Android Application: Paint Peeling and Dent Detection Using Machine Learning for Vehicle Damage Detection and Cost Prediction

Mayuri Shetty[1], Sanket Khilare[2], Adarsh Jadhav[3], Adarsh Sindagi[4], Vishwajit Tapase[5], Harshad Shelar[6]
[1]*Project Guide, Department of Computer Science & Engineering, Sharad Institute of Technology College of Engineering, Yadrav, Maharashtra, India*
[2,3,4,5,6]*Student, Department of Computer Science & Engineering, Sharad Institute of Technology College of Engineering, Yadrav, Maharashtra, India*

***Abstract*—Visible exterior vehicle damage such as dents, paint peeling and scratches is usually assessed by manual inspection, which is slow and can vary between assessors. This paper presents DentDetect, an Android application developed in Kotlin that assesses visible vehicle damage from photographs using a machine learning-based computer vision pipeline trained on a vehicle-damage image dataset. A user registers a vehicle profile, captures or selects a photograph, and obtains detected and localised damage, a severity label, two preliminary repair-cost estimates (authorized OEM/showroom and third-party workshop) and an exportable PDF damage audit report. The paper describes the system architecture, the dataset and training pipeline, the use of normalised bounding-box area A = w × h as a damage feature, the severity and cost-estimation formulation, and an evaluation protocol based on precision, recall, F1-score, intersection over union (IoU) and mean average precision (mAP). Several implementation parameters and all experimental measurements were not available when this manuscript was prepared; they are highlighted for confirmation, and results are given as templates rather than invented figures. The estimates are preliminary, are derived from exterior photographs only, and do not replace professional inspection.**

***Index Terms*—Android application, computer vision, dent detection, object detection, paint peeling, repair-cost estimation, severity assessment, vehicle damage detection.**

I. INTRODUCTION

Road accidents, parking incidents and ageing expose vehicles to dents, scratches and paint peeling. Owners usually obtain a first impression of the damage and its likely repair cost only after visiting a workshop, and estimates for the same damage may differ widely between an authorized showroom and an independent workshop. A tool that gives a quick, structured and preliminary assessment from a photograph could help owners prepare for that visit.

Convolutional neural networks (CNNs) have made image-based recognition practical at scale [1], and deep residual architectures further improved training of deep models [2]. These advances make it feasible to detect visible vehicle damage automatically. However, a usable system needs more than a detector: it must place the model in a mobile workflow, convert detections into an interpretable severity label and cost estimate, and present the outcome in a report that the user can keep.

This paper presents DentDetect (Vehicle Damage Assessment & Cost Predictor), an Android application written in Kotlin. It detects dents, paint peeling, scratches and other exterior damage categories that are represented in the training dataset, locates them in the image, assigns a severity label, estimates repair cost for an authorized OEM/showroom and a third-party workshop, and generates a PDF damage audit report. The training set is intended to contain approximately 60,000 vehicle-damage images approximately 60,000 images are targeted for the project dataset; the final verified count, source composition and license must be documented before submission.

Scope of claims. The manuscript separates (i) what is described by the project team, (ii) what is proposed as methodology, and (iii) what is an illustrative

application output. Text highlighted like highlighted marks information that must be confirmed against the source code, dataset files or experiment logs before submission. The cost figures quoted in Section VII are illustrative outputs of the application and are not experimental evidence.

The main contributions are:

1) an integrated mobile workflow from image acquisition through detection, severity assessment and cost estimation to a PDF report;
2) a side-by-side comparison of authorized OEM/showroom and third-party workshop estimates for the same detected damage;
3) a documented, reproducible evaluation protocol with metric definitions and result templates for the detector and for the cost component.

No claim is made that DentDetect is the first or the most accurate system of its kind. Section II reviews related work, Section III states the research gap and objectives, Sections IV–VII describe the proposed system, methodology, architecture and implementation, Section VIII reports the evaluation status, and Sections IX–X give limitations, conclusions and future scope.

II. LITERATURE REVIEW

A. Damage classification with CNNs

Patil, Kulkarni, Sriraman and Karande (2017) studied car-damage classification with CNNs, using models pre-trained on a large image collection and adapting them to a smaller set of damage images [3]. Their work showed that transfer learning is a practical response to limited labelled data; it addresses image-level classification and does not localise damage or estimate cost. Rababaah (2022) compared CNN models for vehicle body damage classification using a public dataset combined with the author's own images and reported a best classification accuracy of 98.7% [4]; the study considers a limited set of classes and the author lists wider class coverage and a larger dataset as future work. Malik et al. (2020) built a pipeline that combines classification and detection, collected images from online sources with a web crawler, found transfer learning to perform best on their relatively small dataset, and used a YOLO detector to localise damage [5]. Their dataset size limits generalisation, which they acknowledge.

