International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 10 Issue: 11 | Nov - 2026 SJIF Rating: 8.659 ISSN: 2582-3930

VisionCare: Portable Smart Retinal Screening System for Early Eye Disease Detection Using Edge AI and IOT

Manjunath Badiger[1], Aditya Javoor[2], Shriniketan Kulkarni[3], Lohit Talwar[4], Prof. Rajat Acharya [5]

[1234] Students, B.E. Computer Science and Engineering , KLS Vishwanathrao Deshpande Institute of Technology, Haliyal

[5] Guide, Department of Computer Science and Engineering, KLS Vishwanathrao Deshpande Institute of Technology, Haliyal

Affiliated to Visvesvaraya Technological University, Belagavi, Karnataka, India

[1] 2vd24cs404@klsvdit.edu.in | [2] 2vd24cs400@klsvdit.edu.in

[3] 2vd23cs067@klsvdit.edu.in | [4] 2vd24cs402@klsvdit.edu.in

[5] rra@klsvdit.edu.in

***Abstract*—VisionCare is a connected prototype for preliminary retinal screening with deep learning. Retinal disease can end in irreversible blindness, yet cost and distance keep many people from reaching a specialist. The prototype provides first-level screening and lets patients read their own reports. It combines a React/Vite interface for healthcare workers, a FastAPI backend, OpenCV and Pillow for preprocessing, a PyTorch Vision Transformer (ViT-Base Patch16-224), PostgreSQL for records, ReportLab for PDF generation and a Flutter app for patients. A healthcare worker uploads a retinal image; the system scales it to 224 × 224 pixels, applies CLAHE contrast enhancement and a circular mask, and normalises the pixel values, after which the Vision Transformer assigns one of five classes: Normal, Diabetic Retinopathy, Glaucoma, Cataract or Age-related Macular Degeneration (AMD). The paper describes the architecture, algorithms, interfaces and an evaluation plan. Formal testing is incomplete and the proposed Raspberry Pi imaging unit has not been built into the system, so no accuracy, latency or clinical-validity figures are reported. The output is a first screening aid only and cannot stand in for a doctor's diagnosis.**

Keywords—Retinal screening, Edge AI, Internet of Things, Vision Transformer, Image processing, Mobile health

I. INTRODUCTION

Clear vision supports independence, education and livelihood. Many eye conditions progress gradually, so patients frequently consult a doctor only once their sight has visibly changed. Remote and low-income regions have few specialist imaging devices and few trained ophthalmologists. A connected screening system can gather images and follow-up details in a single place, although a trained professional must still review every result.

Prior studies report that deep learning can find retinal disease in fundus photographs [1], and broader reviews survey artificial intelligence in ophthalmology [2]. Edge computing, which performs processing near the point where data is collected, is one motivation for portable designs [3]. These works supply background only; none of them measures VisionCare, which relies on its own data, model checkpoint and validation set.

VisionCare joins a web workflow for healthcare workers, a backend that preprocesses and classifies images, patient-linked records, PDF reports and a mobile report viewer. Classification is done by a Vision Transformer (ViT) [6]. A portable Raspberry Pi acquisition unit belongs to the design but is only proposed; it has been neither assembled nor validated, and the project report states no model accuracy or clinical evaluation. The paper therefore treats the software route, in which a healthcare worker uploads an image, separately from the hardware route.

The paper contributes four things: a full software architecture, a step-by-step specification from image upload to report, block diagrams and flowcharts, and a plan for testing the system with adequate evidence. The interface screens and the configured checkpoint merely show what exists so far. They reveal nothing about classification quality, system reliability or patient benefit, so we keep the evidence from software upload, camera capture, model evaluation and clinical validation apart.

The rest of the paper is organised as follows. Section II presents the materials, procedures, algorithms, architecture and evaluation methods. Section III states the problem and objectives. Section IV explains the methodology, including feasibility, testing and release review. Section V reports the present evidence and limitations, and Section VI concludes with the next steps.

