Universiteit Leiden

A Synthetic, Observability-Annotated Dataset for Viewpoint-Dependent Vehicle Damage Detection

Stefan-Ioan Diaconu (s3836010)

Supervisors:
Dr. D.M. Pelt & J.T. te Beest MSc

Bachelor Thesis– Data Science and Artificial Intelligence

Leiden Institute of Advanced Computer Science (LIACS)
liacs.leidenuniv.nl

July 2, 2026

Abstract

Assessing whether a vehicle is intact or damaged is done manually at enormous scale, and it is inherently viewpoint-dependent: a dent obvious from one angle is invisible from another. Training a convolutional classifier to automate the task needs many labelled images, which are scarce because real damage photographs are constrained by privacy and cannot be staged at scale. We address this by building the data synthetically. This thesis presents a procedurally generated, domain-randomized dataset of vehicle damage, produced entirely from open-source tools (the Blender Python API and publicly available 3D car models) comprising roughly 9,000 labelled images of five vehicles in intact and damaged states. Its novel element is a per-view observability annotation that records, for every image, whether the damage is easily seen, partially seen, or hidden from the camera. We treat the dataset as the object of study and use a ResNet-18 classifier, and a confidence-gated active-vision loop built on it, as instruments to test what it supports. The dataset trains a classifier that generalizes across unseen lighting (≈ 0.78) and across vehicles as far as its body-category coverage reaches (0.74–0.82 for a held-out car with a category counterpart, 0.42 for the lone van without one). The observability annotation is faithful: detection accuracy follows the order it predicts. Confidence-gated acquisition adds no mean accuracy. Its limit is the classifier's calibration, which collapses on the out-of-category vehicle and there doubles as an out-of-distribution signal.

Contents

1 Introduction

Assessing the physical condition of a vehicle is a task carried out at enormous scale. Motor insurance alone is a global market of roughly 2.3 trillion USD in annual premiums. Passenger cars account for nearly three-quarters [23]. Behind every policy lies the same basic judgement: is this vehicle intact or damaged? This judgement is made over and over: by an insurer settling a claim, by a car-sharing or rental fleet inspecting a vehicle at each handover [29], by a leasing company processing a return, by a used-car platform grading a trade-in, and by a manufacturer checking a panel at the end of the production line. The sums involved are enormous. Europe's insurers alone paid out an estimated 2.7 billion EUR in claims every day in 2019 [13], and damage assessment sits on the critical path of each such transaction.

Today that assessment is largely manual. A person inspects the vehicle and decides whether it carries damage, from a minor dent to structural failure. This has two weaknesses: it is slow, and its outcome depends on one inspector's judgement, which makes it inconsistent. It is also costly when the analysis is wrong. In insurance, the gap between what a claim should cost and what is actually paid (*claims leakage*, caused by overpayments and by undetected or exaggerated damage) is a recognised drain on the industry, and manual validation cannot keep pace with the speed that modern, express claims handling demands [10]. Automating damage detection would make the assessment faster and more consistent, cut these losses, and free skilled inspectors for the cases that require them [30].

1.1 The Situation

Automating this judgement is a computer-vision problem, and the standard tool for it is deep learning. A convolutional neural network learns to recognise damage from labelled example images, which suits damage that varies in shape and location. The vehicle-damage literature is built almost entirely on this approach (Section 2.8) [30, 10]. Its cost is data: training a network to a usable accuracy takes a large set of labelled images [17].

Labelled vehicle-damage images at that scale are scarce. Real photographs of damaged cars sit behind insurance and privacy constraints, and damage cannot ethically be staged on real vehicles at the volume a network needs [30]. The data that the task demands does not exist in the quantity required to train on directly.

The usual way around this scarcity is to generate the data synthetically: we render three-dimensional car models procedurally, producing labelled images in any quantity. Rendering brings its own risk (the reality gap) because a network can learn artefacts of the rendering software instead of real damage. The standard response is domain randomization [28].

Generating the data ourselves also gives a control that real photographs cannot: we set the camera viewpoint for every image. Viewpoint is decisive, because a dent on the rear panel is obvious from behind and absent from the front. We record, for each rendered image, whether its damage is easily seen, partially seen, or hidden from the camera. This per-view observability label is what makes the dataset novel.

The question this thesis investigates follows directly: can a synthetic dataset of this kind train a convolutional classifier to tell damaged cars from intact ones? On top of the classifier we add an uncertainty-driven active-vision loop, which decides when its current view is too ambiguous and requests another viewpoint.

1.2 Research Question

The question raised at the end of the previous section can be stated precisely. Throughout this thesis, the dataset is the object under study, and the classifier is the instrument we use to test it. By training a convolutional network, we measure whether the dataset is good enough to learn from. The main research question is therefore:

> **Can a procedurally generated, domain-randomized synthetic dataset of multi-view vehicle imagery, annotated with per-view damage observability, train a convolutional classifier that distinguishes intact from damaged vehicles and generalizes beyond the exact rendering conditions it was trained on?**

This question divides into four sub-questions. The first asks whether the dataset can be built at all. The other three each isolate one property of the dataset once it exists.

- **SQ1 (dataset generation). Can existing open-source tools (the Blender Python API and publicly available 3D car models) be used to generate, at scale, a labelled dataset of intact and damaged vehicles suitable for training a damage classifier?** Answering it establishes that the artifact this thesis studies can be produced from freely available components, and that its labels come for free from the generation process
- **SQ2 (generalization). To what extent does the dataset train a classifier that generalizes beyond its rendering conditions (to an unseen lighting environment and an unseen vehicle) and does that generalization depend on what the dataset contains?** Answering it shows whether the dataset's domain randomization suppressed render-specific shortcuts, and whether its coverage of vehicle shapes is wide enough to carry a held-out one.
- **SQ3 (annotation utility). Does the per-view observability annotation predict detectability, i.e., does the classifier detect damage labelled *easily_seen* more reliably than damage labelled *hidden*?** Higher accuracy and confidence on the visible cases would show the annotation marks what it claims to.
- **SQ4 (downstream use). Does the dataset's observability structure support a downstream, confidence-gated acquisition policy, and what bounds does that support?** Here the active-vision loop serves as evidence of what the dataset and its observability annotation make possible.

1.3 Contributions

This thesis makes three contributions.

First, we present a synthetic dataset of vehicle damage, together with the procedural Blender pipeline that generates it from open-source tools and publicly available 3D car models. The dataset contains roughly 9,000 labelled images of intact and damaged cars across five vehicle models. These are rendered from a range of viewpoints under domain-randomized lighting and backgrounds (Section 5.3). Every image is annotated by the pipeline with its camera viewpoint, vehicle model, and damaged region. Its novel element is a per-view observability label: for each damaged image, whether the damage is easily seen, partially seen, or hidden from the camera.

Second, we investigate empirically whether this dataset can train a convolutional neural network to classify vehicle damage. We treat the difficulties encountered during training, and the dataset revisions they forced, as findings that motivate the final pipeline decisions (Section 5.3).

Third, we use the dataset to build an uncertainty-driven active-vision loop, in which the classifier's own confidence decides when to request an additional viewpoint. This demonstrates a downstream use that the dataset and its observability annotation make possible.

1.4 Thesis Overview

The remainder of this thesis is organized as follows. Section 2 reviews the background and related work (Convolutional Neural Networks, the ResNet architecture, transfer learning, synthetic data generation, the sim-to-real reality gap, and domain randomization) and positions the work against the literature on active vision and automated vehicle-damage detection. In Section 3 we present the methodology: the design of the system and the synthetic dataset, how we generate the dataset, how we train the classifier, and how the active-vision loop is built on top of it. Section 4 reports the experiments and their results. Section 5 interprets them, documents the implementation challenges we met in making the pipeline work across heterogeneous vehicle models and the dataset revisions they forced, and examines the threats to their validity. Section 6 concludes and points to directions for future work.

This work was carried out as a BSc thesis at the Leiden Institute of Advanced Computer Science (LIACS), Leiden University, under the supervision of Dr. D.M. Pelt.

2 Background and Related Work

The classifier learns the intact-versus-damaged decision by supervised learning: it is shown a set of labelled images and adjusts its parameters to reduce the errors it makes on them. How well it learns scales with the number of labelled examples, and a deep network needs them in large numbers [17], far more than the largest public vehicle-damage datasets provide (Section 2.8) [30, 9]. Generating the images synthetically is what removes this limit.

A photograph of a car panel at the input resolution used in this work (224 x 224) contains roughly 150,000 pixel values. A scratch, dent or a deformation could appear anywhere across the spanning surface. A fully connected network assigns a separate weight to every input. To recognise a dent, it would therefore have to learn what that dent looks like independently at every possible location. This inflates the parameter count and gives no guarantee that a pattern learned in one region transfers to another. Convolutional neural networks (CNNs) [18] avoid this. They exploit a property of images that fully connected networks ignore: a local pattern is the same wherever it occurs.

2.1 Convolutional Neural Networks

The central operation is the convolution. A small grid of learnable weights, called a filter, slides across the image. At each position it produces a single response measuring how strongly the local patch matches the pattern the filter encodes. Formally, for an input feature map I and a filter K

of size $k \times k$, the response at location (i, j) is

$$S(i,j) = \sum_{m=1}^{k} \sum_{n=1}^{k} I(i+m,\, j+n)\, K(m,n), \tag{1}$$

the weighted sum of the local patch under the filter. Because the same filter K is applied everywhere, a feature it detects in one part of the image is detected identically in any other, so the network learns that pattern once rather than once per location. Damage is a local disruption of an otherwise smooth, regularly shaded panel: an edge, a crease, or a shading discontinuity where the surface folds. These are exactly the local patterns a convolutional filter responds to. Because such disruptions can fall anywhere on the car, position-independent detection matters most here.

Training begins with random filters and adjusts them by gradient descent. It repeatedly nudges the weights in the direction that reduces the classification error, until the filters respond to patterns useful for the task. Between convolutional layers, a pooling operation downsamples each feature map by keeping only the strongest response in each small region. Pooling reduces the spatial size of the representation and makes the network tolerant to small shifts in where a feature appears. It also lets later layers combine simple features into more complex ones: early layers respond to edges, such as the rim of a dent, and deeper layers combine those edges into a representation of a deformed region. Stacking many such filters builds a hierarchy of increasingly abstract features. Deep stacks are difficult to train, a problem addressed by the residual architecture described in Section 2.2.