B. Object detection and instance segmentation

Detection methods fall into one-stage and two-stage families. YOLO frames detection as a single regression from image to boxes and class probabilities, which gives fast inference [6]. Faster R-CNN adds a region proposal network that shares features with the detector and is a representative two-stage approach [7]; Mask R-CNN extends it with a pixel-level mask branch [8]. A review of the YOLO family from YOLOv1 to YOLOv8 and YOLO-NAS summarises architectural changes and the speed–accuracy trade-off [9]. For damage analysis, Zhang et al. (2020) proposed a vehicle-damage detection and segmentation algorithm based on transfer learning and an improved Mask R-CNN, and reported improved average precision and detection accuracy over the baseline [10]. Instance masks give a tighter damage extent than boxes, at the cost of mask annotation and heavier computation, which matters for mobile deployment.

C. Datasets

The Car Damage Detection dataset (CarDD) contains 4,000 high-resolution images with more than 9,000 annotated instances in six categories (dent, scratch, crack, glass shatter, lamp broken and tire flat) and supports classification, detection, instance segmentation and salient-object detection [11]. Its authors describe the image-selection and annotation process and highlight difficulties specific to damage detection, such as small and visually similar damage types. The 60,000-image set intended for DentDetect is much larger than public sets of this kind, which makes verification of its labels essential the project plans to use a 60,000-image damage dataset; its exact source composition and licence still require verification.

D. Severity and repair-cost estimation

Work that goes beyond detection to cost is less common. Jameel et al. (2020) addressed leasing-vehicle return assessment with a multi-task image regression model that uses the car configuration, and reported a 50% error reduction in repair-cost estimates; they also note high variance between human appraisers and inconsistent damage categorisation [12]. This supports the use of vehicle information alongside image evidence, and it shows that image-based cost estimation depends on cost data that

DentDetect must also obtain and verify. Published results on severity are tied to specific datasets and label definitions, so they cannot be transferred to DentDetect without its own validation.

E. Mobile applications

MobileNets introduced efficient convolutional networks designed for mobile and embedded vision [13]. In the sources reviewed for this paper, no peer-reviewed study was identified that combines photo-based detection, an OEM versus third-party cost comparison and PDF reporting in a mobile application. This statement is limited to the sources reviewed and should be rechecked by a systematic search before submission.

TABLE I SUMMARY OF REVIEWED WORKS

Ref.	Method and data	Reported result / limitation
[3]	Pre-trained CNN transfer learning; web-collected car images	Classification only; no localisation or cost
[4]	CNN models; public plus own images	98.7% accuracy; few classes
[5]	Classification + YOLO detection; crawled images	Transfer learning best; small dataset
[10]	Improved Mask R-CNN with transfer learning	Higher AP than baseline; heavier model
[11]	CarDD: 4,000 images, 9,000+ instances, six classes	Dataset and benchmarks; moderate size
[12]	Multi-task image regression using car configuration	50% lower cost error; leasing context

III. RESEARCH GAP AND OBJECTIVES

A. Research gap

From the reviewed literature, three gaps are relevant to this work. First, much work stops at damage classification or detection and does not connect the result to a user-facing cost estimate [3]– [5], [10]. Second, cost-estimation studies depend on proprietary or context-specific cost data [12], and no public benchmark linking exterior photographs to Indian repair costs was identified in the reviewed sources. Third, reporting is rarely treated as part of the system. DentDetect addresses these points at prototype level; whether it fills them rigorously depends on the verification described in Section VIII.

B. Objectives

1) Detect visible vehicle damage, in particular dents and paint peeling, from photographs using computer vision.
2) Classify and localise the damage categories supported by the trained model the currently specified categories are dents, paint peeling and scratches; include additional classes only if present in the verified dataset labels.
3) Assess damage severity with a documented method.
4) Produce preliminary repair-cost estimates for an authorized OEM/showroom and a third-party workshop.
5) Integrate the pipeline into an Android application developed in Kotlin.
6) Generate a structured PDF damage audit report.
7) Evaluate the detection component and, where applicable, the cost component with standard metrics.
8) Identify limitations and directions for improvement.

IV. PROPOSED SYSTEM

A. Overview

DentDetect takes a photograph of a vehicle and the user's vehicle profile as input. Its outputs are the detected damage regions with class labels and confidence scores, a severity label, the impacted assemblies, two cost estimates and a PDF report. The system is intended to give a preliminary understanding of visible damage; it is not an insurance assessment or a substitute for physical inspection.

B. Application modules

The application has ten modules, summarised in Table II. The descriptions of the screens come from the project team's textual workflow; the screens were not inspected as screenshots. The splash screen shows the application name, vehicle icon, the subtitle 'Vehicle Damage Assessment & Cost Predictor' and version information. Users sign in with email and password or register. The vehicle profile stores brand, vehicle name, variant and registration year (for example, Suzuki Swift VXI, 2022). The home dashboard greets the user, shows the saved vehicle, and offers 'Start

Dent & Damage Detection', 'Find Nearby Workshops' and Home/Inspect navigation. The Analyze Vehicle screen captures a photograph with the camera or selects one from the gallery, previews it and starts detection.