II. METHODS

A. Materials Used

VisionCare is assembled from the following software components. Healthcare workers use an interface built with React [10] and Vite. The backend is written in Python with FastAPI [11], runs on Uvicorn and exposes REST services [7]. JSON Web Tokens (JWT) authenticate requests [8]. OpenCV [12] and Pillow prepare the images, while PyTorch [9] loads the ViT checkpoint and performs inference. PostgreSQL holds user, patient, scan, result and report metadata, and ReportLab produces the downloadable PDFs. Patients use an app developed with Flutter [13] and Dart. Together these form a modular chain from image upload to

report access, with the API as the only gateway to the data and the model. Table I lists the role of each module.

I. WHAT EACH SOFTWARE MODULE DOES

Module	Input	Responsibility / output
Web client	Worker session, patient, image	Select patient, submit scan, display result
FastAPI service	REST request and JWT	Validate, authorise and call processing modules
Preprocessing	Validated image	Model-ready normalised image
ViT inference	Prepared image and checkpoint	Five class scores and predicted category
PostgreSQL	Patient, scan and result metadata	Persistent records and retrieval
Report / mobile	Scan result and patient session	PDF generation and patient report access

The classifier is a ViT-Base Patch16-224 checkpoint with five output classes. Its name does not reveal the data source, class balance, test-set separation or diagnostic performance, and the project report documents none of these. We therefore cannot give a dataset size, split or held-out metric.

The planned imaging unit would consist of a Raspberry Pi controller, a camera, a lens assembly, controlled illumination, a power supply or battery, a network interface, local storage and a mount or enclosure, which together would capture an image and pass it to the software. Table II gives each part, its intended role and its status to date.

II. RASPBERRY PI IMAGING UNIT – PROPOSED PARTS AND CURRENT STATUS

Part	Intended role	Current status
Raspberry Pi	Runs the camera and sends images	Proposed, not assembled
Camera	Takes the eye/fundus image	Image quality not checked
Lens assembly	Focuses the retinal image on the sensor	Alignment still to be done
Illumination	Lights the eye during capture	Safety and brightness not checked
Power, enclosure, Wi-Fi	Battery or supply, housing, image transfer	Described only

During development the web client runs in a browser through Vite, which can proxy requests to the API served by Uvicorn. The clients, backend, database and proposed device all require a stable network connection. Before deployment, service addresses, credentials, allowed origins, HTTPS, upload limits and file permissions must be configured. A local run or a successful demonstration does not show that the system will stay available, scale or remain secure in production. Likewise, using a well-documented framework does not validate the application or the model; validity depends on the tests described here.

B. Key Procedures and Techniques

1) Request handling and validation. A healthcare worker signs in, selects a patient and uploads an image. Upload and analysis are separate API calls: POST /api/scans/upload receives a patient identifier and an image file, and POST /api/scans/{scan_id}/analyze starts analysis of a stored scan. The project report gives no further detail on the responses, and these endpoints have not yet been formally tested.

A request is accepted only when the caller is authorised, the patient link is valid and the file meets the allowed type and size. The report names these error cases: a missing file, a malformed image, an unknown scan ID, an inaccessible patient record, an unavailable database and missing model weights. In each case the backend should return a controlled error rather than a false success.

2) Image preprocessing. Table III lists the input steps. Resizing gives the fixed 224 × 224 shape that ViT-Base Patch16-224 expects. CLAHE improves local contrast without over-amplifying noise, the circular mask hides the border outside the fundus area, and normalisation rescales pixel values to match the model's training. The steps must run in the same order and with the same parameters as during training. The exact values still need to be confirmed from the code before any evaluation can be repeated.

III. IMAGE PREPARATION STEPS BEFORE THE MODEL

Stage	Operation	Purpose / condition
Decode and validate	Open a supported image; reject a malformed file	Keep invalid input away from the model
Resize	224 × 224 pixels	Match the ViT input shape
Contrast	CLAHE	Improve local contrast; parameters still to be confirmed
Mask	Circular fundus-field mask	Remove the irrelevant border area
Normalise	Training-time pixel convention	Keep the inference input the same as in training

3) Data lifecycle and result persistence. Each scan receives its own ID, which ties together the uploaded image, the preprocessing request, the model output and the PDF report, while the scan status tracks progress (Table IV). After inference we store the metadata, predicted category, class scores, model details and report reference against the patient record, and ReportLab then writes the PDF. The PDF carries the scan reference and date, the image, the preliminary class, the scores, the model details and a note that a professional must interpret the result.

