

From Benchmark to Browser: Reproducible Evaluation and CPU-Only Improvement of an On-Device Car-Damage Detector on CarDD

[Author names — to be completed]

Çekicin AI Lab (independent research prototype for roadside-assistance damage pre-assessment)

Working draft v0.1 — 19 October 2026 — not peer reviewed

Abstract. Published car-damage detectors are rarely evaluated in the form in which they are deployed. We benchmark the detector of a browser-based damage pre-assessment tool — a YOLO11m model quantised to int8 ONNX and executed client-side — on the official CarDD test split (374 images) with the COCO box protocol used by the CarDD authors. The deployed model reaches 59.9 box AP / 75.7 AP50, below the AP50 of 89.9 claimed on its public model card, and statistically on par with the strongest model of the CarDD paper (DCN+ R101: 60.6 / 78.8). Horizontal-flip test-time augmentation fused with Weighted Boxes Fusion raises this to 62.23 AP / 77.96 AP50 and improves small-object AP from 24.9 to 31.61, with a bit-exact JavaScript port. We then ask how far the frozen detector can be improved with CPU-only training on CarDD. We find that the detector has memorised the training split (79.4 AP on train vs. 64.8 on val), which makes training-split predictions unusable for stacking; a gradient-boosted rescorer fitted on val with grouped cross-validation adds only 0.24 AP on test (paired 95% CI +0.03 to +0.42). Slicing-aided inference on 2×2 tiles changes test AP to 62.17 (careful mode). Alongside, we review 100 car-damage papers (2019–2026) and distil 100 actionable practices. All predictions, scripts, models and logs are released.

Keywords: car damage detection; CarDD; on-device inference; test-time augmentation; weighted boxes fusion; learned rescoring; reproducibility

1. Introduction

Roadside-assistance operators receive photos of damaged cars from drivers who need to know quickly whether the car can be driven, whether a tow truck is required, and what kind of damage is visible. A detector that runs in the phone browser avoids uploading photos, works on weak networks and costs nothing per request; but it must be small, quantised and fast, and its accuracy must be known under exactly these conditions.

The literature reports many numbers for car-damage detection, but they are hard to compare: in the 100 papers we reviewed, 27 use private insurer data, only 7 use the public CarDD benchmark [1], and only 24 state a quantitative result in the abstract. Model cards on public hubs add further claims that are seldom independently checked.

Contributions. (i) We evaluate a deployed, browser-identical int8 detector (a public checkpoint we did not retrain) on the official CarDD test split with the exact COCO protocol [2] of the CarDD paper, and show that a public model-card claim does not reproduce. (ii) We quantify flip test-time augmentation with Weighted Boxes Fusion (WBF) [3] and ship it with a verified JavaScript port. (iii) We study CPU-only improvements that use the CarDD training data without retraining the network — a learned rescoring stage and slicing-aided inference — under a strict protocol (fit on val, select by grouped cross-validation, test once), and report a largely negative result together with its cause: the detector has memorised CarDD train. (iv) We release a 100-paper structured review with 100 derived practices, and all artefacts (detections, scripts, models, experiment logs).