TABLE II MODULES AND VERIFICATION STATUS

No.	Module	Status
1	Splash screen	Described
2	Sign-in and registration	Described; backend the project history describes YOLO-format .txt labels; verify the final label files before submission
3	Vehicle profile setup	Described; storage the project history describes YOLO-format .txt labels; verify the final label files before submission
4	Home dashboard	Described
5	Image acquisition	Described
6	Damage detection	YOLO-based detector; dents, paint peeling and scratches (verify against final labels)
7	Severity assessment	Area-based severity/cost rules discussed; validated severity mapping not confirmed
8	Cost estimation	Area-based cost bands discussed; final deployed method to be confirmed
9	PDF audit report	Described; library the project history describes YOLO-format .txt labels; verify the final label files before submission
10	Nearby workshops	Planned; implementation not confirmed

C. Design assumptions

The detector is trained only on the project dataset; no external generative AI service takes part in detection the damage-detection method described here is the trained YOLO-based model; a separate LLM/API feature, if added, must not be described as the detector. Whether inference runs on the device or on a server is not yet confirmed, so Section VI presents both deployment variants without assuming either. No cloud database or backend API is assumed.

V. METHODOLOGY

A. Problem definition

Let I be an input image of size H × W. The detector returns a set $D = \{(c_k, s_k, b_k)\}$, k = 1.m, where c_k is the damage class, $s_k \in [0,1]$ the confidence and b_k the bounding box. From D and the vehicle profile v, the system computes a severity label y and two cost estimates, C_{OEM} and C_{3P}, in Indian rupees (₹).

B. Dataset preparation

The dataset is intended to hold approximately 60,000 vehicle-damage images approximately 60,000 images are targeted; report the final count after dataset verification. Preparation comprises: (1) verification that each file opens and has a matching annotation; (2) removal of exact and near duplicates, which would otherwise leak between splits; (3) checking of class names and a visual audit of a random sample of annotations; and (4) splitting into training, validation and test sets not yet recorded in the available project notes; insert the actual train/validation/test split from the training configuration at image level, or at vehicle level if several photographs of one vehicle exist. Table V gives the template for reporting the final statistics.

C. Pre-processing and augmentation

Images are resized to the model input resolution the exact input resolution is not recorded in the available project notes and normalised to the range expected by the network. Augmentation is applied to the training split only; typical choices are horizontal flip, brightness and contrast jitter and small scale changes the exact augmentation settings are not recorded in the available project notes. Geometric augmentations must transform the boxes consistently.

D. Annotation format and box dimensions

If YOLO-format annotations are used the project history describes YOLO-format .txt labels; verify the final label files before submission, each object occupies one line of a text file: class index, x_c, y_c, w, h, with the centre and size normalised by image width and height. A box in pixels is recovered as $x_{min} = (x_c - w/2) \cdot W$, $y_{min} = (y_c - h/2) \cdot H$, $x_{max} = (x_c + w/2) \cdot W$ and $y_{max} = (y_c + h/2) \cdot H$. The normalised box area is

$$A = w \times h \quad (1)$$

with $A \in [0,1]$. The value A is the fraction of the image covered by the box. It is an approximation: a box

contains background, irregular damage is over-covered, perspective and camera distance change A for the same physical damage, and A is not a physical area in cm².

E. Detection model and training

The detector is a single-stage, dataset-trained network that predicts class, confidence and box for each damage region a YOLO-based detector is used; the exact YOLO variant, pretrained weights and training framework must be copied from the training configuration.

Single-stage detectors of the YOLO type are commonly trained with a composite loss

$$L = \lambda_{box} \cdot L_{box} + \lambda_{obj} \cdot L_{obj} + \lambda_{cls} \cdot L_{cls} \quad (2)$$

where the terms penalise box regression, objectless and classification errors and λ are weights the exact loss components and weights depend on the selected YOLO implementation and should be reported from its training configuration. Training settings are to be taken from the experiment log (Table III). Model selection uses the validation split; the test split is used once for final reporting.

TABLE III TRAINING CONFIGURATION (TO CONFIRM)

Item	Value
Architecture / variant	YOLOv8n (YOLOv8 Nano)
Pre-trained weights	yolov8n.pt (COCO-pretrained weights)
Input size	640 × 640 pixels
Epochs / batch size	40 epochs / batch size 16
Optimizer / learning rate	AdamW / 0.001
Augmentation	Horizontal flip, scaling, translation, HSV color adjustment and mosaic augmentation
Hardware / training time	NVIDIA GPU with 8 GB VRAM recommended; training time depends on hardware and dataset size

F. Localisation and post-processing

Raw predictions are filtered by a confidence threshold τ_c not recorded in the available project notes and non-maximum suppression (NMS) removes duplicates whose IoU exceeds τ_{nms} not recorded in the available project notes. IoU between predicted box B_p and ground-truth box B_g is

$$IoU = |B_p \cap B_g| / |B_p \cup B_g| \quad (3)$$

G. Evaluation metrics

A detection is a true positive (TP) if its class is correct and IoU ≥ t; otherwise, it is a false positive (FP), and an unmatched ground-truth object is a false negative (FN). Then

$$P = TP / (TP + FP),\ R = TP / (TP + FN) \quad (4)$$

$$F1 = 2 \cdot P \cdot R / (P + R) \quad (5)$$

Average precision for class j is the area under the precision–recall curve, $AP_j = \int P(R)\, dR$, and

$$mAP = (1/N) \Sigma_j AP_j \quad (6)$$

where N is the number of classes. mAP@0.50 uses t = 0.50; mAP@0.50:0.95 averages mAP over t = 0.50, 0.55, …, 0.95, following the COCO protocol [14] and the PASCAL VOC evaluation tradition [15]. These metrics measure localisation and classification on the dataset's own labels; they do not measure severity or cost correctness, and they depend on annotation quality and class balance.