2.2 ResNet

Making a network deeper should never hurt it. A deeper network can always reproduce a shallower one by copying it in the early layers and passing the input through the remaining layers unchanged, so it can match the shallower network's training error exactly. In practice the opposite happens: beyond a certain depth, adding layers *increases* the training error. This is the degradation problem. This is different from the training signal weakening as it propagates back through many layers (degradation remains even when that signal is kept strong). The cause is instead that a plain stack of layers cannot easily learn the identity mapping (reproducing its input unchanged) so it never reaches the harmless solution that its own capacity guarantees.

He et al. [12] remove this obstacle with residual networks. They change what each block of layers computes: instead of producing its output from scratch, a block computes only the *change* to apply to its input, and a shortcut connection adds the original input back to that change. Where a plain block would have to learn a target mapping $H(x)$ directly, a residual block instead computes

$$y = F(x) + x, \tag{2}$$

where x is the block's input, F the transformation its layers learn, and the added x the shortcut connection. The block therefore only has to learn the residual $F(x) = H(x) - x$. To leave its input unchanged, a block now only has to drive $F(x)$ to zero, instead of learning the identity mapping from scratch. The degradation problem disappears, because adding a block can no longer make the network worse than leaving it out. This is what makes networks of hundreds of layers trainable, and why residual architectures became a standard choice for image classification.

We use ResNet-18, the shallowest of the standard residual variants. A deeper variant such as ResNet-50 has more capacity, but also far more parameters to fit, and our dataset is small for that capacity: roughly 9,000 images of a single object category. On a dataset this size, a larger network is more likely to memorise the rendering artefacts that domain randomization is designed to suppress (Section 2.6).

2.3 Transfer Learning

Training a deep network from random initialization needs a large amount of data. Our dataset is small by that standard as noted in Section 2.2, so we use transfer learning: we initialize the network with weights already trained on a large, general image collection, then adapt it to damage detection.

That collection is ImageNet [8] . It is a dataset of over a million natural photographs spanning a thousand categories. In its early layers, a network trained on ImageNet learns visual features that are not specific to any one category: edges, gradients, corners, and textures. This is the basic vocabulary from which natural images are built. These low-level features are useful for almost any vision task, including ours, since the edges and shading discontinuities that signal a dent are built from exactly that vocabulary. Reusing them means the network starts with a working visual front end and only has to adapt its more task-specific layers.

Adapting the network means replacing its task-specific head and fine-tuning on our data. We give the exact procedure in Section 3.3. Pretraining brings a second benefit specific to our setting: the reused features come from real photographs. The network therefore brings knowledge of real-image statistics into a task where every training example is rendered, giving it a head start on the gap between rendered and real images that Section 2.5 discusses.

2.4 Synthetic Data Generation

The scarcity of labelled vehicle-damage data, noted in Section 1, is what motivates the approach here. Instead of collecting and hand-labelling photographs of real damaged cars, we generate the training images synthetically. We do this by rendering three-dimensional car models in virtual scenes [31].

This sidesteps the data problem on three fronts. First, the supply of images is unlimited: we can render any number of pictures, from any angle, under any lighting condition, for any car. Second, the labels are exact and free, because the generator fixes the state of each car (intact or damaged, and where) before the image exists. Third, because we control every condition of a render, properties that would be incidental in a photograph (such as the viewpoint or the lighting) become variables we set deliberately and record alongside the image.

This control is the foundation the rest of the pipeline builds on. It comes with one caveat: a rendered image may not resemble a real photograph closely enough for what the network learns to carry over. That problem is the subject of Section 2.5.

2.5 The Reality Gap

The control that makes synthetic data useful is also its weakness. A renderer only approximates the real world (its lighting, materials, and sensor characteristics differ from those of a real photograph).

This means that a network trained only on renders may fit features specific to the rendering, and fail on real images. This mismatch between the training and deployment distributions is the reality gap [1]: the risk that a classifier trained on rendered cars will not carry over to real ones. The gap can be narrowed by making the renders more realistic or by training so that the differences stop mattering. We take the second route and describe it in Section 2.6.

2.6 Domain Randomization

Domain randomization is the second approach. Rather than making each rendered image look more like a photograph, Tobin et al. [28] vary the conditions under which images are rendered. This process is done so widely that the real world becomes just one more variation the network has already seen. Properties such as lighting, background, and camera viewpoint are randomized from image to image, so that no single rendering condition is reliably tied to either class.

This removes the artefacts the network would otherwise exploit (Section 2.5). If the background changes on every image, it cannot indicate whether a car is damaged. If the shadow pattern caused by the lighting is random, lighting cannot carry information about the class. The only property left consistently tied to the label is the damage itself, and that property is still present in a real photograph.

In our pipeline we randomize the lighting and background (both drawn from a set of environment maps) and the camera viewpoint. Section 3.2 gives the exact properties and the ranges over which they vary.

Having established the technical background, we now position this work against two bodies of literature: active vision, and automated vehicle-damage detection.

2.7 Active Vision

Active vision frames perception as a process in which the observer acts. Rather than computing a one-shot mapping from a fixed input to an output, a system with control over its camera (its viewpoint, focus, or zoom) uses what it has already inferred to choose the next observation it acquires. The idea is long-standing. Bajcsy [2] introduced active perception in the late 1980s, arguing that perception is exploratory and not a passive process: an observer that moves and refocuses can resolve ambiguities a static one cannot. It predates deep learning and has been studied across robotics, embodied vision, and object recognition. More recently it has been paired with deep classifiers: Jayaraman and Grauman [14] learn a policy that selects informative views for recognition, and Shang and Ryoo [26] study active vision under limited observability with reinforcement learning.

Our design sits in a more restricted corner of this literature than most learned policies, and two contrasts mark the boundary. The first is the action space. Many learned viewpoint policies cast selection as a continuous control problem and use reinforcement learning to optimise the path a camera takes through space [26]. Ours instead chooses among the five discrete named viewpoints already used to generate the dataset. The second is the structure of the decision. Reinforcement-learning approaches typically frame acquisition as a long sequential problem with a reward shaped over the whole trajectory. Ours takes at most a few additional views and aggregates them in one shot. From this literature we borrow a single premise: that a model's confidence in its

current prediction can decide whether to acquire another view. We describe the resulting policy in Section 3.4.

This premise carries an implicit assumption. Confidence-gated acquisition is only well-founded if the model's confidence is calibrated, that is, if its reported probability matches the true likelihood of the prediction being correct. Guo et al. [11] show that modern neural networks are frequently miscalibrated, and tend to be overconfident. Whether our classifier's confidence is informative enough to guide acquisition is therefore an empirical question, and one we return to in Section 4.4.

2.8 Automated Vehicle-damage Detection

Automated vehicle-damage detection is studied almost entirely as supervised learning on real photographs, and is driven by the automation of insurance claims. The task has been approached at three levels of granularity.

It was first framed as image classification. Patil et al. [24] fine-tune ImageNet-pretrained CNNs to label a panel by damage type, and Dwivedi et al. [10] extend this with a larger model set and a YOLO detector that also localises the damaged region. A 96% classification accuracy across seven damage types is reported.

van Ruitenbeek and Bhulai [29] push the localisation further, training detectors to place bounding boxes over twelve damage categories on more than ten thousand images for a car-sharing inspection setting.

Wang et al. [30] extend the task to pixel-level instance segmentation, contouring the exact extent of each dent, scratch, or crack. Across this progression from classification to detection to segmentation, the ingredients are the same: real photographs, transfer learning from ImageNet, and an insurance or fleet-inspection motivation.

What has held the field back is data. Because real damage photographs are constrained by privacy and insurance confidentiality (Section 1), most of this work is conducted on small, private datasets, which prevents methods from being compared on common ground. Wang et al. [30] address this with CarDD, the first large-scale public benchmark for the task: four thousand images with over nine thousand annotated damage instances across six damage types. Hoang et al. [9] follow with VehiDE, a similar public benchmark of roughly fourteen thousand images and eight categories, and Pérez-Zarate et al. [25] release TartesiaDS, labelled under the supervision of professional insurance appraisers. These releases are recent and still few. The scarcity of public data is the condition that motivates this work, and it is why we generate our data synthetically (Section 2.4).

First, every dataset above assumes the damage is already in frame: each spreads its images across many vehicles but shows each damage instance once, from a pose that reveals it. We do the opposite. We re-image a small set of damage instances across viewpoint and lighting, so the same damage is captured both where it is visible and where it is not. This is what lets the dataset model the viewpoint dependence of damage, that whether a dent appears at all depends on the pose it is seen from. Prior work does not: where viewpoint enters at all, it is held fixed or treated as noise. Li et al. [20] normalise away the appearance change that viewpoint causes when matching damage across claim photographs. Lee et al. [19] fix every image to a single three-quarter orientation. CarDD records the camera angle only as a dataset statistic. Second, and as a direct consequence, no prior dataset annotates whether damage is observable from the pose an image was taken from. This per-view observability annotation is the differentiator of our dataset (Section 3.1). Third, synthetic

imagery with domain randomization, established in robotics and autonomous driving (Section 2.6) and applied to defect and part inspection in manufacturing [32, 22], is rarely applied to panel-level vehicle damage. We adopt it because it is what makes the camera viewpoint a controllable variable in the first place.

3 Methodology

We divide the work into three phases. In the first phase we generate the synthetic dataset of intact and damaged cars by rendering three-dimensional models under randomized conditions. In the second phase we train the convolutional classifier on this dataset and evaluate how well it generalizes across cars, viewpoints and lighting. In the third phase we build an active vision loop on top of the trained classifier. It uses the classifier's confidence to decide when an additional viewpoint is required. This chapter describes all three phases: Section 3.2 presents the generation pipeline, Section 3.3 covers the classifier's training and evaluation, and Section 3.4 sets out the active vision loop.

3.1 Overview: System and Dataset Design

We build the dataset as a controlled experiment. We hold the damage roughly fixed and vary only where the camera sits relative to it, so that observability (whether the damage falls inside the camera's view) is the variable we study. Real photographs cannot give us this control. It would require identical damage on identical cars, photographed from prescribed angles, and that combination is exactly what data scarcity makes unavailable. Rendering the cars ourselves removes the obstacle: we fix the damage and vary the viewpoint during image construction.

Every rendered view carries its own observability label. According to where the camera sits relative to the damaged region, we assign each view one of three levels: *easily_seen*, *partially_seen*, or *hidden*. This annotation is what distinguishes our dataset. Because the label is recorded for every image, we measure directly how the classifier's accuracy depends on whether the damage is in view, and we do not have to infer that relationship after training. The annotation is also exact: each image arrives with its class, viewpoint, and observability known by construction, so the dataset needs no manual labelling. We compute the observability label geometrically, and Section 3.2 gives the procedure.