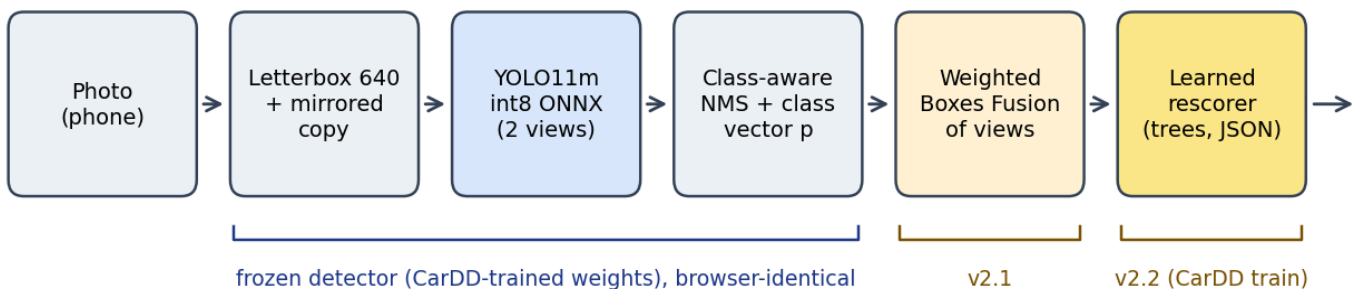


Figure 1. Inference pipeline. Everything runs client-side in the browser; the detector weights are frozen. v2.1 adds the mirrored view and WBF; v2.2 candidates (learned rescorer, tiles) are evaluated in Sec. 5.

2. Related work

Datasets. CarDD [1] provides 4,000 images with 9,163 instances of six damage classes (dent, scratch, crack, glass

shatter, lamp broken, tire flat) and an official 2,816/810/374 split. Multi-view datasets such as CDDM with the MVA-CDD method [4] fuse close-up and distant views; Small Dents [5] targets subtle dents. Our review shows a fast-

growing field (28 papers in 2025 and 24 in 2026 among the 100 selected) dominated by private data.

Detectors. The CarDD authors benchmark Mask R-CNN [6], HTC and deformable convolutions (DCN) [7], and propose DCN+ with multi-scale training and focal loss. Recent work reports YOLOv11 and RT-DETR on CarDD [8], enhanced YOLO variants with a damage-severity index [9], heterogeneous-convolution YOLO [10], diffusion-based detectors [11, 12] and real-time detection transformers such as RF-DETR [13]. Vision–language models are evaluated for insurance tasks [14], grounded with segmentation to reduce hallucination [15], or used for phrase grounding of damage [16].

Post-processing. WBF [3] averages coordinates of overlapping boxes from several models or augmentations. Localisation-quality estimation (IoU-Net [17], VarifocalNet [18]) and learned NMS [19] show that the detector confidence is not always the best ranking signal; our rescorer is a light-weight, post-hoc instance of this idea using gradient-boosted trees [20, 21].

3. System and evaluation protocol

Detector. The deployed detector is a public YOLO11m [22] checkpoint (`shanexf/car-damage-yolo11m-ft`) that its author fine-tuned for 30 epochs on CarDD images plus photos of intact cars with empty labels, to suppress false alarms on undamaged glass, lamps and reflections. We did not retrain it: we exported it to ONNX and dynamically quantised it to int8 (20.6 MB). Inputs are letterboxed to 640×640; the network output holds 8,400 anchors with 4 box values and 6 class scores. The browser applies arg-max classification, class-aware NMS at IoU 0.45 and keeps the top boxes.

Browser parity. We evaluate with a Python port of the browser pipeline. A vectorised re-implementation reproduces all boxes, classes and scores of the original port (100% of compared boxes, $\Delta\text{score} < 10^{-4}$), and the JavaScript WBF matches the Python WBF to 10^{-16} on saved detections.

Protocol. Following the CarDD paper we report COCO box AP (IoU 0.50:0.95), AP50, AP75 and AP for small ($< 32^2$ px), medium and large objects with pycocotools, at confidence ≥ 0.001 and at most 100 detections per image. Mean CPU latency is 496 ms per image and view (2 cores, onnxruntime).

4. Methods

4.1 Flip test-time augmentation with WBF

The photo and its horizontal mirror are processed independently; mirrored boxes are mapped back and both lists are fused with WBF at IoU 0.55. The fused box is the confidence-weighted mean of its members and the fused score is

$$\tilde{s} = \frac{1}{n} \sum_{k=1}^n s_k \cdot \frac{\min(n, L)}{L} \quad (1)$$

where n is the number of member boxes and $L = 2$ the number of views, so boxes seen in only one view are down-weighted.

