLUDE, we see issues in `PythonScriptPlugin` due to Python callback receiving a borrowed struct `FMoviePipelineOutputData` and the GIL is released for an extended period (e.g., during `subprocess.run()`). The Garbage Collector (GC) worker thread traces a now-dangling `StructInstance` and reads garbage `UObject` values, leading to a crash in `HandleObjectReference`. We mitigate the error by a two-pronged runtime patch: patching the plugin binary and managing garbage collection in the attack script. The only modification made on the source-level between the plugin Python used in ALLUDE and the default UE5 plugin is that we add a pointer validity check. We check

Table S1. Shared Hyper parameters for all optimizers, identical across all detectors and 100 LHS cells used in evaluation.

Parameter	Value	Additional Information
ϵ (L_∞ budget)	$128/255 \approx 0.502$	pixel space $[0, 1]$
α (step size)	$\epsilon/8 = 16/255 \approx 0.0627$	PGD / AutoPGD init
batch_size	10	frames per gradient update
max_batches	4	40 frame-samples per outer iter
Frame sampling	stride-sampled	indices $[0, n/40, \ldots, n-1]$
clip_min / clip_max	0.0 / 1.0	post-step pixel range
ϵ-ball projection	per-iter	L_∞ around clean texture
AMP	enabled	`torch.cuda.amp.autocast`
Starting texture	random noise	uniform $[0, 1]$
Targeted?	False / True	cls (DETR, YOLO) / det (FRCNN)

Table S2. Per-attack algorithm details. Total calls per cell is the product of `max_iter` with `max_batches`.

Attack	max_iter	Total calls	Update rule
FGSM	1	4	$x \leftarrow x + \epsilon \cdot \text{sign}(\nabla\mathcal{L})$ (single step)
PGD	10	40	$x_{t+1} \leftarrow \Pi_\epsilon\big(x_t + \alpha \cdot \text{sign}(\nabla\mathcal{L})\big)$
AutoPGD	10	40	momentum-EMA on $\nabla\mathcal{L}/\text{mean}(\lvert\nabla\mathcal{L}\rvert)$, step $\alpha \cdot \text{sign}(\text{mom})$
AutoAttack	20	80	4-phase ensemble (Sec. S1.2; APGD-CE $\rightarrow$ APGD-DLR $\rightarrow$ FAB $\rightarrow$ Square)

Table S3. AutoAttack 4-stage ensemble. Best-iterate snapshot restored on transition to next stage; the final returned texture is the best-loss iterate across all four phases [3].

Stage	Iters	Algorithm	Initial step	Step rule
0	1–5	APGD-CE	2ϵ	halved when no improvement
1	6–10	APGD-DLR	restored best	same halving
2	11–15	FAB	α	boundary pursuit
3	16–20	Square	α	query-efficient random search

if `StructInstance` is a dangling pointer holding a small garbage value, and if so the guard skips tracing instead of crashing in `HandleObjectReference`. The advantage of these runtime patches is that they do not require rebuilding Unreal Engine from scratch.

S1.5. Two-Process Architecture

Running a Unreal Engine alongside Mitsuba and Pytorch in a single Python process incurs three independent toolchain conflicts on Linux: (i) Divergent `libstdc++` revisions in the Unreal Engine binaries `libUnrealEditor-InputCore.so` and PyTorch prebuilt wheels are `libtorch_python.so` loaded together causing duplicate C++ exception-handling symbols (ii) Incompatible CUDA runtime (PyTorch/Dr.Jit GPU stack) and threading runtime states (UE's Vulkan based renderer); and (iii) Static OpenMP/TBB linkage in each component, which deadlocks when co-resident. Therefore, we separate the system into two cooperating processes,involving two independent Python interpreters and dependency stacks. UE interpreter drives scene capture and texture re-import in the parent process, while a separate conda interpreter is used for PyTorch and Mitsuba processes, performing gradient computation and texture optimization in a child process. The embedded interpreter never imports the GPU stack. Each iteration spawns a fresh child to which the attack configuration is passed as a serialized JSON message over `stdin`; and the child returns optimizer state which the parent imports. Because a new child is created and torn down on every attack iteration, its CUDA context is established and released cleanly each time, circumventing previously seen issues.