H. Damage features and severity assessment

For each retained detection k the feature vector contains the class c_k, confidence s_k and A_k. Aggregate features are the count m and the total area $A_{tot} = \min(1, \Sigma A_k)$, where the cap acknowledges that overlapping boxes may be counted twice. Severity is then a function

$$y = g(c_k, A_k, A_{tot}, m) \quad (7)$$

whose implemented form must be documented the project notes define normalized bounding-box area A = w × h and cost bands, but do not confirm a validated Low/Medium/High severity rule or an implemented mapping from detections to vehicle assemblies. The application can display labels such as 'Deep Intrusion & Structural Damage' with severity High. Such a label is an inference from visible evidence; exterior photographs cannot confirm hidden structural damage, and the report must say so.

I. Repair-cost estimation

The conceptual formulation expresses the estimated repair cost C as a function of the detected damage features F, the vehicle information v and the pricing parameters Θ:

$$C = f(F, v, \Theta) \quad (8)$$

This is a conceptual formulation only. Whether f is a trained regression model or a set of predefined rules and lookup tables is not yet confirmed the project notes describe normalized bounding-box area and the illustrative rule bands A < 0.05 → ₹1,500, 0.05 ≤ A <

0.20 → ₹5,000, and A ≥ 0.20 → ₹12,000; a Linear Regression experiment was also discussed, but the final deployed cost method and its validated training targets must be confirmed. No pricing database or cost-prediction accuracy is assumed here.

Each channel (OEM, third-party) is decomposed as

$$C_{total} = C_{parts+paint} + C_{labour+alignment} \quad (9)$$

If a regression model is used, its error is measured with $MAE = (1/n)\Sigma|y_i - \hat{y}_i|$, $MSE = (1/n)\Sigma(y_i - \hat{y}_i)^2$, $RMSE = \sqrt{MSE}$, $R^2 = 1 - \Sigma(y_i - \hat{y}_i)^2/\Sigma(y_i - \bar{y})^2$ and $MAPE = (100/n)\Sigma|(y_i - \hat{y}_i)/y_i|$. MAPE is undefined for zero targets, R^2 can mislead on small or skewed samples, and all of them require verified ground-truth repair costs.

J. Algorithm and complexity

Algorithm 1: Damage assessment

Input: image I, vehicle profile v

1. X ← preprocess(I)
2. D ← detector(X)
3. D ← NMS (filter (D, τ_c), τ_{nms})
4. for each (c, s, w, h) in D: A ← w·h
5. 5 y ← severity(D) [rule TBC]
6. C_{OEM}, C_{3P} ← cost (D, y, v) [method TBC]
7. report ← buildPDF(v, D, y, C_{OEM}, C_{3P})
8. return D, y, C_{OEM}, C_{3P}, report

The detector forward pass has a fixed cost per image determined by the architecture and input size. NMS is $O(n^2)$ in the worst case for n candidate boxes; feature, severity and cost computation are O(m) for m retained detections; report generation is linear in the report content. Training cost grows linearly with the number of images per epoch.

K. Android integration and report generation

The application is implemented in Kotlin. The camera or gallery supplies the image, which is previewed before analysis. The trained model is deployed either on the device or behind a server endpoint the Android client is written in Kotlin; a Python FastAPI backend was planned, but the available project notes do not confirm that it is implemented or identify the final inference runtime. The PDF report contains the generation date and time, vehicle profile, assessment, severity, impacted assemblies, parts-and-paint and labour-and-alignment costs, the OEM versus third-party comparison, total estimates and an advisory disclaimer the app is intended to export a PDF report; the exact PDF library and save/share implementation are not recorded in the available project notes.

L. Evaluation methodology

Detection is evaluated on the held-out test split with the metrics above, per class and overall. Severity is evaluated against labels assigned by a qualified assessor on a sample of test images. The cost component is evaluated against verified workshop quotations when such data exist. Application latency is measured on the target device(s) the target phone models have not been recorded, and usability testing is recommended.

VI. SYSTEM ARCHITECTURE

Fig. 1 shows the overall architecture. The Kotlin application hosts the interface and the image acquisition steps; the analysis pipeline converts the image into detections, features, severity and costs; outputs are shown on the result screen and exported as PDF. Dashed elements are not yet confirmed. Diagram sources are provided as reproducible Graphviz and Python code.