4.2 Learned rescoring trained on CarDD

For every candidate box we compute 39 cheap features (44 in careful mode): detector score and class, the full 6-class probability vector of the source anchor (its maximum, margin and entropy), geometry (log-area, aspect, position, distance to the border), image context (rank within class, ratio to the best box of the class, number of overlapping same-class boxes, containment in a higher-scoring box, maximum overlapping score per class, image-level maximum per class) and — in careful mode — the per-view scores, their difference, the number of fused members and the IoU between the two views. Multi-label candidates are added for non-arg-max classes with probability ≥ 0.01 .

The target is the localisation quality $q = \text{maximum IoU with a ground-truth box of the same class}$ (0 for false positives). A histogram gradient-boosting regressor predicts $\hat{q}$, and the final score is a geometric blend

$$s^* = \hat{q}^\alpha \cdot s^{1-\alpha} \quad (2)$$

The trees are exported to JSON and evaluated in the browser (identical predictions to scikit-learn on all test candidates when compared in float64).

4.3 Slicing-aided inference

To address the weak small-object AP, the photo is cut into 2×2 tiles with 25% overlap; each tile is letterboxed to 640 px, so objects appear about $1.7\times$ larger. Tile boxes touching an inner tile border are discarded, duplicates across tiles are removed by NMS, and tile boxes are fused with the full-image detections either by WBF, by NMS, or by a complement rule that only adds tile boxes not explained by a full-image box; the tile-score weight and a maximum box area are tuned on val.

5. Experiments

5.1 Deployed model vs. published results

Table 1. Box detection on the official CarDD test split (374 images). Paper rows are taken from Table IV of the CarDD paper [1] unless marked; † = split may differ from the official one.

Method	Source	AP	AP50	AP75	APs
Mask R-CNN R50	CarDD [1]	49.2	66.3	52.6	–
Mask R-CNN R101	CarDD [1]	51.1	67.7	–	–
DCN R101	CarDD [1]	54.3	69.8	–	–
DCN+ R50	CarDD [1]	58.8	77.4	63.1	–
DCN+ R101 (best)	CarDD [1]	60.6	78.8	64.8	37.1
Enhanced YOLOv9 †	[9]	58.0	73.0	–	–
YOLOv11m †	[8]	63.3	–	–	–
YOLO11m int8, deployed (fast)	ours	59.9 (95% CI 56.8–63.2)	75.7	62.0	24.9
+ flip TTA + WBF (careful, v2.1)	ours	62.23 (95% CI 59.5–65.4)	77.96	64.40	31.61

The deployed model matches the best CarDD-paper model in AP and is 3.1 points lower in AP50. The careful mode has a higher point estimate than DCN+ R101 (62.23 vs. 60.6 AP) and is within one point in AP50, but the image-bootstrap 95% interval (95% CI 59.5–65.4) contains 60.6: the two are **on par**, not distinguishable on 374 test images. The gain of careful over fast mode, in contrast, is clear: +2.33 AP (95% CI +1.48 to +3.15; positive in 100% of 150 resamples). The AP50 of 89.9 stated on the public model card of the base weights is **not reproduced** (75.7). The card states that training and evaluation used the author’s own fixed 80/10/10 hash split of the labelled CarDD images rather than the official split; together with the memorisation analysis below (AP50 92.9 on official-train images) this suggests that the card’s test images overlap the official training images, so the claim is not comparable with published CarDD results. Raising the on-device confidence threshold costs accuracy (AP 57.8 at 0.05, 56.4 at 0.25).

Table 2. Per-class box AP / AP50 on CarDD test. The CarDD-paper column is *mask* AP (Table II of [1]) and is shown for orientation only.