We therefore ask whether the dataset can train a usable damage detector, and use the active vision loop as further evidence that observability affects the classifier at inference and not only in how the data is labelled. The scope is deliberately narrow. We work with synthetic images only, a binary intact-versus-damaged decision, five car models, six HDRI lighting environments, and damage produced geometrically by surface displacement.

3.2 Synthetic Dataset Generation

We generate the dataset in Blender, an open-source 3D creation suite,[1] driven entirely through its Python API. The pipeline is scripted across four modular Python files (Figure 1), which makes the dataset reproducible. A single run imports a car model from a fixed pool of vehicles, applies

[1] https://www.blender.org/

procedural damage to a chosen region of the body, places a camera at a sampled viewpoint, sets the lighting and background, and renders an image. This image contains a metadata record describing how that image was produced. Repeating the run across car models, damage locations, viewpoints, and lighting conditions produces the full dataset.

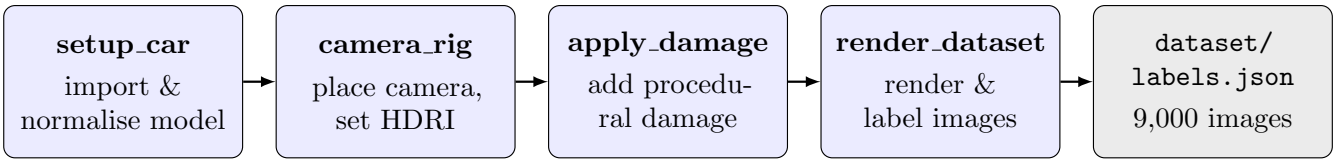

Figure 1: The four-stage synthetic data generation pipeline. Each stage is an independent Blender Python script. The final stage iterates over car models, lighting environments, damage zones, and viewpoints to write the labelled dataset (Equation (7)).

Model Normalisation We normalise each imported model to a common size before any damage or rendering. The models come from different authors ([3, 15, 6, 7, 27]) and arrive at inconsistent scales. For example, one model was exported in centimetres and another in metres. Because the camera sits at a fixed distance from the car, these raw scales would make some cars fill the frame and others appear as specks. We therefore scale each model uniformly by a factor

$$s = \frac{T}{\max(\Delta x,\, \Delta y,\, \Delta z)}, \qquad T = 4.5, \tag{3}$$

where $\max(\Delta x, \Delta y, \Delta z)$ is the longest side of the model's world-space bounding box and $T = 4.5$ Blender Units the fixed target, and we centre it at the origin. A car from any source is then framed consistently by a camera at the same distance, so one viewpoint definition applies to every model.

Procedural Damage We apply the damage procedurally, which lets us expand the dataset by changing parameters. The mechanism stacks two Blender modifiers on each damaged panel. A subdivision-surface modifier first inserts additional geometry into the panel without changing its shape. We use simple subdivision instead of the default Catmull-Clark [5], so the panel's edges and intended form are preserved. A displacement modifier then acts on the subdivided mesh. It pushes each vertex outward along its surface normal by an offset driven by a CLOUDS noise texture (noise scale 2.0, depth 2). The overall depth of the deformation is set by a strength parameter described below. The result is a localised, irregular bulge or crumple that reads as physical damage when rendered. We subdivide before displacing so that there is enough geometry for the deformation to register on any mesh. Section 5.3 explains the low-polygon failure mode that this avoids.

To keep the damage consistent across cars imported at different native scales, we set the displacement strength as a fraction $\alpha = 0.18$ of the car's overall size D, taken as the largest dimension of its world-space bounding box, so a dent on a small car is proportionally as deep as one on a large car. Because Blender's displace modifier works in each mesh's local coordinate system [4], we divide this world-space target by the part's mean local scale s_{part} before passing it to the modifier:

$$d_{\text{local}} = \frac{\alpha\, D}{s_{\text{part}}}, \qquad \alpha = 0.18. \tag{4}$$

We likewise set the noise texture's coordinates to global, so the dent pattern keeps the same physical scale on every car regardless of mesh orientation or local scale.

Damage Targeting Not every mesh in a model is a valid damage target: a car file contains glass, lights, mirrors, interior components, and small accessories that should not receive a dent. The pipeline therefore selects candidates through a sequence of filters. It rejects a part with fewer vertices than a fixed threshold (100 vertices), which removes small accessories. It rejects a part whose bounding-box centre lies in the lowest quarter of the car's height, which removes underbody parts, or whose largest world-space dimension falls below 15% of the car's overall size. Finally, it rejects a part whose name contains a non-panel keyword such as glass, wheel, lamp, mirror, or badge. Section 5.3 explains why the size filter is needed as a backstop to the name filter.

Damage is targeted by zone instead of individual parts. We assign each surviving candidate to one or more of four damage zones (front, rear, left, and right) according to its bounding-box centre in the coordinate frame fixed during model setup. Because the assignment uses world-space position and not mesh names, we can request damage at a semantic level (*damage the rear of the car*) on any model, whatever its naming convention. This name-independent scheme handles most differences between source models, but two of the five cars expose its limits. We describe both cases in Section 5.3.

Randomisation and Viewpoint Sampling We randomise the conditions under which each image is produced so the classifier cannot key on the rendering conditions. Section 2.6 gives the motivation. Two properties vary: the lighting and the background. Both come from a set of high-dynamic-range environment maps (HDRIs), each an outdoor or indoor scene that lights the car and forms its backdrop, so switching the map between renders changes the lighting and background together (Figure 4). We sample the camera viewpoint as well. We place the camera on a sphere around the car at a distance drawn uniformly between 6 and 8 Blender Units, with its angle drawn for each image from one of five named viewpoint regions (top, front, left, right, and rear). Repeated renders of the same car and damage state therefore differ in framing. We express the camera position in spherical coordinates:

$$(x, y, z) = \big(r \sin\phi \cos\theta,\ r \sin\phi \sin\theta,\ r \cos\phi\big), \qquad r \sim \mathcal{U}(6,\, 8) \tag{5}$$

Where ϕ is the polar angle from vertical, ranging from 6° (near top-down) to 72° (near horizontal), and θ the azimuth that controls the direction around the car. Each named viewpoint corresponds to a defined range of these angles: the *top* region spans 6° to 45° and the four lateral regions span 45° to 72° over their respective quadrants, and we sample a specific position uniformly within that range.

Visibility Labelling Whether the damage is visible in an image depends on the relationship between the damaged zone and the camera viewpoint: a dent on the rear is plainly visible from a rear view, partly visible from the side, and not visible from the front. We derive the observability label introduced in Section 3.1 deterministically from this relationship (Figure 2 shows the geometry); Figure 6 shows the three resulting levels on rendered samples. Writing z for the damaged zone, c

for the camera viewpoint, and adj(z) for the two quadrants adjacent to z, the observability label is

$$v(z, c) = \begin{cases} \textit{partially\_seen} & c = \textit{top}, \\ \textit{easily\_seen} & c = z, \\ \textit{partially\_seen} & c \in \text{adj}(z), \\ \textit{hidden} & \text{otherwise.} \end{cases} \tag{6}$$

A damaged image is thus labelled *easily_seen* when the camera faces the damaged zone directly, *partially_seen* when the camera is overhead or in an adjacent quadrant, and *hidden* when the camera is in the opposite quadrant from the damage. Recording this label for every image is what lets us measure how detection accuracy depends on whether the damage is in view.

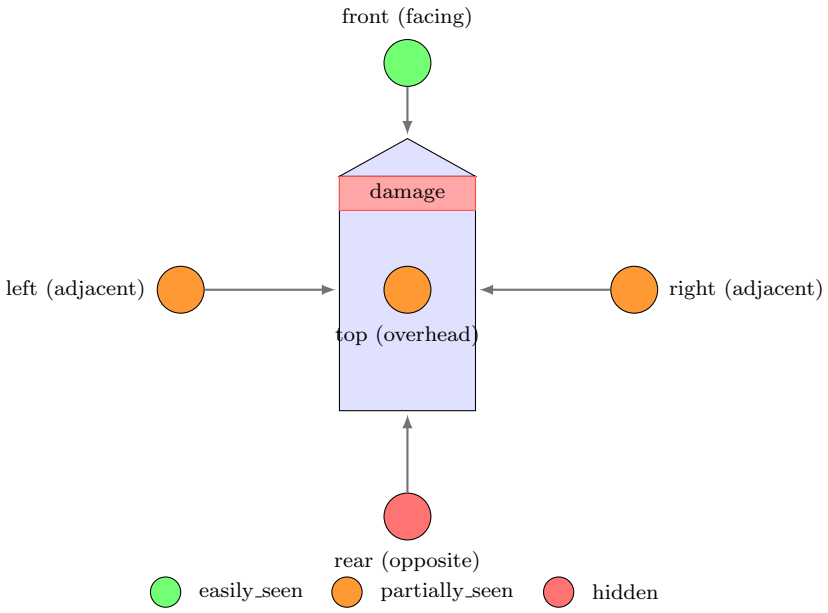

Figure 2: Observability geometry for one damaged zone (in this example, in the front), corresponding to Equation (6). The viewpoint facing the damage labels the image *easily_seen* (green). The two adjacent side viewpoints and the overhead *top* viewpoint label it *partially_seen* (orange). The opposite viewpoint labels it *hidden* (red). Viewpoint placement is schematic.

Because the label is a heuristic based on zone and viewpoint rather than a per-pixel test of what the camera sees, we checked it directly. For each car we sampled images labelled *hidden* and inspected them by eye to confirm that the damage was indeed not visible. On four of the five cars the label was faithful: damage marked hidden was occluded, with only mild leakage at the extreme camera angles inside a viewpoint range. This leakage comes from sampling a range of angles within each viewpoint: images near the edge of a range catch a sliver of the damaged zone. We treat it as expected in-range variation, not a labelling error. The Golf Mk7 is the exception. Its whole-body deformation (Section 5.3) is visible from every viewpoint, so its images labelled hidden still show damage. For this reason we exclude the Golf Mk7 from the visibility analysis in Section 4.3.