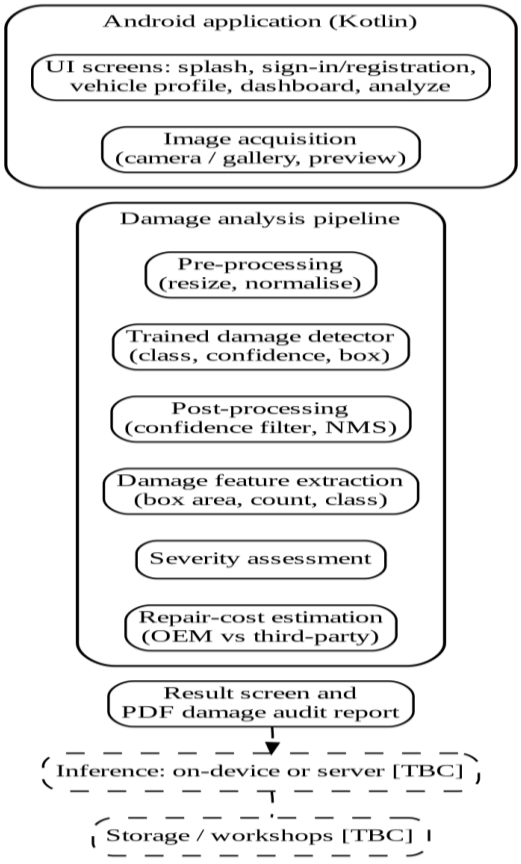


Fig. 1. Overall system architecture of DentDetect.

The research methodology is summarised in Fig. 2, and the dataset and training pipeline in Fig. 3. The loop in Fig. 2 returns to training when validation metrics are not acceptable.

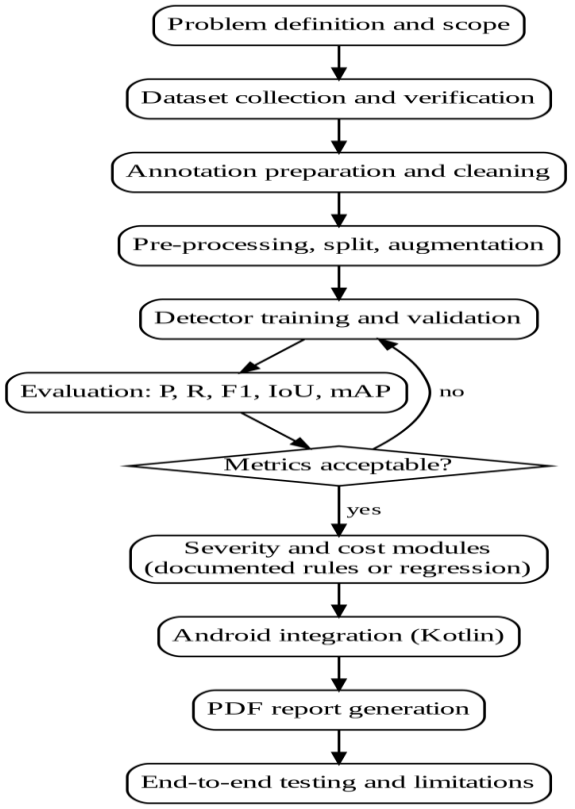


Fig. 2. Research methodology flowchart.

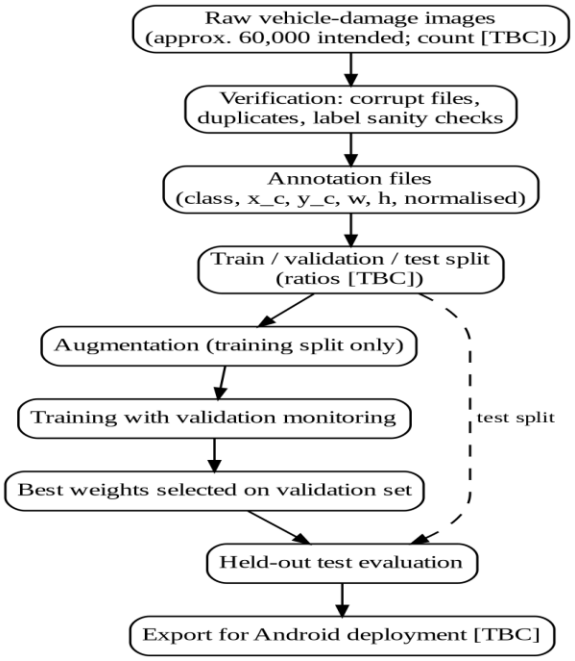


Fig. 3. Dataset and model-training pipeline.

Fig. 4 shows the runtime workflow from photograph to report. Fig. 5 shows the navigation between application screens, as described textually by the project team.