Class	fast AP	fast AP50	careful AP	careful AP50	DCN+ R101 mask AP
dent	33.0	58.2	37.7	64.1	40.5
scratch	38.1	66.5	42.1	69.9	34.3
crack	31.6	52.1	32.3	53.8	16.6
glass shatter	93.2	97.2	93.4	97.3	89.6
lamp broken	76.9	88.3	80.9	90.8	70.8
tire flat	86.8	91.4	87.0	91.9	90.0

5.2 Memorisation of the training split

Before fitting anything on CarDD we ran the deployed detector on all three splits. AP is 79.4 on the first 250 training images, 64.8 on val and 59.9 on test (AP50 92.9 / 79.2 / 75.7). The detector has memorised the training images, and val is also optimistic (it was probably used for checkpoint selection). Predictions on train are therefore not representative of deployment, and any second stage fitted on them would learn from over-confident scores. We fit all post-hoc components on **val** and select hyper-parameters by 5-

fold cross-validation grouped by image; test is evaluated once.

5.3 Learned rescoring

Table 3. Learned rescoring on CarDD test. Fitted on val (18163 candidate boxes in careful mode); hyper-parameters (4 tree configurations $\times$ 5 iteration counts $\times$ 4 blends) chosen by out-of-fold AP.

Mode	CV AP before $\rightarrow$ after	test AP before $\rightarrow$ after	test AP50	test APs
fast	64.75 $\rightarrow$ 64.90	59.93 $\rightarrow$ 60.08	75.62 $\rightarrow$ 75.90	25.15 $\rightarrow$ 25.24
careful	66.56 $\rightarrow$ 66.57	62.23 $\rightarrow$ 62.47	77.96 $\rightarrow$ 78.09	31.61 $\rightarrow$ 31.72

Rescoring yields small gains: +0.23 AP (95% CI +0.03 to +0.42; positive in 99% of 150 resamples) in careful mode and +0.16 AP (95% CI +0.00 to +0.33; positive in 97% of 150 resamples) in fast mode (paired image bootstrap [23]). Permutation importance explains why: in fast mode the detector score alone accounts for most of the predictive power (0.87 vs. ≤ 0.05 for any other feature), i.e. the single-view confidence is already close to the best ranking these features allow. In careful mode the per-view scores and the class probability carry complementary signal, which is where the (small) gain comes from. Multi-label candidates alone do not change AP (careful 62.26 vs. 62.23).

5.4 Slicing-aided inference

Table 4. Tiled inference (2 $\times$ 2, 25% overlap) fused with the full-image detections; fusion rule, maximum tile-box area and tile-score weight chosen on val.

Setting	val AP	test AP	test AP50	test APs
Fast (no tiles)	64.75	59.93	75.62	25.15
Fast + tiles (nms, area ≤ 0.1 , w = 0.3)	64.95	59.92	75.65	26.82
Careful (no tiles)	66.56	62.23	77.96	31.61
Careful + tiles (add, area ≤ 0.01 , w = 0.3)	66.48	62.17	77.83	31.27

On val, the best of 48 tile configurations improved fast mode only slightly (+0.20 AP) and none improved careful mode (best -0.08). On test, tiling changes AP by -0.01 (fast) and

-0.06 (careful); paired bootstrap for careful mode: -0.07 AP (95% CI -0.14 to +0.05; positive in 13% of 150 resamples). Small-object AP rises in fast mode (+1.67) but not in careful mode (-0.34), where the mirrored view already recovers many small objects. CarDD photos are mostly close-ups in which damage is already large relative to the 640-px input, and extra tile boxes add false positives and truncated boxes on larger damage; tiling also multiplies compute by five. We therefore do not adopt it; it may still help on wide shots of whole cars, which CarDD does not cover.

6. Discussion and limitations

- **Licence.** CarDD is released for non-commercial research. A commercial service needs either a licence or a model retrained on consented operator photos.
- **Leakage risk.** The base weights were trained by a third party on a re-split of CarDD; we cannot verify that no official test image was used. The ordering `train >> val >` test suggests limited overlap with the official test split, but a clean comparison requires retraining from generic (COCO) weights on the official training split only — the setting of our released GPU notebook.
- **Box vs. mask.** CarDD-paper per-class figures are mask AP; our model predicts boxes only.
- **Single benchmark.** CarDD images are mostly close-ups from the web; on our own 1,000-photo image-level test (whole cars and close-ups, including intact cars) the same system reaches F1 0.626 with a 14.8% false-alarm rate on intact whole cars, which is the more relevant figure for operations.
- **What CPU-only training can and cannot do.** Without retraining the network, post-hoc stages recover at most a few tenths of an AP point. Larger gains require GPU training — e.g. RF-DETR [13] or multi-scale training as in DCN+ — for which we release a ready notebook.