IV. STAGES A SCAN GOES THROUGH

Scan status	Condition / transition
Uploaded	Image and patient link accepted
Queued	Valid scan ID created
Processing	Decoding and model inference started
Completed	Result and report references saved
Failed	Error returned; no false success state

Failures can occur during inference or saving, and the system must never report success when they do. If the database is down, the result is not displayed as saved. If model weights are missing, no stale or invented prediction is returned. A retry reuses the same scan ID, so it should not create a second analysis or report. The project report lists these as expected error handling, but no recorded test of them exists yet.

4) Communicating the screening result. On screen the worker sees the scan reference, the preliminary category and the class scores, together with a reminder that a qualified professional must interpret the output. The scores are not calibrated probabilities, and the class order must remain paired with the checkpoint mapping or the labels would be wrong. A further risk is that a blurred or non-retinal image still receives a category. The project report calls safe handling of such images a requirement but describes no validated image-quality check. Until one is tested, the output must not be taken as proof that a disease is present or absent.

The PDF repeats the warning shown in the web view. It lists the scan and date, the preliminary category and scores, the model name where available and the professional-review note. A downloaded report should never sound more certain than the screen. We also want a worker's review to look different from automated inference, and a report should not imply that a clinician confirmed a category unless one did. When notifications are enabled, their delivery status is logged separately from report creation. In a later integrated version, each image should also be tagged as an upload or a device transfer.

5) Patient-scoped access, security and privacy. Patients sign in through the mobile app and read only their own reports. On each protected request the server identifies the patient from a verified session and confirms that the patient owns the requested item; a patient ID sent by the app is never trusted on its own. A patient with no reports sees an empty list, and an expired session prompts a new sign-in. The link between result, PDF, scan and patient is checked at five points: upload, analysis, database write, report creation and retrieval, which lowers the chance that one patient's result appears in another's session.

JWT authentication protects the endpoints that handle patient data, and passwords are stored as hashes. Images and PDFs are never kept at guessable paths, and every upload is checked for type and size. Secrets, HTTPS, allowed origins, file permissions and upload limits would be reviewed before deployment. Demonstration data should be synthetic or de-identified, and logs should exclude personal health details. All of this is a set of self-imposed requirements, not a security certification.

C. Algorithms Used

1) CLAHE contrast enhancement. CLAHE (Contrast Limited Adaptive Histogram Equalisation) boosts contrast within small image regions, and its built-in clip limit prevents excessive noise amplification. Its parameters remain to be confirmed from the code.

2) Circular fundus-field masking. The circular mask blanks the dark area surrounding the fundus so that the classifier sees only the retina.

3) Pixel normalisation. Normalisation rescales pixel values to the convention used in training; without it, the inference input would differ from the data the model learned from.

4) Vision Transformer (ViT-Base Patch16-224). A ViT first divides the image into fixed-size patches. Each patch is embedded, position information is added, and transformer encoder layers then relate the patches to one another [6]. A 224 × 224 input with 16 × 16 patches yields 196 patches. A classification head maps the result to one of five categories: Normal, Diabetic Retinopathy, Glaucoma, Cataract and AMD. The class order in the checkpoint must match the label mapping exactly, and the output scores are relative values rather than calibrated probabilities.

5) Upload-to-report procedure. Algorithm 1 gives the screening steps of the project workflow; it does not claim that every test behind them has passed. The server verifies the session, the patient and the input before running the model. The result and the report are stored against the scan, so a later request does not depend on a file path held by the client.

Algorithm 1: Preliminary retinal screening
Input: worker session, patient ID, retinal image
Output: scan record, model output, screening PDF
1: Check the session and the worker's permission
2: Check the patient link and that an image is present
3: Check the image type and size limit
4: If invalid, return an error
5: Create the scan record and store the image reference
6: Decode the image and resize it to 224 × 224
7: Apply CLAHE, the circular mask and normalisation
8: Load the ViT checkpoint and the ordered class mapping
9: Run inference and collect the five class scores
10: Save the class, scores and scan status
11: Build the PDF with the preliminary-result notice
12: Return the scan and report references to the authorised client

6) Authorised patient report access. Algorithm 2 shows how the server determines who is asking and releases a report only to the patient who owns it.