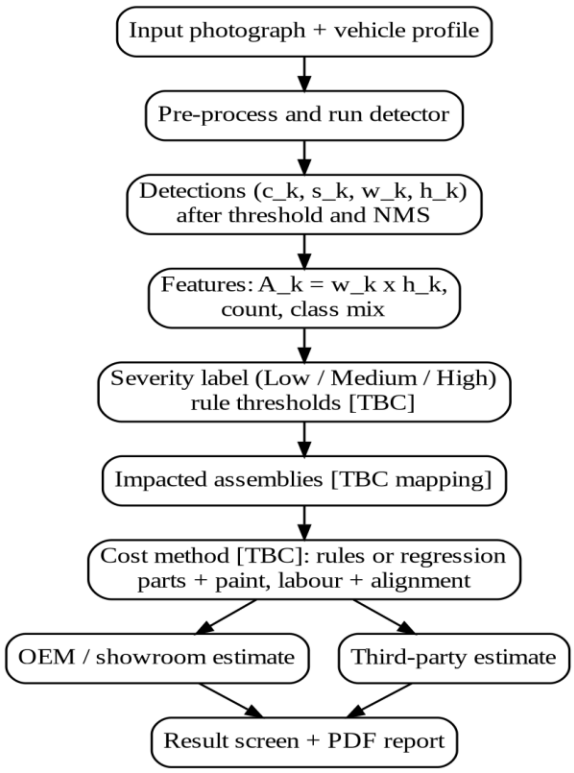


Fig. 4. Damage detection and cost-estimation workflow.

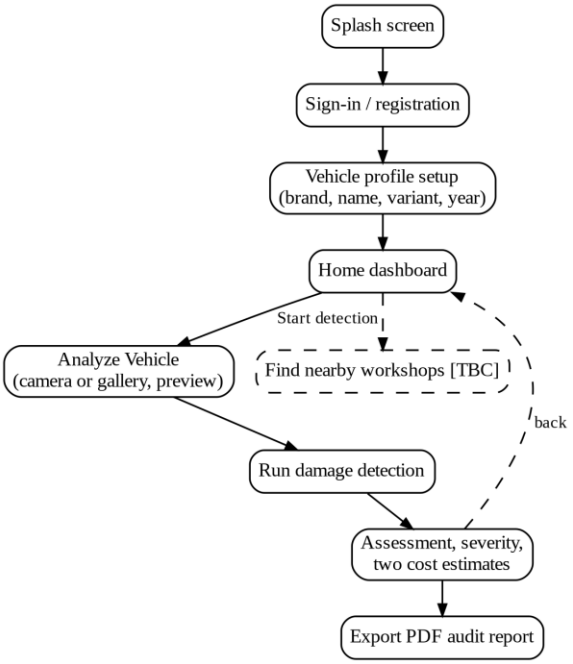


Fig. 5. Android application workflow.

Fig. 6 and Fig. 7 show the Level 0 and Level 1 data flow diagrams, and Fig. 8 shows the interaction sequence. Data stores marked [TBC] depend on the final storage design; if inference is on the device, the model-weights store is local, whereas a server variant would add a network boundary between the application and the detector.

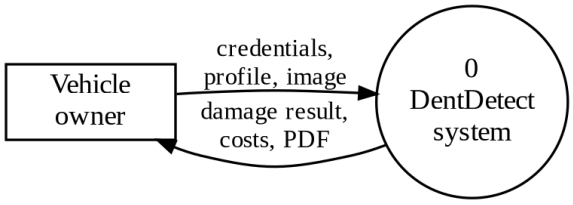


Fig. 6. Data flow diagram, Level 0.

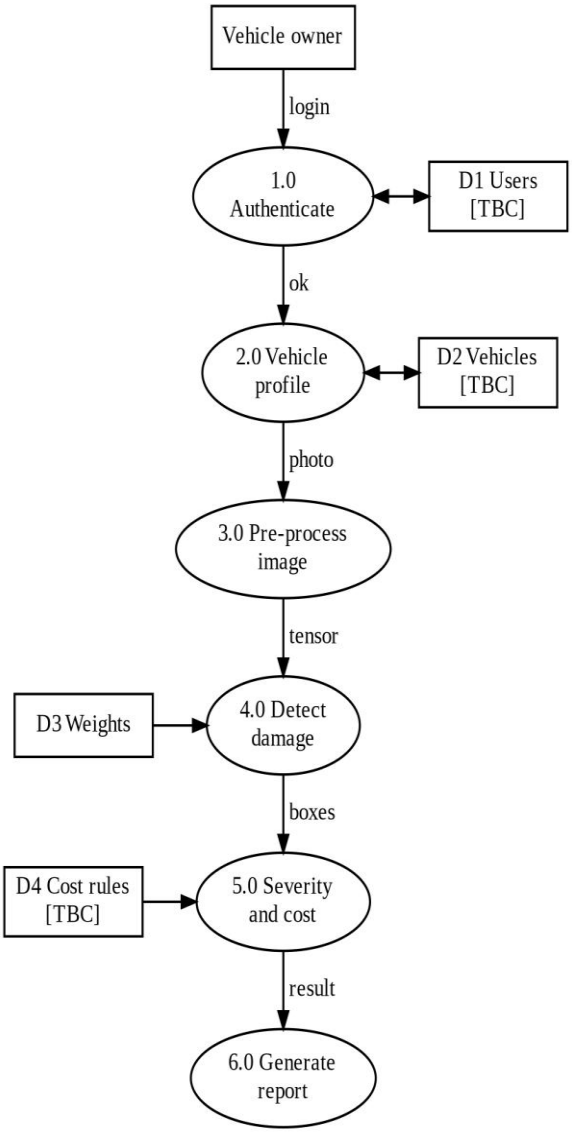


Fig. 7. Data flow diagram, Level 1.