S1.6. Command-Line Interface Arguments

To enable ease in experimentation, we allow the user to modify the parameters given to ALLUDE from the CLI which is used to launch the attack. Detailed implementation instructiuon can be found in the github repository. The parts of the attack that can be changed by the CLI are: number of iterations, epsilon value, alpha value, source directory (which contains parameters such as original texture, scene information, XML configuration, and target object), along with Unreal Engine map information and the scene name in Unreal Engine. All other parameters such as detection

Table S5. The five camera trajectories in ALLUDE. Frame counts are fixed by the `LevelSequence` asset per trajectory archetype, while the camera path is encoded as a per-frame transform list in `camera.json`.

Trajectory	Frames	Camera path
static_frontal	48	Stationary; frontal viewpoint at object-centroid height.
static_elevated	48	Stationary; raised altitude with downward pitch.
tight_attack_orbit	96	Elliptical orbit, ∼5–10 m radius around object, full 360°.
wide_orbit	120	Elliptical orbit, ∼20–30 m radius, full 360°.
detail_flythrough	120	Linear flyover, ∼6–30 m traversal, varying pitch and distance.

model, detector task, number of frames in a sequence, attack type, Mitsuba to UE scale, maximum batches to allow optimization over (in case you have a batch limit), detection threshold, etc are exposed in `config/config.yaml`

S1.7. Modernized CARLA Assets

ALLUDE ships a UE 5.7–ported CARLA asset library, decoupled from the CARLA backend. Total: 18 GB on disk, 14,142 Unreal asset files (`13,912 *.uasset + 230 *.umap`). Additional information on each of these scenes is provided in Table S6, while additional information on the CARLA assets is given in Table S7.

Town08 and Town09 are absent from CARLA's open-source release.

HuggingFace release. `huggingface.co/datasets/allude-occluded/real-carla-assets-anon`. The release excludes the auto-regenerable `Saved/`, `Intermediate/`, `DerivedDataCache/`, directories and only includes relevant asset content (`Content/`,`Config/`, `CarlaAssets.uproject`)

Migration to Unreal 5.7. The most recent version of CARLA's driving content, including town levels, static-mesh, material, and the texture assets, was authored against CARLA's bundled Unreal Engine Fork (Unreal Version 5.5). We re-imported this content into a dedicated 5.7 project using Unreal's forward-compatible content migration, which transfers each asset together with its dependencies while preserving the original paths to avoid breaking any references. All broken CARLA blueprints that failed the content migration were removed from the resulting scene. After the migration, we cleared the `DerivedDataCache` so that Unreal re-derives shaders and platform data instead of serving stale artifacts from the older engine.

Table S6. Modernized CARLA towns shipped with ALLUDE. Town01 and Town03 are included but unused in the LHS ablation (covered by Town02 and Town10HD respectively).

Town	File	Setting
Town01	Town01_Opt.umap	Small village
Town02	Town02_Opt.umap	Small town
Town03	Town03_Opt.umap	Urban
Town04	Town04_Opt.umap	Rural
Town05	Town05_Opt.umap	Suburban
Town06	Town06_Opt.umap	Urban grid
Town07	Town07_Opt.umap	Small town
Town10HD	Town10HD_Opt.umap	Dense urban

S2. Evaluation

S2.1. Literature Attacks

Published physical-camouflage attacks couple their texture parameterization, differentiable rendering bridge, and detector loss into method-specific pipelines that do not transfer across frameworks. Re-implementing each end-to-end would change the attack, not reproduce it.

We therefore take the published adversarial textures and deploy them, unmodified, on a common target object (the Audi e-tron) rendered through our framework across a fixed configuration set. This isolates the rendered, deployed appearance (What the object detector sees) from the specific attack pipeline. Table S8 describes the six evaluated tex-

Table S4. Per-detector loss formulations used by the attack optimizer. DETR and YOLO operate under `task=cls` (untargeted); FRCNN under `task=det` (targeted suppression).

Detector	Task	Loss formulation
DETR (`facebook/detr-resnet-50`)	cls	Argmax-query cross-entropy: select query $j^* = \arg\max_q P(y\|q_j)$, then $\mathcal{L} = \mathrm{CE}(\mathrm{logits}[:, j^*, :], y)$.
YOLOv11n (ultralytics)	cls	Argmax-anchor sigmoid BCE: best-anchor selection, one-hot binary cross-entropy against target; sigmoid-probs cast to fp32 pre-BCE for AMP compatibility.
Faster R-CNN (`fasterrcnn_resnet50`)	det	Sum of all task losses in `model.train()` mode: $\mathcal{L} = \mathcal{L}_{\mathrm{cls}} + \mathcal{L}_{\mathrm{box}} + \mathcal{L}_{\mathrm{obj}} + \mathcal{L}_{\mathrm{rpn\_box}}$, with per-frame {label, boxes} from `target_det.json`.