Render Loop and Dataset Composition The full dataset is produced by iterating over every combination of car model and environment map, and for each combination the pipeline renders

an intact pass and a damaged pass. In the intact pass the car carries no damage and is rendered from each of the five viewpoints twelve times, with the camera resampled within the viewpoint range on every render. In the damaged pass we apply damage to each of the four zones in turn, and for each zone we again render from all five viewpoints twelve times. Every image is written to disk with an entry in a single metadata file. The entry records the class (intact or damaged), car model, environment map, and viewpoint. For damaged images it also records the damaged zone, the deformed parts, the computed visibility, and the exact camera angles. The composition is fully factorial: across N_{car} cars, N_{hdri} environment maps, one intact pass plus N_{zone} damaged zones, and N_{view} viewpoints each rendered R times, the dataset size is

$$|\mathcal{D}| = N_{\text{car}}\, N_{\text{hdri}}\, (1 + N_{\text{zone}})\, N_{\text{view}}\, R = 5 \cdot 6 \cdot 5 \cdot 5 \cdot 12 = 9{,}000, \tag{7}$$

of which 1,800 are intact and 7,200 damaged. The damaged images outnumber the intact ones because the damaged pass covers four zones and the intact pass none. We handle the resulting class imbalance during training, as described in Section 3.3.

Generated Samples Figure 3 and Figure 4 show what the pipeline produces. Figure 3 is the five-vehicle pool: two sports coupés, two hatchbacks, and one van, the body-shape variation whose coverage Section 4.2 tests. Figure 4 holds the vehicle and viewpoint fixed and varies only the environment map. It isolates the lighting and background randomization of Section 2.6.

Both figures also expose a property of the viewpoint sampling. Every tile is a *front* render, yet the car sits at a slightly different angle in each, because Equation (5) draws a fresh angle within the viewpoint's range on every render. A named viewpoint is therefore a distribution over nearby poses, not a single fixed one, and this within-viewpoint variation is itself part of the randomization the dataset applies.

Figure 3: The five vehicle models in the dataset, each shown intact under the same environment and the *front* viewpoint. Three body categories: two sports coupés (GT3 RS, Corvette), two hatchbacks (Golf Mk1, Golf Mk7), and one van (ID Buzz). This composition, two categories represented by a pair and one by a single model, is what the leave-one-car-out analysis of Section 4.2 exploits.

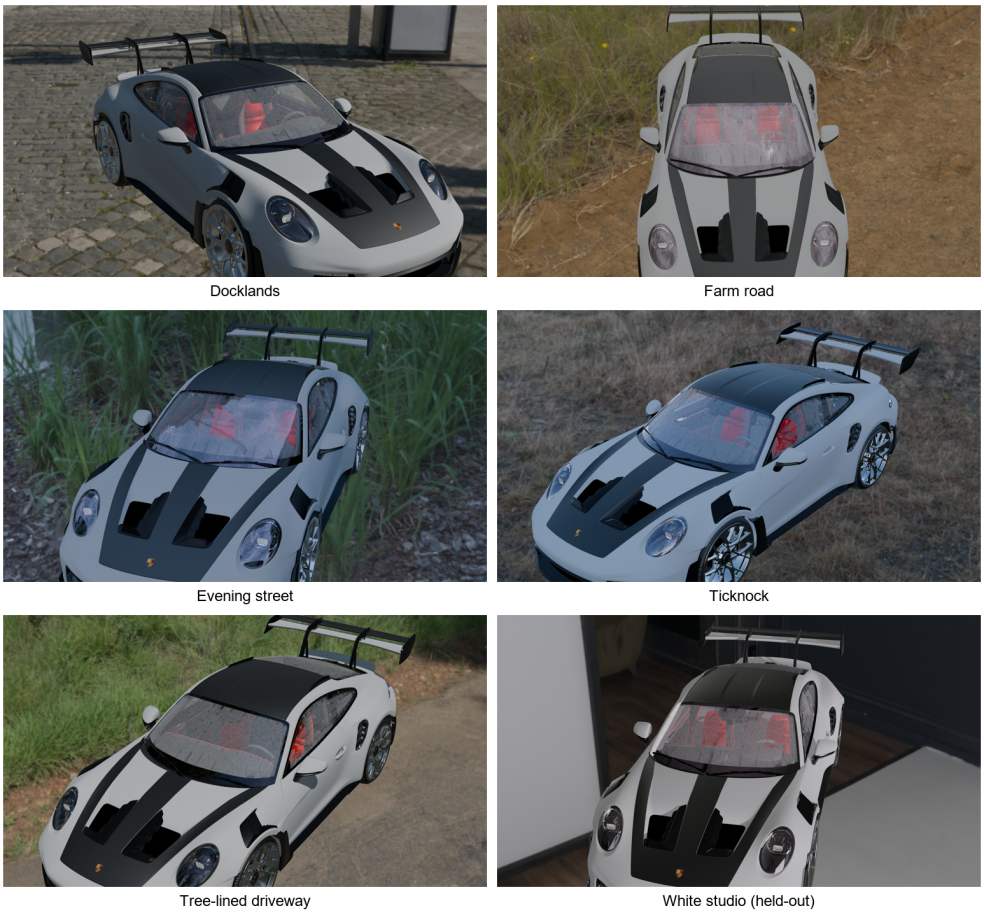

Figure 4: The same vehicle (GT3 RS), rendered from the *front* viewpoint under the six HDRI environment maps used by the pipeline. Each map supplies both the lighting and the background, so switching maps between renders varies the two together. The *white_home_studio* environment (bottom right) is held out of training for the cross-environment generalization test (Section 4.2).

3.3 Classifier Training

We use a ResNet-18 convolutional neural network, fine-tuned from weights pretrained on ImageNet. Section 2.3 sets out why we fine-tune and why we don't train from scratch. We replace the network's final fully connected layer (which on ImageNet produces a thousand-way classification) with a new layer of two outputs for the intact and damaged classes, and keep the pretrained weights of all other layers as the starting point. The two outputs are logits z_0 and z_1, converted to class probabilities by the softmax function

$$p_c = \frac{e^{z_c}}{e^{z_0} + e^{z_1}}, \qquad c \in \{0, 1\}. \tag{8}$$

The larger of p_0 and p_1 gives the predicted class, and its value is the confidence score that the active-vision loop later reads (Section 3.4). We then fine-tune the whole network on the synthetic dataset, adapting the pretrained visual features to the damage-detection task.

We resize and center-crop each image to 224×224 pixels (the input size the pretrained network expects), and normalise it with the ImageNet channel statistics. During training, we apply light

augmentation on top of the variation already built into the renders: a 224×224 random crop from a 256-pixel resize, horizontal flips, and colour jitter of ± 0.2 in brightness and contrast. Because the dataset already varies viewpoint and lighting, we keep this augmentation mild. It adds minor pixel-level diversity and doesn't compensate for missing variety in the underlying images.

We divide the dataset into training, validation, and test sets. We hold out one car and one environment map entirely: the test set is every image that involves the held-out car or the held-out HDRI, with no overlap with training. Test images therefore differ from training images both in their exact pixels and in the vehicle they depict or the lighting under which they were rendered. From the images that remain after holding out the test set, we drop every damaged image whose damage is *hidden* (occluded from the camera, Section 3.2). Such an image attaches a damaged label to a view with no visible damage, which would teach the classifier to read an intact-looking panel as damaged. We split the rest between training and validation in roughly an 85/15 ratio. The test set keeps all three visibility levels, so the *hidden* images held out of training still appear there for the per-visibility analysis of Section 4.3.

The dataset is class-imbalanced: the four-zones-per-damaged-pass generation structure produces four times as many damaged images as intact, a 1:4 ratio. Trained on that distribution directly, the network would minimise its loss most cheaply by defaulting to the majority class. It would predict *damaged* on every image, reaching 0.80 accuracy with no learned features. We prevent this by weighting the classification loss inversely to class frequency, so that errors on the smaller intact class carry proportionally more gradient signal. Concretely, for an image of true class c with predicted probability p_c (Equation (8)), the training objective is the weighted cross-entropy

$$\mathcal{L} = -\, w_c \log p_c, \qquad w_c = \frac{N}{2\, N_c}, \tag{9}$$

where N_c is the number of training images of class c and $N = N_0 + N_1$ the total, so the rarer intact class receives the larger weight. We weight the loss instead of oversampling the minority class: a weighted sampler would re-show the under-represented intact images several times per epoch, and with only 1,800 intact images (many sharing a car, an HDRI, and a similar viewpoint) that repetition risks the network memorising the rendering artefacts of individual images instead of the intact concept. Loss-weighting reaches the same balance without duplicating data.

We train with the Adam optimizer [16] (learning rate $\eta = 10^{-4}$, weight decay 10^{-4}) and a cosine learning-rate schedule [21], at batch size 32, for fifteen epochs. Adam adapts the step size for each parameter from running estimates of the gradient's first and second moments: writing $\hat{m}_t$ and $\hat{v}_t$ for the bias-corrected estimates at step t, each parameter θ is updated as

$$\theta_{t+1} = \theta_t - \eta\, \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}, \tag{10}$$

and the cosine schedule decays η smoothly toward zero across the fifteen epochs. After each epoch, we evaluate on the validation set and retain the checkpoint with the lowest *unweighted* validation loss as the final model. We compute this validation loss without the class weights used in training, so that it reflects the real, imbalanced distribution that the model is tested on. The checkpoint we keep is the one that generalises on that distribution. We select on loss instead of on accuracy because accuracy on a small, imbalanced validation set is a noisy, discretized signal: it jumps in coarse steps as individual predictions flip across the decision boundary. Thus, an epoch that flips a few borderline intact predictions can post a high accuracy reading on a model that has already

begun to overfit. Validation loss is continuous and reflects the model's confidence on both classes, so it tracks generalisation through a smoother early-stopping signal.

3.4 Active Vision

The classifier in Section 3.3 produces a prediction from a single image and ignores the fact that the same car can be photographed from many angles. We add an active vision component on top of the classifier that treats *which* viewpoint to use as a decision the system makes for itself. It starts from one image, reads the classifier's confidence in that image's prediction, and decides on that basis whether to acquire another view of the same car or commit to the current answer.

We keep the policy deliberately simple. The research interest is whether the classifier's confidence signal carries enough information to guide viewpoint selection at all. A more elaborate controller would obscure which design choices produced any effect. The policy draws viewpoints from the same five named regions used during dataset generation (top, right, left, rear, and front). At inference, it selects an initial viewpoint at random, classifies the image, and reads the classifier's maximum-softmax probability as a confidence score. If this score exceeds a fixed threshold, set to 0.85, the system commits to the prediction. Otherwise, it samples a second, previously unused viewpoint, classifies it, and combines the two predictions by averaging their softmax probabilities. The loop repeats, adding one viewpoint at a time, until either the aggregated confidence exceeds the threshold or it reaches a maximum number of viewpoints.