Algorithm 2: Authorised patient report access
Input: patient JWT, requested report ID
Output: PDF or controlled error
1: Check the token signature, role and expiry
2: Get the patient identity from the verified claims
3: Load the report and the ownership of its scan
4: If the patient does not own the report, deny access
5: If the report exists and the patient owns it, return the PDF
6: Otherwise return a not-found or authentication error

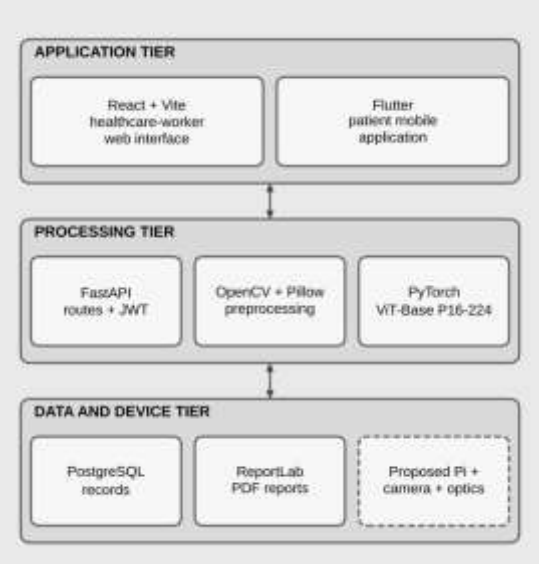
7) Safe failure handling. Algorithm 3 covers the response to a failed input check or dependency. A failed operation must never look like a completed analysis, and the API should keep validation errors, unavailable services, denied access and successful results clearly distinct.

Algorithm 3: Request failure handling
1: No image supplied: reject before analysis
2: Bad file type or decoding fails: reject safely
3: Unknown scan ID: return a not-found error
4: Ownership check fails: deny report access
5: Database unavailable: do not claim the result was saved
6: Checkpoint unavailable: do not return a prediction
7: Image quality uncertain: keep the screening caution

D. System Architecture

1) Three-tier architecture. VisionCare is divided into three tiers: application, processing, and data and device (Fig. 1). User interfaces communicate only with the API and never reach the database or the model directly. The application tier contains the React/Vite interface for healthcare workers and the Flutter app for patients. Neither client connects to PostgreSQL or loads the checkpoint, and the backend checks patient-specific access on each protected report request.

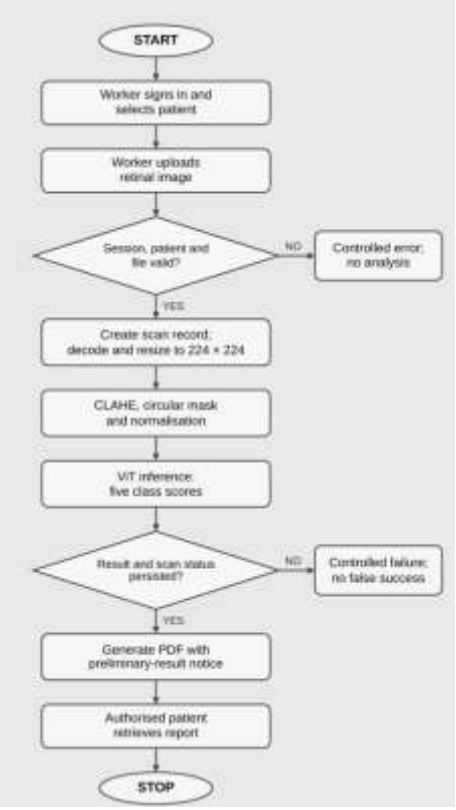
The processing tier hosts the FastAPI routes, preprocessing, PyTorch inference, PostgreSQL storage and ReportLab reporting. FastAPI validates requests and manages scan operations, calls preprocessing and inference, saves scan metadata and starts ReportLab. In the data model one account maps to one patient record, a patient may have many scans, and each scan has one result and one report. Notifications are optional and depend on configuration.