Table S7. Asset categories in the modernized CARLA project with 14,142 Unreal asset files including 13,912 `.uasset` and 230 `.umap`, ∼18 GB on disk.

Category	Path	Count / Size
Maps	Content/Carla/Maps/	8 `.umap`
Object meshes	Content/CV_Pipeline/Meshes/	10 per-class `SM_*.uasset`
Materials	Content/CV_Pipeline/Materials/	10 base MIs + ∼100 scene MIs
Level sequences	Content/Sequences/CV_Experiments/	∼100 LevelSequences
Pedestrian meshes	Content/Scanned3DPeoplePack/	∼620 MB
Weather FX	Content/Pipeline/FX/	3 rain + 3 fog assets
Weather blueprints	Content/Carla/Blueprints/	2 (`BP_CarlaWeather`, `BP_GeneralSceneSettings`)
FAB store props	Content/Fab/	misc environmental props

tures: two controls (Benign, Random), three published camouflage attacks (CAMOU, FCA, RAUCA), and our ALLUDE texture.

S2.2. UV Unification of Literature Attacks

The published camouflage textures are baked for the UV unwrap of its original vehicle mesh. Applying it to a different mesh whose UV layout differs will scramble the pattern, causing the rendered vehicle to be unfaithful to the original implementation. To avoid these inconsistencies, we utilize the `pytorch3d_Etron` unwrap for the published textures and the `SM_EtronParked` texture for ALLUDE's adversarial texture Table S9.

The two meshes share the topology, but differ in their unwrap approach. Applying a published asymmetric texture as symmetric would mirror the adversarial pattern, creating an unfaithful deployment of the published work. We re-unify the CARLA e-tron body UV to the `pytorch3d_Etron` layout so that the published textures are able to be applied exactly as intended to the vehicle. While the UV layouts of the attacks differ between published attacks and the ALLUDE texture, both geometries are identical, allowing direct comparison between attacks.

Method. We performed the re-UV in Unreal Engine 5.7 through `GeometryScript`, only editing the `UV0` and leaving the material slots and scale untouched. The process is as follows: (1) create a editable dynamic mesh through copying the original CARLA mesh (`copy_mesh_from_static_mesh(SM_EtronParked)`). (2) Transfer the published work unwrap (`CARLA_Etron_pytorch3dUV.obj`) per triangle through `set_mesh_triangle_u_vs` (by value), with a V-flip (`v_ue = 1 - v_obj`) to match the UE texture convention. (3) Utilize the `copy_mesh_to_static_mesh(..., replace_materials=False)` to write the UV0 back.

Verification. After the swap, the body material slot (slot 2) and all other 6 slots are intact, the bounding scale ($485.6 \times 203.3 \times 164.9$ cm) is identical, and the body UV is now a single asymmetric tile. A red-channel / single-texture rendering test produced full asymmetric coverage matching the consumer's reference render, confirming that the unwrap is correct end-to-end.

S2.3. LHS Experiment

We model per-cell attack success as a binary outcome (advmAP@50 less than 0.5) over the $N = 86$ scored cells, using using logistic regression on five categorical factors: optimizer (4 levels), scene–object pair (10 levels), trajectory (5 levels), weather (9 levels), and detector (3 levels). With reference-level (dummy) encoding this yields 26 predictor columns. Because the predictor count is large relative to N, we apply L2 regularization. For each factor, "Variance explained" is the drop in McFadden psuedo-R^2 when that factor's columns are removed and the model is refit. Per factor significance is assess by a likelihood-ratio test between the full model and the model with that factor removed, referenced to a chi-squared distribution with degrees of freedom equal to $d - 1$

S2.3.1. Scene Details

ALLUDE has 10 object, town, and setting context pairs sampled by the LHS. Each scene is crossed with all 5 trajectories $\times$ 9 weather conditions $=$ 45 cells; the balanced LHS samples 10 per object for a total of 100 cells. Refer to Table S10 for further details. Fig. S1 shows the rendered output of all 100 sampled cells.