Two parameters control the policy. The confidence threshold sets how cautious the policy is: a lower value commits earlier and uses fewer views on average. A higher value demands stronger agreement before stopping. The viewpoint cap bounds how many views a single decision can use. The viewpoint set is finite, so the loop would terminate in any case. The cap bounds the worst-case number of views, and hence the cost, of a single decision. We set the threshold to 0.85 and the view cap to three, an operating point where the policy stops after one view on high-confidence inputs and requests up to two more on uncertain ones. We also sweep the threshold over $\{0.70, 0.80, 0.85, 0.90, 0.95\}$ to trace the accuracy-versus-views trade-off.

To test whether the policy does anything useful, we compare it against two baselines that use the same dataset and the same trained classifier. The first is single-view classification: one viewpoint chosen at random, no further information (the bare classifier's prediction taken as the answer). This baseline asks whether multi-view information helps at all. The second is a fixed-budget random multi-view baseline: rather than letting the policy decide when to stop, we sample a fixed number of viewpoints (one through five) at random and average their softmax outputs the same way the policy does. Matching the random baseline to the number of views the policy used on average isolates the contribution of the policy's *decision* from the contribution of simply having more views.

We evaluate each policy on the held-out test set and report two figures: classification accuracy, and the mean number of viewpoints used per scene. Accuracy answers whether the policy classifies correctly. The mean-views figure answers what that accuracy costs. A policy that matches a random multi-view baseline's accuracy using fewer views on average has extracted information from the confidence signal. One that needs more views to reach the same accuracy has not. We also decompose the results by class (intact and damaged) and by the visibility of the damage in the first viewpoint (*easily_seen*, *partially_seen*, *hidden*). This decomposition is the reason for the visibility labelling defined in Section 3.2: it lets us measure separately whether the policy helps when the initial view was already informative and when it was not.

To check that these results are not artefacts of a single random seed, we run each policy four times with seeds 42, 7, 13, and 99 controlling the initial-viewpoint draw and any later sampling. We report means across the four runs together with their standard deviation. This way the spread reveals any result that depends on a single lucky seed.

4 Experiments and Results

This chapter evaluates the dataset. We ask whether it can train a convolutional classifier to tell damaged cars from intact ones, and where that ability breaks down. The classifier and the active-vision loop are not the objects of study. They are the instruments through which we read what the dataset supports. The chapter follows the thesis's four sub-questions: whether the dataset can be generated from open-source tools at all, shown qualitatively in Section 4.1 (SQ1). Whether the dataset trains a classifier that generalizes beyond its rendering conditions, and how far that reach extends (SQ2). Whether the dataset's observability annotation predicts detectability (SQ3). Finally, whether that annotation can drive a downstream acquisition policy (SQ4).

4.1 The Generated Dataset

The first question (SQ1) is whether the dataset can be produced at all from freely available components. It can. The pipeline of Section 3.2, scripted entirely against the Blender Python API and run on publicly available 3D car models, produced the full dataset without manual annotation: 9,000 labelled images of five vehicles, in intact and damaged states, across six lighting environments and five viewpoints. Each image carries its class, vehicle, viewpoint, and (for damaged images) damaged zone and observability level, all written by the generator at render time (Section 3.2). This subsection reports that result qualitatively. The subsections that follow test what the dataset, once generated, is worth.

Producing the dataset is cheap and fully automated. Rendering at 1920×1080 in EEVEE at 64 samples, under Blender 5.0.1 on an NVIDIA GeForce RTX 4050 laptop GPU, the pipeline writes one image every 1.29 seconds. A single vehicle's 1,800 images complete in roughly 39 minutes, and the full set renders unattended with no manual step. Each PNG occupies about 3 MB, so the complete dataset is roughly 27 GB on disk. Because the four pipeline stages are deterministic and resume-safe, the dataset is reproducible and extends to new vehicles or damage types at the same per-image cost.

Its composition is fully factorial (Equation (7)): 1,800 intact and 7,200 damaged images, spread evenly over five vehicles, six lighting environments, and five viewpoints. The observability annotation partitions the damaged images in a fixed 1:3:1 ratio (1,440 *easily_seen*, 4,320 *partially_seen*, and 1,440 *hidden*) since each damaged zone faces exactly one of the five viewpoints, sits opposite another, and is seen only partially from the remaining three (Equation (6)). The five vehicles are public models authored at widely differing native scales, and making procedural damage register consistently across that heterogeneity is the engineering problem taken up in Section 5.3.

Figure 5 shows a representative vehicle (the Corvette) in both states, rendered from the same viewpoint. The procedural displacement reads as physical deformation, a localised bulge or crumple of the panel, distinct from the smooth, regularly shaded surface of the intact car. It appears under the varied lighting and backgrounds that domain randomization introduces (Section 2.6). The

damage is visible as a disruption of the panel. It is not a recoloured or texture-mapped overlay. This is the cue the classifier is asked to learn.

Figure 5: A representative vehicle from the generated dataset, rendered intact (top) and with procedural displacement damage (bottom) from the same viewpoint, under domain-randomized lighting and backgrounds. The damage appears as a localised deformation of the front-left panel, distinct from the smooth intact surface.

The dataset's defining feature is qualitative as well as recorded: the same damage looks different from different viewpoints. Figure 6 shows one damaged zone seen at the three observability levels the generator assigns, *easily_seen*, *partially_seen*, and *hidden*. The deformation that dominates the frame when the camera faces the zone is barely present from the side and absent when the camera sits opposite it. This is the viewpoint dependence the dataset is built to capture. The visible difference between these levels is what the per-view observability annotation labels and what the quantitative test of Section 4.3 measures. Because this label is a geometric zone-and-viewpoint heuristic, a view marked *hidden* can still catch a sliver of the damage at the edge of a viewpoint range (Figure 6, above the right side mirror), as Section 3.2 discusses.

Figure 6: The same front-zone damage on one vehicle at the three observability levels assigned by the generator: *easily_seen* (top, camera facing the damage), *partially_seen* (middle, camera to the side), and *hidden* (bottom, camera opposite). The deformation is prominent from the facing viewpoint, partial from the side, and absent from the opposite one.

4.2 Generalization Across Vehicles and Lighting

Our primary experiment is a leave-one-car-out (LOCO) sweep. The held-out protocol of Section 3.3 removes one car and one environment from training and tests on every image involving either. We run it five times, holding out each of the five cars in turn while keeping the *white_home_studio_4k* environment held out throughout. Each run measures two things at once: how well the dataset trains a detector for the held-out vehicle, and how well it transfers to the unseen lighting. Testing a single held-out car would describe only that car. Running every car turns *does the dataset generalize*

into *which generalization the dataset's composition supports*, the form of the question the rest of the chapter answers. The active-vision experiments (SQ4) reuse the policy, baselines, and four-seed evaluation defined in Section 3.4.

The five cars in the dataset fall into three body categories: two sports coupés (the GT3 RS and the Corvette), two hatchbacks (the Golf Mk1 and the Golf Mk7), and one van (the ID Buzz). Two categories therefore contain a pair, and one contains a single model. The leave-one-car-out sweep tests what this composition is worth: for each car, we ask whether training on the other four produces a classifier that recognises damage on the car it never saw.

Table 1 reports the held-out accuracy for all five cars. The four cars that share their category with another model generalise to within a narrow band: held out, each is classified between 0.739 (Golf Mk1) and 0.822 (Golf Mk7). The ID Buzz is the exception. As the only van, it has no body-category counterpart in training, and held out it falls to 0.423.

Held-out car	Body category	Category mate in training	Accuracy
GT3 RS	sports coupé	yes (Corvette)	0.801
Corvette	sports coupé	yes (GT3 RS)	0.801
Golf Mk1	hatchback	yes (Golf Mk7)	0.739
Golf Mk7	hatchback	yes (Golf Mk1)	0.822
ID Buzz	van	no	0.423

Table 1: Leave-one-car-out generalization. Each row holds out one car entirely and reports the classifier's accuracy on that unseen car ($n = 1800$). The four cars with a body-category mate in training are classified at 0.739-0.822. The lone van, with no mate, falls to 0.423.

These accuracies sit on a test cell that is four-fifths damaged, so a classifier that predicted *damaged* on every image would already score 0.80 (Section 3.3). Figure 7 plots each car against this baseline, and the split is visible at a glance: the four in-category bars stand level with the dashed line while the van's bar drops far beneath it. On the van the classifier does worse than the constant prediction, because it misses damage it was trained to detect. The per-class and per-visibility behaviour behind these numbers (which shows the classifier responds to the damage instead of guessing the majority class) is taken up in Section 4.3.

Lighting is the other axis that the sweep varies, and the dataset covers it better than it covers vehicle shape. Every run also holds out the *white_home_studio_4k* environment, so each trained car is tested under lighting absent from its training. Pooled over the four trained cars in the van's run, accuracy under this unseen environment is 0.778 (within the same band as the in-category held-out cars, and far above the van's 0.423). The dataset's domain randomization therefore transfers across lighting, while transfer across vehicles is bounded by whether the target's body category appears in training.

The reading for the dataset is direct. Its six environment maps span the lighting variation a new scene introduces, but its five cars span body shape too sparsely: a category represented by a single model supports no generalization to that model when it is held out, whereas a category represented by a pair does. Cross-vehicle generalization is a property of the dataset's composition, not of the classifier alone.