1. VisionCare three-tier block diagram (dashed block: proposed hardware, not integrated).

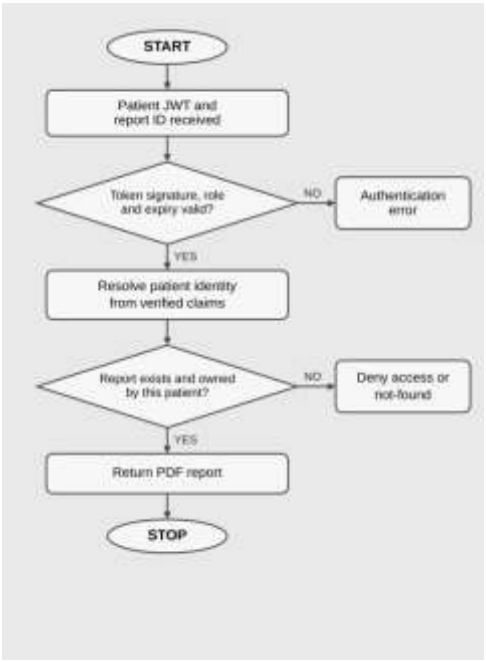
The acquisition tier is the proposed Raspberry Pi camera unit. Its assembly, optical alignment, image quality, lighting safety and image transfer remain unverified. Until they are checked, uploading an existing image is the separate software path that we demonstrate.

2) Screening workflow flowchart. Fig. 2 follows a single scan from the healthcare worker's sign-in through patient association, image validation, analysis, result storage and report creation, to retrieval by the patient. The result remains preliminary and tied to its scan record throughout, and the scan ID links each stage.



2. Software screening flowchart.

3) Patient report access flowchart. Fig. 3 shows how a report request is authorised. The patient app obtains reports through a protected API, never from a public, guessable path. The server rechecks permissions after a session expires, and a request for another patient's report is denied.



3. Patient report access flowchart.

4) Conceptual data model. The data model separates user identity from scan and report content (Table V). One account belongs to one patient record, a patient may have several scans, and each scan has one analysis result and one report. The report reference and creation time allow the mobile app to request the document later. Database constraints and backend checks should keep the scan, result and report bound to the same patient, especially when a record is updated or a request is retried.

V. MAIN DATA ENTITIES

Entity	Fields or information	Cardinality / role
User / Account	Login ID, role, status, password hash	One account per patient record
Patient	Patient ID and approved profile	One patient, many scans
Scan	Scan ID, patient ID, file, time, status	Each scan has one analysis and one report
Analysis result	Predicted class, scores, model reference	One result per scan
Report	PDF reference and creation time	Fetched by the authorised patient

E. Statistical Analysis Methods

This subsection describes how the model will be evaluated. None of it has been carried out, and the project report contains no such metrics. A labelled held-out dataset covering the same five categories is required. We will document its source and licence, the image count per class, how it is separated from the training and tuning data, the preprocessing settings, the checkpoint version, the evaluation date and the hardware used.

Precision, recall and F1-score will be computed for every class and a confusion matrix presented. Accuracy alone is insufficient, because it can look strong even when a class with few images is handled poorly.

$$Precision = TP / (TP + FP) \quad (1)$$

$$Recall = TP / (TP + FN) \quad (2)$$

$$F1 = 2PR / (P + R) \quad (3)$$

Here P and R denote precision and recall, and TP, FP and FN are the true-positive, false-positive and false-negative counts for a class. Low precision means many predictions for a class are wrong; low recall means examples of the class are being missed. The F1-score balances the two but is best read together with the class counts and the confusion matrix. Frequent confusion between two categories may stem from overlapping classes, poor image quality, ambiguous labels or inconsistent preprocessing between training and inference. We would inspect the images and annotations to find out, and we would not alter labels after seeing test results.

We will also report inference time, stating the hardware, batch size and warm-up and which stages are counted (decoding, preprocessing, model execution, saving and PDF creation). For reproducibility we retain the true and predicted label of each image so that the confusion matrix can be rebuilt. We will state how unreadable images, duplicates and ambiguous labels were handled, and the preprocessing order

will match that used at inference. If the test set changes, we will record the reason and keep the earlier result.

III. Problem Statement

Retinal disorders can cause irreversible vision loss. Specialist imaging equipment and trained eye-care professionals are unevenly distributed, particularly in remote and low-income communities. Screening therefore needs a workflow that connects image submission, automated first-pass analysis, patient records and follow-up while keeping a professional involved. A model output is screening support, not a diagnosis.