S2.3.2. Object Details

Table Table S11 gives details about the per-object mesh, Material Instance (MI), perturbed material parameter, and UV mask (if any).

S3. Dataset Licensing, Impact Statement

S3.1. Licensing & Asset Distribution

ALLUDE is open-source under the BSD-3 license. The HuggingFace release (allude-occluded/real-carla-assets-anon) ships only content that is (a) CARLA-derived and

Table S8. Descriptions for all 6 evaluated textures: Benign, random, CAMOU, FCA, RAUCA, and ALLUDE

Attack	Description
Benign	Stock vehicle paint (control).
Random	A random-noise texture (control; bounds "any non-trivial texture").
CAMOU	[35] — Learned camouflage; originally optimized against **Mask R-CNN** in a black-box setting through a learned clone (surrogate) network. Earliest of the three; weakest even against its own detector.
FCA	[26] (Full-coverage Camouflage Attack) — full-vehicle coverage; originally optimized **white-box against YOLOv3**.
RAUCA	[36] (Robust And Universal Camouflage Attack) — neural-renderer-based camouflage; originally optimized **white-box against YOLOv3**; the strongest published baseline.
ALLUDE	(ours) — A single fixed texture optimized **natively against DETR** with a targeted (suitcase-class) misclassifications objective at $\epsilon = 256$, 2048^2 resolution, optimized at the static-elevated / clear-noon viewpoint and applied **unchanged** to every condition.

Table S9. UV layout of the CARLA `SM_EtronParked` mesh versus the `pytorch3d_Etron` published-texture convention. The two meshes are topologically identical (same face count); they differ only in UV unwrap, which is why the published textures must be re-unified onto the CARLA layout (Sec. S2.2).

	CARLA `SM_EtronParked`	`pytorch3d_Etron`
U range	$u \in [0, 2]$ (wrapped / tiled)	single $u \in [0, 1]$
Symmetry	left/right mirrored (symmetric)	asymmetric (front $\neq$ rear, L $\neq$ R)
Vertices	20,747	13,487
Topology	23,145 faces (identical)	23,145 faces (identical)

CC-BY licensed, or (b) our own original modifications. Third party-assets governed by non-redistributable licenses, such as Fab-store assets and RenderPeople scanned meshes, have been removed from the release. Users can obtain them separately through their original distributors.

S3.2. Impact Statement

ALLUDE enables adversarial attack optimization in a controlled simulation environment. Its primary intended use is defensive, where researchers working on safety-critical-applications can quantify attack surfaces on vision models before deployment.

Table S10. The 10 (object, town, setting context) pairs sampled by the LHS. Each scene is crossed with all 5 trajectories $\times$ 9 weather conditions $=$ 45 cells; the balanced LHS samples 10 per object (100 cells total).

Object	Town	Setting context
bench	4	rural park bench
bicycle	10	urban curbside bicycle
bus	10	parked bus, wide street
car	10	parked sedan, downtown stall
fire_hydrant	2	small-town sidewalk hydrant
motorcycle	10	downtown sidewalk motorbike
parking_meter	10	city corner parking meter
pedestrian	5	suburban sidewalk person
stop_sign	7	rural intersection sign
traffic_light	6	urban-grid intersection

Figure S1. Rendered output of all 100 LHS-sampled cells (10 objects × 10 sampled configurations each), spanning the crossed trajectory and weather conditions.

Table S11. Per-object mesh, Material Instance (MI), and perturbed material parameter.

Object	COCO	Mesh (`SM_*`)	MI	Param
bench	15	`Bench01`	`bench`	`BaseColor`
bicycle	2	`RoadBike`	`bicycle`	`AdversarialAlbedo`
bus	6	`MitsubishiFusoRosa_Parked`	`bus`	`AdversarialAlbedo`
car	3	`DodgeChargerCop_Parked`	`car`	`AdversarialAlbedo`
fire_hydrant	11	`FireHdrant`	`fire_hydrant`	`AdversarialAlbedo`
motorcycle	4	`Harley`	`motorcycle`	`Diffuse`
parking_meter	14	`Parkingmeter`	`parking_meter`	`Diffuse`
pedestrian	1	`rp_mei_posed_001`	`pedestrian`	`Albedo`
stop_sign	13	`Stop01`	`stop_sign`	`SpeedSign_d`
traffic_light	10	`SM_TrafficeLight_Merged`	`traffic_light`	`AdversarialAlbedo`