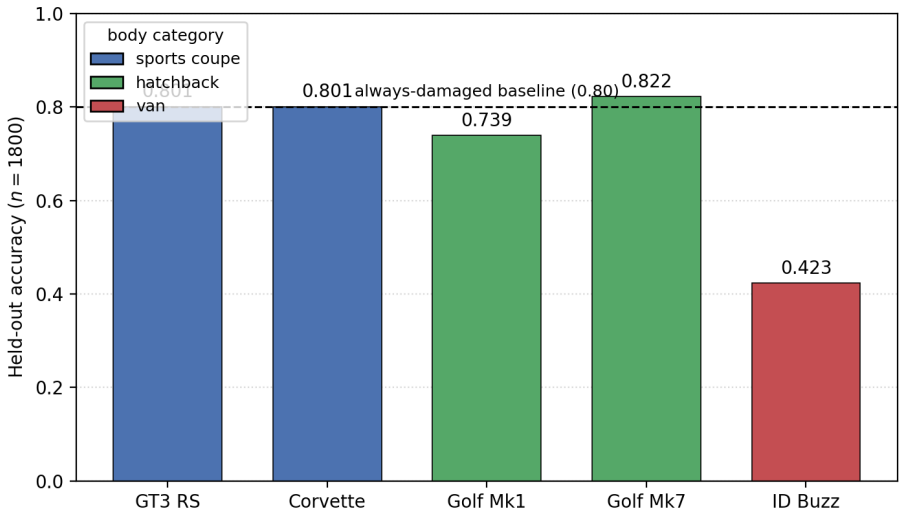

Figure 7: Leave-one-car-out held-out accuracy for each of the five vehicles ($n = 1800$ per bar), coloured by body category. The dashed line marks the 0.80 accuracy a constant *damaged* prediction would reach on the four-fifths-damaged test cell (Section 3.3). The four cars with a body-category mate in training sit at or near the baseline; the lone van, with no mate, falls far below it.

4.3 The Observability Annotation Predicts Detectability

The dataset's novel element is the per-view observability label, which records for each damaged image whether the damage is *easily_seen*, *partially_seen*, or *hidden* from the camera (Section 3.2). For the label to be useful it must predict what it names: damage marked visible should be detected more often than damage marked hidden. Hidden damage is held out of training (Section 3.3), so the drop to the *hidden* column reflects both the occlusion the label marks and the absence of hidden examples in training. The cleaner comparison is *easily_seen* against *partially_seen*, both present in training: detectability already falls as the damage moves from facing the camera to the side.

Each leave-one-car-out run provides an independent test of the label on a different mix of cars. Table 2 reports the four runs and Figure 8 plots them. In all four, accuracy is ordered *easily_seen* > *partially_seen* > *hidden* (the order the label predicts) and the direction never reverses. Figure 8 makes the regularity plain: within each vehicle the three bars step down in that fixed order, and the descent holds whether the vehicle sits near the top of the axis or near the bottom. The gap is widest on the Golf Mk1 (0.877 down to 0.773) and narrowest on the sports coupés (0.981 down to 0.931). The van keeps the same ordering (0.564 down to 0.463) even though its overall detection is poor (Section 4.2): the annotation tracks detectability independently of how well the car itself generalises.

The Golf Mk7 is excluded from this test. Its damage is not confined to the requested zone. It is spread across the whole body (Section 5.3), so an image its label calls *hidden* still shows damage. The label no longer means what it asserts, and its hidden-view accuracy would measure nothing about occlusion.

Run (held-out car)	easily_seen	partially_seen	hidden
GT3 RS	0.981	0.956	0.931
Corvette	0.975	0.957	0.935
Golf Mk1	0.877	0.868	0.773
ID Buzz	0.564	0.538	0.463

Table 2: Damaged-class accuracy by observability level. Each row is one leave-one-car-out run; the cell aggregates every damaged image in that run's test set ($n = 480$ easily_seen, 1440 partially_seen, 480 hidden). In every run the order *easily_seen* > *partially_seen* > *hidden* holds. The Golf Mk7 is excluded (Section 5.3).

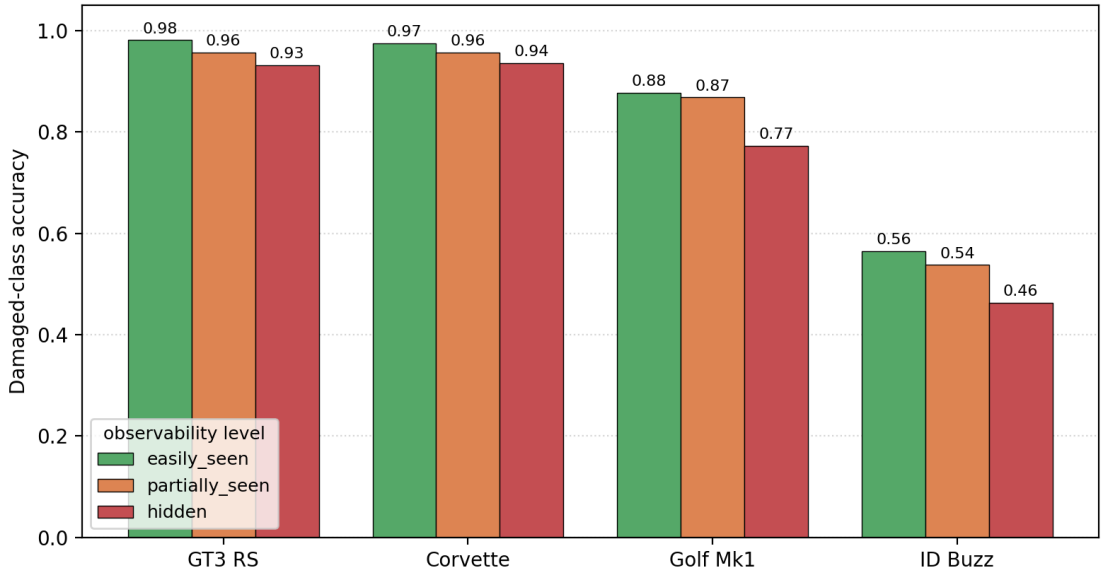

Figure 8: Damaged-class accuracy by observability level, for each leave-one-car-out run (the Golf Mk7 is excluded, Section 5.3). Within every vehicle the accuracy decreases monotonically from *easily_seen* to *partially_seen* to *hidden*, the order the annotation predicts, regardless of the vehicle's overall detection level. The van shows the same ordering at a far lower absolute accuracy (Section 4.2).

The reading for the dataset is that the observability annotation is faithful: a label assigned geometrically at render time predicts whether the classifier can detect the damage. This is what makes the annotation usable as a controlled variable, and it is the signal the active-vision loop in Section 4.4 consumes.

4.4 Downstream Use: Confidence-gated Active Vision

The observability annotation proved to be useful (Section 4.3), so a system could in principle use the classifier's confidence to decide when its current view is too uninformative and another is needed. The active-vision loop of Section 3.4 does exactly this. We evaluate it on the ID Buzz held-out split of Section 3.3 (a 50-scene sample, i.e. 40 damaged, 10 intact, in which the unseen van dominates)

against the single-view and fixed-budget random multi-view baselines defined in Section 3.4.

Confidence gating does not improve accuracy. Single-view classification scores 0.545 (sd 0.071 across the four seeds). The loop scores 0.530 (sd 0.057) while taking 2.26 views per scene against a cap of three. Drawing the same number of views at random rather than by confidence lands in the same place (0.560 at four random views). The per-class picture matches: on the damaged scenes the loop scores 0.475 against single-view's 0.487, and on the intact scenes 0.750 against 0.775 (differences inside the seed noise). The loop spends more than twice the images of a single view and returns no accuracy for them.

The reason is in the confidence the loop gates on. Gating helps only when confidence separates correct predictions from incorrect ones. Within this sample, the two car groups behave oppositely. On the trained cars, correct single-view predictions carry a median confidence of 0.98 and incorrect ones 0.81 (a gap a threshold can act on). On the van the two collapse together: correct predictions sit at 0.75 and incorrect ones at 0.71 (Figure 9). The two panels of Figure 9 show the contrast directly: on the trained cars the correct-prediction confidences pile up near one, well clear of the incorrect ones, whereas on the van the correct and incorrect distributions sit on top of each other. The classifier is no more confident when it is right than when it is wrong, so the gate fires without information and the extra views it buys carry none. This is the calibration that Section 2.7 named as the precondition for confidence-gated acquisition, and the van is where the dataset fails to supply it: the same coverage gap that sinks generalisation in Section 4.2 (a body category with no training counterpart) also strips the confidence signal that the loop would need.

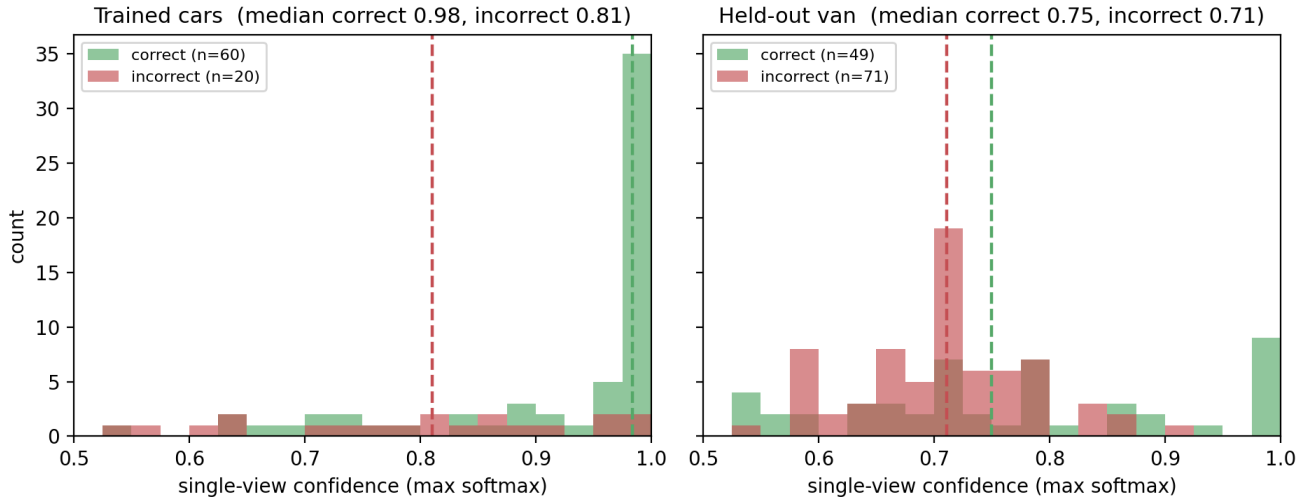

Figure 9: Single-view prediction confidence (maximum softmax probability), split by whether the prediction was correct, pooled over the four active-vision seeds on the ID Buzz held-out split. On the trained cars (left) the classifier is markedly more confident when correct than when wrong (medians 0.98 vs 0.81, dashed lines), so a confidence threshold has a signal to gate on. On the held-out van (right) the two distributions overlap (medians 0.75 vs 0.71): confidence no longer separates correct from incorrect predictions, which is why gating buys the loop no accuracy (Section 4.5 states the sample sizes).

The collapse has a useful by-product: because confidence falls uniformly on the unseen category

instead of staying high on wrong answers, the classifier's failures there are legible in its own confidence. We take up that property in Section 5.2.