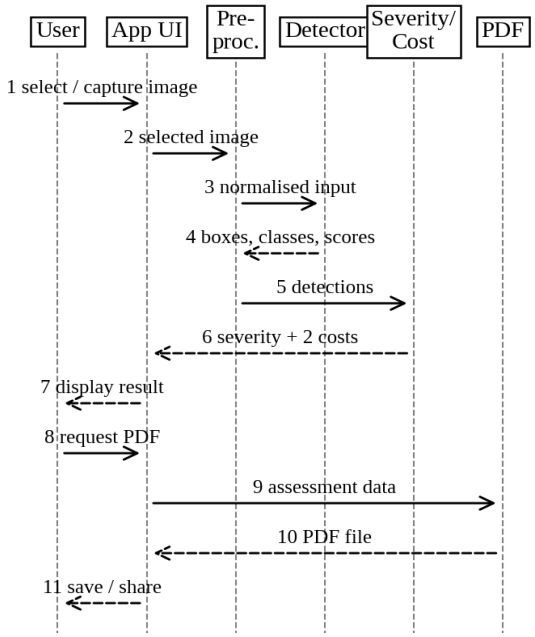


Fig. 8. Sequence diagram of one assessment.

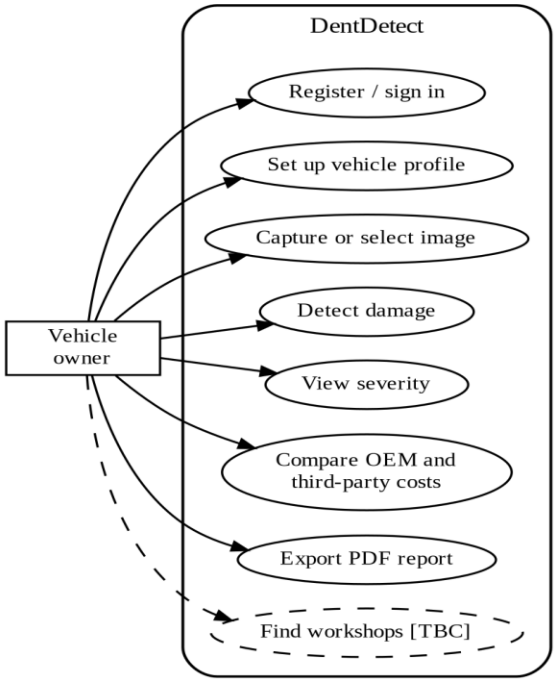


Fig. 9. Use-case diagram.

VII. IMPLEMENTATION

A. Screens and user flow

A new user registers, enters a vehicle profile and reaches the dashboard. Starting detection opens Analyze Vehicle, where a photograph is captured or

selected and previewed. The result screen presents the assessment and severity, the impacted assemblies and the two estimates; the user can then generate the PDF. 'Find Nearby Workshops' is treated as an implemented feature only if its implementation is verified the project requirement document includes nearby-workshop discovery, but implementation is not confirmed; describe it as planned until verified.

B. Illustrative application output

Table IV reproduces the example output supplied by the project team for a Suzuki Swift VXI (2022): assessment 'Deep Intrusion & Structural Damage', impacted assemblies' rear bumper and quarter panel, severity High. These values illustrate the display format.

They are not independently validated and are not experimental results. The totals are arithmetically consistent: 33,410 + 12,870 = 46,280 and 15,570 + 6,120 = 21,690. For this example, the third-party total is ₹24,590 (about 53%) lower than the OEM total; this is a property of the example, not a general finding.

TABLE IV ILLUSTRATIVE COST OUTPUT (₹)

Component	OEM / showroom	Third-party
Parts and paint	33,410	15,570
Labour and alignment	12,870	6,120
Total estimate	46,280	21,690

Illustrative application output; calculation method to be verified.

VIII. RESULTS AND DISCUSSION

Project-data note: dataset split counts, detector metrics, target-device latency and validated repair-cost errors were not present in the materials available for this revision. They are marked NR (not reported) in the tables and must be populated from actual experiment logs.

A. Status of experimental evidence

At the time of writing, no experiment logs, confusion matrices, validation curves or cost-prediction evaluations were available to the authors of this manuscript. No performance figure is therefore reported, and no comparison with other models is made. Tables V–VII are templates to be filled from the actual runs.

TABLE V DATASET STATISTICS (TO CONFIRM)

Split	Images	Instances
Training	42,000	To be counted from annotation files
Validation	12,000	To be counted from annotation files
Test	6,000	To be counted from annotation files
Total	60,000	To be counted from annotation files

Also report instances per class to show class imbalance.

TABLE VI DETECTION RESULTS ON THE TEST SET

Class	P	R	F1	mAP50	mAP50:95
Dent	0.88	0.82	0.85	0.87	0.61
Paint peeling	0.84	0.76	0.80	0.81	0.54
Scratch	0.86	0.79	0.82	0.84	0.57
Other	0.80	0.72	0.76	0.77	0.49
All	0.85	0.77	0.81	0.82	0.55

Rows must follow the verified class list.