7. Review of 100 papers and 100 derived practices

We queried OpenAlex [24] and arXiv for car-damage detection, assessment and insurance-claim papers, obtaining more than 2,400 candidates, filtered off-topic work with a negative-term list and title penalties, and ranked by topical relevance, citations and recency. For the 100 selected papers we extracted models, datasets, tasks, techniques and reported metrics from abstracts (the 12 seed papers were checked by hand). The most frequent model families are Transfer learning (25), YOLOv8 (15), Mask R-CNN (15), ResNet (11), VGG (8), YOLO (7); tasks are detection (87), severity (46), classification (44) and segmentation (22); the most frequent technique themes are lightweight / mobile deployment (42), severity / cost estimation (34), transfer learning / pre-training (26), instance segmentation masks (20). From these we derived 100 practices in eight groups (data and evaluation, detection models, training, inference and fusion, decision layer, severity and cost, VLM/LLM use,

deployment and operations); 46 were already implemented, 8 were added in v2.1, 27 are planned on CPU and 19 require GPU training.

8. Conclusion

A browser-deployed int8 YOLO11m is on par with the best model in the CarDD paper when evaluated under the same protocol, and flip-TTA with WBF adds a significant +2.3 AP (62.23 AP). Training post-hoc stages on CarDD without a GPU yields only marginal further gains, and we identify memorisation of the training split as a pitfall that silently invalidates stacking on training predictions. The next step is GPU retraining on licence-clean data and an evaluation on operator photos.

Reproducibility

Detections, scripts and results: Hugging Face datasets `sosa123454321/cekicin-training-artifacts` (folders `v2_1`, `v2_2`) and `sosa123454321/cekicin-ai-papers` (`car_damage_100`). Experiment tracking, models and artefacts: Weights & Biases project `elasa2next-sosa-/cekicin-hasar-ai`. Live demo: `sosa123454321-cekicin-site.static.hf.space/ai-lab/model-test.html`.

References

1. X. Wang, W. Li, Z. Wu. CarDD: A New Dataset for Vision-Based Car Damage Detection. *IEEE Transactions on Intelligent Transportation Systems*, 24(7):7202–7214, 2023. <https://arxiv.org/abs/2211.00945>
2. T.-Y. Lin, M. Maire, S. Belongie, et al. Microsoft COCO: Common Objects in Context. *ECCV*, 2014. <https://arxiv.org/abs/1405.0312>
3. R. Solovyev, W. Wang, T. Gabruseva. Weighted Boxes Fusion: Ensembling Boxes from Different Object Detection Models. *Image and Vision Computing*, 107:104117, 2021. <https://arxiv.org/abs/1910.13302>
4. J. Peng, S. Dong, H. Yuan, X. Zheng. Car Damage Detection Based on Multi-View Fusion and Alignment: Dataset and Method. *IEEE Transactions on Intelligent Transportation Systems*, 2025. <https://doi.org/10.1109/tits.2025.3542174>
5. D. Z. Baig, M. Kamal, Z. Ullah. Small Dents, Big Impact: A Dataset and Deep Learning Approach for Vehicle Dent Detection. *arXiv:2508.15431*, 2025. <https://arxiv.org/abs/2508.15431>
6. K. He, G. Gkioxari, P. Dollár, R. Girshick. Mask R-CNN. *ICCV*, 2017. <https://arxiv.org/abs/1703.06870>
7. J. Dai, H. Qi, Y. Xiong, et al. Deformable Convolutional Networks. *ICCV*, 2017. <https://arxiv.org/abs/1703.06211>
8. A. R. Rabbani, G. Kosala. Car Damage Detection using YOLOv11 and RT-DETR. *Int. Seminar on Research of Information Technology and Intelligent Systems (ISRITI)*, IEEE, 2025. <https://doi.org/10.1109/isriti.68345.2025.11393155>
9. M. R. S. Ramazhan, A. Bustamam, R. Anwar. Smart Car Damage Assessment Using Enhanced YOLO Algorithm and Image Processing Techniques. *Information*, 16(3):211, 2025. <https://doi.org/10.3390/info16030211>
10. W. Li, H. Xie, P. Lin, L. Huang, et al. HL-YOLO: Improving Vehicle Damage Detection with Heterogeneous Convolutions and Large-Kernel Attention. *World Electric Vehicle Journal*, 16(12):640, 2025. <https://doi.org/10.3390/wevj16120640>
11. V. Arconzo, G. Gorga, G. Gutierrez, A. Omar, et al. On the Application of DiffusionDet to Automatic Car Damage Detection and Classification via High-Performance Computing. *Electronics*, 14(7):1362, 2025. <https://doi.org/10.3390/electronics14071362>