The van is an out-of-category vehicle, so the null result above could be a property of that mismatch and not the loop. To separate the two, we repeat the evaluation on a held-out car that does have a body-category mate in training: the Golf Mk7, a hatchback whose mate (the Golf Mk1) is in the training set. The classifier is far more accurate here to begin with. Single-view accuracy 0.825, against 0.545 on the van. However, the loop again fails to raise it. Across the four seeds, the mean accuracy is identical with and without the loop (0.825 single-view, 0.825 active vision at 1.75 views per scene), while the across-seed standard deviation falls from 0.062 to 0.009.

The reduced variance comes from the intact class. Single-view intact accuracy ranges from 0.10 to 0.60 across the four seeds (mean 0.300, sd 0.187): the outcome depends on which viewpoint is drawn first. Aggregating views removes that dependence by committing the classifier's standing bias toward the damaged class deterministically. Intact accuracy is 0.200 at every seed (sd 0.000) and damaged accuracy is 0.981 (sd 0.011), against 0.956 single-view. This is the same behaviour seen on the van, read at a different operating point: the loop amplifies the bias the classifier already holds instead of adding evidence, so on the damaged class it confirms a correct call and on the intact class it confirms a wrong one. On the more accurate in-category car, the amplification surfaces as lower variance. On the van it surfaces as no change at all. In neither case does confidence-gated acquisition improve the mean accuracy, which locates the limit in the classifier the dataset trains.

4.5 Sample Sizes and the Basis for the Findings

The three quantitative results in this chapter rest on different amounts of data. The generalisation table (Section 4.2) is the best supported: each held-out car is tested on 1,800 images, so the leave-one-car-out accuracies and the category-mate pattern they show are not small-sample artefacts. The observability test (Section 4.3) is backed by 480 easily_seen, 1,440 partially_seen, and 480 hidden damaged images per run, and its ordering holds across four independent runs. The active-vision results (Section 4.4) rest on the least data: a 50-scene sample, of which only 10 are intact.

At that size, individual active-vision point estimates carry real uncertainty. An intact accuracy measured on ten scenes is best read as "about one in ten", not as a rate stated to the percentage point. A damaged accuracy on forty scenes has a confidence interval several points wide. We therefore rest no active-vision claim on a single number.

The active-vision conclusion (that confidence gating adds no mean accuracy, and that the limit is the classifier's calibration) is supported across conditions. The loop ties both the single-view and the random multi-view baseline across all four seeds. The confidence separation that the gate needs is present on the trained cars and absent on the van in every seed. The in-category car reproduces the same null at a far higher accuracy. The result survives a max-confidence aggregation rule in place of mean-probability, and adjacent rather than random view selection on the in-category car. No single estimate is precise, but they converge on one reading.

5 Discussion

The previous chapter measured what the dataset supports. This chapter asks what those measurements mean. The dataset itself was built from open-source tools as Section 4.1 shows (SQ1).

We take that as established and interpret what it supports. One result stands on its own: the observability annotation predicts detectability (SQ3), which validates the dataset's defining element. The other two are linked by a single cause. Whether the dataset trains a classifier that generalises to an unseen vehicle (SQ2), and whether that classifier's confidence can drive viewpoint acquisition (SQ4). Both turn on which body categories the dataset covers. We draw that thread first, then note a usable side-effect of the same mechanism. We then return to the pipeline behind the dataset: making it work across heterogeneous vehicle models forced a sequence of design decisions whose residual limits bound what the dataset can be asked to do. We close with the threats to these conclusions and a plain account of what the thesis can and cannot claim.

5.1 Coverage is the Limiting Factor

The generalisation result and the active-vision result answer different questions but share one cause. A classifier trained on the dataset generalises to a held-out vehicle when that vehicle shares a body category with a training car, and fails when it does not: the four in-category cars are recognised at 0.74 to 0.82, the lone van at 0.42 (Section 4.2). The active-vision loop then fails to recover the van. This is not because the policy is weak, but because the classifier's confidence collapses on it, so that correct and incorrect predictions carry almost the same confidence (Section 4.4). These are two views of one phenomenon. The body category the dataset does not cover is the same category on which the classifier is both inaccurate and miscalibrated.

The link runs through what the classifier learned. When a held-out car has a body-category mate in training, the classifier has fitted a car of nearly the same shape, so the held-out car stays inside the distribution it knows and its predictions remain separable. The gate has a signal to act on. When there is no mate, the held-out shape is out of distribution, accuracy drops, and confidence flattens, so there is no signal left to gate on. The loop inherits whatever calibration the dataset's coverage produced. It cannot manufacture a confidence signal that the classifier does not carry.

This places the binding constraint of the downstream system where the active-vision literature usually does not look. That literature treats the policy as the thing to improve: better view selection, better aggregation, learned acquisition (Section 2.7). On this dataset none of those would help, because the policy is not what fails. The classifier's calibration is, and that calibration is fixed by which body categories the dataset contains. The effective lever is therefore the dataset: a van counterpart in training would do more for the loop than any redesign of the loop itself.

The dataset trains a usable detector and supports a confidence-gated loop exactly as far as its body-category coverage reaches, and no further.

5.2 Confidence Collapse as a Usable Signal

The confidence collapse that disables the active-vision loop on the van (Section 4.4) has a second, more useful reading. On the out-of-category vehicle the classifier does not fail silently: it reports low, flat confidence on the cases it gets wrong as readily as on those it gets right. Its incompetence is legible in its own output.

For a deployed inspection system this is the distinction that matters, more than the accuracy figure itself. The dangerous failure is not a wrong answer but a wrong answer delivered confidently, because that is the one a downstream process acts on without review. A classifier whose confidence

collapses outside its training coverage can be made to abstain instead: the same confidence threshold the active-vision loop uses to decide whether to acquire another view can equally decide to withhold a prediction and refer the case to a human. The system then fails by asking for help, not by committing a confident error.

We do not overstate this. The collapse was observed on a single out-of-category vehicle, and we neither built nor tuned an out-of-distribution detector around it. The narrow, defensible claim is that the same miscalibration that bounds the dataset's downstream usefulness (Section 4.4) also makes the system's failures self-announcing. This is a property worth carrying into any deployment of a classifier trained this way.

5.3 Building the Dataset Across Heterogeneous Models

Producing this dataset was an engineering problem before it was an empirical one. A procedural damage pipeline for one hand-picked car is straightforward. One that runs across five models sourced from different libraries, exported at scales that differ by three orders of magnitude, modelled at different polygon densities, and named by different conventions, is not. Each axis of that heterogeneity broke the same pipeline in a different way, and each fix (Section 3.2) had to hold without undoing the others. We summarise here what those fixes cost, because their residual limits bound what the dataset can be asked to do.

Three fixes carry the pipeline across the heterogeneity, and each leaves a boundary. Normalising every model to a common size, and expressing the dent depth as a fraction of that size, keeps the damage proportional. However, the world-to-local conversion it relies on assumes each part's scale is near-uniform across its axes. A strongly anisotropic part would dent unevenly. Subdividing each panel before displacing it guarantees a visible dent on even the lowest-polygon car, but at a render cost set by that worst-case car. Selecting damage targets by physical size instead of by the name makes the filter work on generically named models, but, as a heuristic, it would admit a large non-panel part, a roof rack or a bumper bar, that a richer model set might contain. None of the five cars triggers these boundaries.

Two cars reach those boundaries already, and we keep both as documented limits (they are not discarded). The ID Buzz is authored with its length along the wrong axis, so its zones are mislabelled, until corrected by a single per-model rotation. The Golf Mk7 exports its whole body as one mesh, so requested damage spreads across the entire car. Its *damaged* label stays correct but its visibility label does not, which is why it is excluded from the observability analysis (Section 4.3) while remaining in the generalisation results. The general point is that procedural damage on heterogeneous public CAD assets cannot be made fully model-invariant.

One piece of evaluation infrastructure earned its place the same way. The first end-to-end run reported a single low accuracy that read like a model which had failed to learn the task. A per-car breakdown of the same test set decomposed that number in seconds and showed the opposite: the classifier worked on the trained cars under unseen lighting and failed only on the out-of-category vehicle (Section 4.2). The heterogeneity of the dataset surfaces in the results as per-cell structure, and a cheap per-cell diagnostic is what makes that structure visible at the moment it matters.

5.4 Threats to Validity

The largest threat is that every result rests on rendered images. The reality gap (Section 2.5) applies in full: we evaluated the dataset, and the classifiers it trains, only on synthetic images and never on photographs of real cars. Domain randomization is meant to make the learned features survive that gap (Section 2.6), and the held-out-environment result (Section 4.2) shows the features transfer across rendered lighting. Rendered lighting is not however photographic evidence. No claim in this thesis extends to real vehicles. We read the cross-environment generalisation as a within-simulation proxy for the reality gap.

Every finding is also bounded to one dataset configuration: five car models across three body categories, six environment maps, and a single damage type, geometric surface displacement. The central generalisation result (that a body category covered by one model does not survive being held out, while a paired category does, Section 4.2) rests on three categories and a single van. A larger and more balanced fleet could confirm that pattern or expose a category that breaks it. The same bound applies to the damage itself: displacement produces dents and crumples, not the scratches, cracks, or paint damage real inspection also covers, so the detector's reach is the damage the dataset contains and no wider.

The active-vision results rest on the least data, a 50-scene held-out sample (Section 4.4). We therefore tie no active-vision claim to a single point estimate. The conclusion is the direction that replicates across four seeds, two aggregation rules, and an in-category control car (Section 4.5). The generalisation and observability results, tested on 1,800 and up to 1,440 images per run respectively (Section 4.5), do not carry this caveat.

One mechanism behind the central finding is observed. The classifier carries a standing bias toward the damaged class that the loop amplifies on intact panels (Section 4.4), yet we did not test *why* intact panels are misread as damaged. The leading candidate is specular highlights: a bright reflection on an undamaged panel under some environment maps can mimic the shading discontinuity a real dent produces. We flag this as a hypothesis that we did not test. Isolating it would need a controlled comparison that the present experiments do not provide.