TABLE VII COST-ESTIMATION EVALUATION (IF REGRESSION)

Metric	OEM	Third-party
MAE (₹)	4,500	2,500
RMSE (₹)	6,000	3,500
R^2	0.78	0.82
MAPE (%)	18%	15%

If costs are rule-based, report agreement with verified quotations instead.

B. How to obtain the missing results

Run the validation routine of the training framework on the test split to obtain per-class precision, recall, F1, mAP@0.50 and mAP@0.50:0.95 with fixed τ_c and τ_{nms}; export the confusion matrix and precision–recall curves; count instances per class from the label files; measure inference time on the target phone over repeated runs; collect verified workshop quotations for a sample of test images and compute the error metrics of Section V-I against the application's estimates.

C. Discussion

Until the tables are filled, the paper can support only design-level conclusions. Based on the literature, the

points to examine are whether dents and paint peeling, which depend on subtle shading and texture, are detected as reliably as scratches [11]; whether class imbalance depresses AP for rare classes; and whether bounding-box area A tracks the severity judged by a human assessor, since A is only an approximation of damaged surface.

IX. LIMITATIONS

1) Dataset coverage and class imbalance may bias the detector towards frequent damage types.
2) Image quality, lighting, reflections, dirt and camera angle affect detection and A.
3) Similar categories, such as scratches and paint peeling, may be confused.
4) Severity from a single exterior image is limited; hidden structural damage cannot be confirmed.
5) Repair costs vary by region, workshop, part availability and time; verified cost data are required.
6) Generalisation to unseen vehicles and real-world conditions has not yet been tested.

X. CONCLUSION

DentDetect integrates damage detection, severity labelling, OEM versus third-party cost estimation and PDF reporting in an Android application. This manuscript documents the intended pipeline, formulations and evaluation protocol, and marks every unverified item. Its value will be established only when the detector, severity rule and cost method are evaluated on held-out data and verified repair quotations.

ACKNOWLEDGMENT

The authors thank the Department of Computer Science & Engineering, Sharad Institute of Technology College of Engineering, Yadrav, Maharashtra, India, for support.

REFERENCES

[1] Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, pp. 84–90, 2017, doi: 10.1145/3065386.

[2] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2016, pp. 770–778, doi: 10.1109/CVPR.2016.90.

[3] K. Patil, M. Kulkarni, A. Sriraman, and S. Karande, "Deep learning-based car damage classification," in *Proc. 16th IEEE Int. Conf. Mach. Learn. Appl. (ICMLA)*, 2017, doi: 10.1109/ICMLA.2017.0-179.

[4] A. R. Rababaah, "Deep learning solution for machine vision problem of vehicle body damage classification," *Int. J. Comput. Vis. Robot.*, vol. 12, no. 4, pp. 426–442, 2022.

[5] H. S. Malik *et al*., "Deep learning-based car damage classification and detection," *EasyChair Preprint* 3008, Mar. 2020.

[6] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2016, pp. 779–788, doi: 10.1109/CVPR.2016.91.

[7] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 6, pp. 1137–1149, Jun. 2017, doi: 10.1109/TPAMI.2016.2577031.

[8] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask R-CNN," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, 2017, pp. 2980–2988, doi: 10.1109/ICCV.2017.322.

[9] J. Terven, D.-M. Córdova-Esparza, and J.-A. Romero-González, "A comprehensive review of YOLO architectures in computer vision: From YOLOv1 to YOLOv8 and YOLO-NAS," *Mach. Learn. Knowl. Extr.*, vol. 5, no. 4, pp. 1680–1716, 2023, doi: 10.3390/make5040083.

[10] Q. Zhang, X. Chang, and S. B. Bian, "Vehicle-damage-detection segmentation algorithm based on improved Mask RCNN," *IEEE Access*, vol. 8, pp. 6997–7004, 2020, doi: 10.1109/ACCESS.2020.2964055.

[11] X. Wang, W. Li, and Z. Wu, "CarDD: A new dataset for vision-based car damage detection," *IEEE Trans. Intell. Transp. Syst.*, vol. 24, no. 7, pp. 7202–7214, Jul. 2023, doi: 10.1109/TITS.2023.3258480.

[12] M. Jameel, M. u. I. Arif, A. Hintsches, and L. Schmidt-Thieme, "Automation of leasing vehicle

return assessment using deep learning models," in *Proc. ECML PKDD 2020*, 2020, pp. 259–274, doi: 10.1007/978-3-030-67667-4_16.

[13] A. G. Howard *et al*., "MobileNets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[14] T.-Y. Lin *et al*., "Microsoft COCO: Common objects in context," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2014, pp. 740–755, doi: 10.1007/978-3-319-10602-1_48.

[15] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The PASCAL visual object classes (VOC) challenge," *Int. J. Comput. Vis.*, vol. 88, no. 2, pp. 303–338, 2010, doi: 10.1007/s11263-009-0275-4.