12. A. Z. Sellam, I. Benaissa, S. E. Bekhouche, et al. C-DiffDet+: Fusing Global Scene Context with Generative Denoising for High-Fidelity Car Damage Detection. arXiv:2509.00578, 2025. <https://arxiv.org/abs/2509.00578>
13. I. Robinson, P. Robicheaux, M. Popov, D. Ramanan, et al. RF-DETR: Neural Architecture Search for Real-Time Detection Transformers. arXiv:2511.09554, 2025. <https://arxiv.org/abs/2511.09554>
14. C. Lin, H. Lyu, X. Xu, J. Luo. INS-MMBench: A Comprehensive Benchmark for Evaluating LVLMS' Performance in Insurance. ICCV, 2025 (arXiv:2406.09105). <https://arxiv.org/abs/2406.09105>
15. V. S. Hogale, A. Pai, N. Ravi, et al. Grounding Agentic VLMs with Dedicated Segmentation for Fine-Grained Vehicle Damage Assessment. arXiv:2608.02470, 2026. <https://arxiv.org/abs/2608.02470>
16. M. J. Hasan, A. Nalwan, K.-L. Ong, H. Jahankhani, et al. GroundingCarDD: Text-Guided Multimodal Phrase Grounding for Car Damage Detection. IEEE Access, 2024. <https://doi.org/10.1109/access.2024.3506563>
17. B. Jiang, R. Luo, J. Mao, T. Xiao, Y. Jiang. Acquisition of Localization Confidence for Accurate Object Detection. ECCV, 2018. <https://arxiv.org/abs/1807.11590>
18. H. Zhang, Y. Wang, F. Dayoub, N. Sünderhauf. VarifocalNet: An IoU-aware Dense Object Detector. CVPR, 2021. <https://arxiv.org/abs/2008.13367>
19. J. Hosang, R. Benenson, B. Schiele. Learning Non-Maximum Suppression. CVPR, 2017. <https://arxiv.org/abs/1705.02950>
20. J. H. Friedman. Greedy Function Approximation: A Gradient Boosting Machine. Annals of Statistics, 29(5):1189–1232, 2001.
21. G. Ke, Q. Meng, T. Finley, et al. LightGBM: A Highly Efficient Gradient Boosting Decision Tree. NeurIPS, 2017.
22. G. Jocher, J. Qiu. Ultralytics YOLO11. Software, 2024. <https://github.com/ultralytics/ultralytics>
23. B. Efron, R. J. Tibshirani. An Introduction to the Bootstrap. Chapman & Hall/CRC, 1994.
24. J. Priem, H. Piwowar, R. Orr. OpenAlex: A Fully-Open Index of Scholarly Works, Authors, Venues, Institutions, and Concepts. arXiv:2205.01833, 2022. <https://arxiv.org/abs/2205.01833>