5.5 What Can and Cannot be Claimed

The evidence licenses three positive claims about what the dataset supports, one each for SQ2 through SQ4. SQ1 (that the dataset can be generated at scale from open-source tools) is settled by its existence (Section 4.1). For SQ2, the dataset trains a classifier that generalises across lighting (0.778 on a held-out environment) and across vehicles as far as its body-category coverage reaches: a held-out car with a category mate in training is recognised at 0.739 to 0.822, while the uncovered van falls to 0.423 (Section 4.2). For SQ3, the per-view observability annotation is faithful on the four cars. Their geometry is the one the heuristic fits, with detection accuracy following the order that the label predicts, *easily_seen* > *partially_seen* > *hidden*, in every leave-one-car-out run (Section 4.3). For SQ4, confidence-gated aggregation does not raise mean accuracy. It removes its dependence on the lucky first view. On an in-category held-out car, the seed-to-seed standard deviation falls from 0.062 to 0.009 at an unchanged mean (Section 4.4), and the confidence collapse on the uncovered category doubles as a usable out-of-distribution flag (Section 5.2).

The evidence does not license three further claims. We cannot claim transfer to real photographs: every image is rendered and we never tested on real cars, so the reality gap stands open (Section 2.5).

We cannot claim that active vision improves accuracy over a single view: across both held-out cars its mean ties the single-view and random multi-view baselines, so its contribution is reduced variance, not higher accuracy (Section 4.4). And we cannot claim generalisation to a body category absent from training: the van result shows the dataset's five models span vehicle shape too sparsely for that. This is the same coverage limit identified in Section 4.2.

6 Conclusions and Future Work

This thesis asked whether a procedurally generated, domain-randomized synthetic dataset, annotated with per-view damage observability, can train a convolutional classifier that distinguishes intact from damaged vehicles and generalizes beyond the conditions it was rendered under. The answer is a qualified yes: the dataset trains a usable detector, but only as far as its composition reaches.

The four sub-questions make that boundary precise. The dataset can be built at scale from open-source components alone — the Blender Python API and publicly available 3D car models — with every label produced by the generator and no manual annotation (SQ1, Section 4.1). Once built, it trains a classifier that generalizes across an unseen lighting environment (0.778) but across vehicles only as far as its body-category coverage extends: a held-out car with a category counterpart in training is recognised at 0.739 to 0.822, while the lone van, with no counterpart, falls to 0.423 (SQ2, Section 4.2). The per-view observability annotation is faithful — detection accuracy follows the order the label predicts, *easily_seen* > *partially_seen* > *hidden*, in every run (SQ3, Section 4.3). And the annotation supports a downstream confidence-gated acquisition loop, though that loop does not raise mean accuracy over a single view (SQ4, Section 4.4).

The single thread through these results is that the dataset's body-category coverage, not the classifier or the acquisition policy, is the binding constraint (Section 5). The same missing category that sinks cross-vehicle generalization also collapses the classifier's confidence, and the active-vision loop can only inherit the calibration the dataset produced, not manufacture one it lacks. The effective lever on the downstream system is therefore the dataset itself. The one silver lining is that this collapse is legible: outside its coverage the classifier fails with low, flat confidence rather than confident error, which the same threshold can turn into a signal to abstain.

The contribution is the dataset and its observability annotation, together with the pipeline that produces both. Its value, and its limit, is that everything the classifier can and cannot do is traceable to what the dataset does and does not contain.

6.1 Future Work

Because the central finding is about the dataset's composition, the most direct extension is to broaden the dataset itself. Cross-vehicle generalization is bounded by body-category coverage, so a larger and more balanced fleet (in particular a second van to give that category a counterpart) would test whether the pattern holds and would extend the dataset's reach. The same argument applies to the damage: the pipeline produces geometric displacement (dents and crumples), so adding the scratches, cracks, and paint damage that real inspection also covers would widen the conditions that the dataset can train for. Both are changes to the dataset, and the pipeline is built to absorb them at the same per-image cost (Section 4.1).

Because the dataset is generated deterministically from scripted, resume-safe stages, it is also

straightforward to release. Publishing it with a persistent identifier would let others reproduce these experiments, grow the fleet, and build on the observability annotation directly.

A third direction is to validate the dataset beyond simulation. Every result here rests on rendered images. We never tested against photographs of real damaged cars (Section 5.4). Measuring how far a classifier trained on the dataset transfers to real inspection photographs would show whether its domain randomization closes the reality gap in practice, and not only across the rendered lighting (Section 4.2).

Finally, a secondary, method-level direction follows from locating the bottleneck in the classifier's calibration rather than the acquisition policy. Post-hoc calibration such as temperature scaling could test whether a better-calibrated confidence signal restores the loop's leverage on the categories the dataset leaves uncovered, turning the confidence collapse into a deliberate out-of-distribution detector.

References

[1] Elie Aljalbout, Jiaxu Xing, Angel Romero, Iretiayo Akinola, Caelan Reed Garrett, Eric Heiden, Abhishek Gupta, Tucker Hermans, Yashraj Narang, Dieter Fox, Davide Scaramuzza, and Fabio Ramos. The reality gap in robotics: Challenges, solutions, and best practices, 2025.

[2] Ruzena Bajcsy. Active perception. *Proceedings of the IEEE*, 76(8):966–1005, 1988.

[3] Black Snow. Porsche gt3 rs (3d model). Sketchfab, https://sketchfab.com/3d-models/porsche-gt3-rs-e738eae819c34d19a31dd066c45e0f3d. Licensed under CC BY 4.0.

[4] Blender Documentation Team. *Displace Modifier*, 2025. Blender 5.1 Manual. https://docs.blender.org/manual/en/latest/modeling/modifiers/deform/displace.html.

[5] Blender Documentation Team. *Subdivision Surface Modifier*, 2025. Blender 5.1 Manual. https://docs.blender.org/manual/en/latest/modeling/modifiers/generate/subdivision_surface.html.

[6] Ddiaz Design. 1976 volkswagen golf gti mk1 (3d model). Sketchfab, https://sketchfab.com/3d-models/1976-volkswagen-golf-gti-mk1-1fc46cb37bd748e3bb9355fcedaf3817. Licensed under CC BY-NC-SA 4.0.

[7] Ddiaz Design. 2016 volkswagen golf gti clubsport mk7 (3d model). Sketchfab, https://sketchfab.com/3d-models/2016-volkswagen-golf-gti-clubsport-mk7-492301d401ea4fb7bb1cb63e980051d0. Licensed under CC BY-NC-SA 4.0.

[8] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Fei-Fei Li. In *ImageNet: a Large-Scale Hierarchical Image Database*, pages 248–255, 06 2009.

[9] Hoang Van Dung, Nhan Huynh, Nguyen Tran, Khanh Lê, Thi-Minh-Chau Le, Ali Selamat, and Hien Nguyen. Powering ai-driven car damage identification based on vehide dataset. *Journal of Information and Telecommunication*, 9:1–19, 06 2024.

[10] Mahavir Dwivedi, Hashmat Shadab Malik, S. N. Omkar, Edgar Bosco Monis, Bharat Khanna, Satya Ranjan Samal, Ayush Tiwari, and Aditya Rathi. Deep learning-based car damage classification and detection. In Niranjan N. Chiplunkar and Takanori Fukao, editors, *Advances in Artificial Intelligence and Data Engineering*, pages 207–221, Singapore, 2021. Springer Nature Singapore.

[11] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. On calibration of modern neural networks. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, Proceedings of Machine Learning Research, 2017. arXiv:1706.04599.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[13] Insurance Europe. European insurance in figures, 2019 data. Statistical report, Insurance Europe, 2021. [PDF] Available at European Insurance in Figures.

[14] Dinesh Jayaraman and Kristen Grauman. Look-ahead before you leap: end-to-end active recognition by forecasting the effect of motion, 2016.

[15] JUST GAME. Chevrolet corvette stingray (3d model). Sketchfab, https://sketchfab.com/3d-models/chevrolet-corvette-stingray-9dbce926ee86443f9ec8d7c4a1d231cd. Licensed under CC BY 4.0.

[16] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[17] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, page 1097–1105, Red Hook, NY, USA, 2012. Curran Associates Inc.

[18] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[19] Donggeun Lee, Juyeob Lee, and Eunil Park. Automated vehicle damage classification using the three-quarter view car damage dataset and deep learning approaches. *Heliyon*, 10(14):e34016, 2024.

[20] Pei Li, Bingyu Shen, and Weishan Dong. An anti-fraud system for car insurance claim based on visual evidence. arXiv preprint arXiv:1804.11207, 2018.

[21] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts, 2017.

[22] Ruo-Syuan Mei, Sixian Jia, Guangze Li, Soo Yeon Lee, Brian Musser, William Keller, Sreten Zakula, Jorge Arinez, and Chenhui Shao. Hybrid synthetic data generation with domain randomization enables zero-shot vision-based part inspection under extreme class imbalance. *Journal of Manufacturing Processes*, 168:221–232, 2026.

[23] Mordor Intelligence. Motor insurance market size share analysis - growth trends and forecast (2026 - 2031). https://www.mordorintelligence.com/industry-reports/global-motor-insurance-market, 2026. Market research report.

[24] Kalpesh Patil, Mandar Kulkarni, Anand Sriraman, and Shirish Karande. Deep learning based car damage classification. In *Proceedings of the 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 50–54, 12 2017.

[25] Sergio A. Pérez-Zarate, Daniel Corzo-García, Jose L. Pro-Martín, Juan A. Álvarez García, Miguel A. Martínez-del Amor, and David Fernández-Cabrera. Automated car damage assessment using computer vision: Insurance company use case. *Applied Sciences*, 14(20), 2024.

[26] Jinghuan Shang and Michael S. Ryoo. Active vision reinforcement learning under limited visual observability, 2023.

[27] Sloftm_Carz. Volkswagen id. buzz (3d model). Sketchfab, https://sketchfab.com/3d-models/volkswagen-id-buzz-9e7a3d129960447bb353a444185c47c3. Licensed under CC BY 4.0.

[28] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world, 2017.

[29] R.E. van Ruitenbeek and S. Bhulai. Convolutional neural networks for vehicle damage detection. *Machine Learning with Applications*, 9:100332, 2022.

[30] Xinkuang Wang, Wenjing Li, and Zhongcheng Wu. Cardd: A new dataset for vision-based car damage detection. *IEEE Transactions on Intelligent Transportation Systems*, PP:1–13, 07 2023.

[31] Wenshuai Zhao, Jorge Peña Queralta, and Tomi Westerlund. Sim-to-real transfer in deep reinforcement learning for robotics: a survey. In *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 737–744, 2020.

[32] Xiaomeng Zhu, Jacob Henningsson, Duruo Li, Pär Mårtensson, Lars Hanson, Mårten Björkman, and Atsuto Maki. Domain randomization for object detection in manufacturing applications using synthetic data: A comprehensive study, 2025.