A. Existing Work

- Gulshan et al. developed and validated a deep-learning algorithm that detects diabetic retinopathy in retinal fundus photographs [1]. Ting et al. review artificial intelligence and deep learning across ophthalmology [2], and De Fauw et al. present deep learning for diagnosis and referral in retinal disease [4].
- These studies support image classification and clinical decision support. Their reported performance comes from their own datasets, protocols and systems, so it cannot be transferred to VisionCare.
- Portable acquisition matters where specialist devices are scarce. Reference [5] discusses a Raspberry Pi-based non-contact retinal imaging approach, and edge intelligence refers to running computation near the place where data originates [3].
- VisionCare adopts portability and connected follow-up as design goals, yet its ViT inference currently executes in the backend. Running the model on the proposed Raspberry Pi, including its memory use, energy use and inference speed, has not been measured.
- Earlier work targets particular diseases, datasets and referral settings. VisionCare is a five-category prototype with an integrated record and report workflow, so a direct comparison would need a shared dataset and evaluation protocol.
- Local edge inference, hardware image transfer and optical image quality have not been demonstrated, and the title states the project's aims. A web upload does not verify camera capture, and a model response shows neither accuracy nor clinical usefulness.

B. Objectives

1) Preliminary screening support. Present every model output as screening support, accompanied by a note that a professional must interpret it. It is neither a diagnosis nor a basis for treatment decisions.

2) Connected records and reports. Keep the image, the result, the patient record and the PDF report linked from upload through patient retrieval.

3) Security and privacy. Protect patient data through JWT authentication, server-side ownership checks and protected report retrieval.

4) Portable acquisition (proposed). Progress toward a portable Raspberry Pi imaging unit. This depends on validated image quality, safe lighting, reliable power and a verified path from capture to review.

5) Separate evidence. Report evidence from software upload, physical capture, model evaluation and clinical validation separately, so that each claim refers to the part that was actually tested.

IV. Methodology

The work is organised into five phases: a) project scope and feasibility; b) software development method; c) testing and verification method; d) release traceability and review; and e) hardware integration and evaluation method.

A. Project Feasibility Study

1) Operational feasibility. The software route begins with an existing image uploaded by a healthcare worker. The image then passes through validation, preparation, model inference, database storage, PDF generation and patient retrieval. A healthcare worker remains in the loop, and the report directs the patient to professional interpretation. A model score is not a confirmed diagnosis and should not be the sole basis for a treatment decision. Development uses a browser-based React/Vite client, a Python FastAPI service served by Uvicorn, PostgreSQL and a Flutter app.

2) Technical feasibility. The technology choices are as follows.

- **Frontend:** React with Vite for healthcare workers, and Flutter with Dart for patients.
- **Backend:** Python FastAPI REST services served by Uvicorn, with JWT authentication.
- **Image processing:** OpenCV and Pillow handle decoding, CLAHE, masking and normalisation.
- **AI model:** PyTorch ViT-Base Patch16-224 with a five-class head.
- **Data and reports:** PostgreSQL stores the records and ReportLab builds the PDF reports.
- **Hardware:** a Raspberry Pi acquisition unit, proposed but not yet assembled.

Portability depends on image quality, safe lighting, reliable power and a validated path from capture to review. Until hardware and software are integrated, uploading an existing image and capturing with the camera are treated as two separate input routes.

B. Testing and Verification Approach

The project report divides testing into unit, API, integration, interface, security, model and hardware tests. Unit tests cover validation, preprocessing, label mapping and report formatting. API tests cover upload, analysis, retrieval and error cases. Integration tests connect React, FastAPI, PostgreSQL, the model and Flutter, and security tests attempt unauthorised requests and access to another patient's data. Hardware checks must wait until the device exists.

Module-level checks use test images to confirm decoding, the 224 × 224 size, mask behaviour and normalisation, and to compare the class order with the checkpoint. A generated PDF is opened to look for the scan ID, date, class, scores, model information and disclaimer. API tests verify that a

failed operation leaves no false success behind. Integration testing follows the real service path: web upload, scan record, analysis request, preprocessing, model, PostgreSQL result, PDF generation and patient retrieval. The scan ID links the evidence, and personal details stay out of the logs.

A software release is verified in three steps. First, isolated checks cover decoding, missing and malformed files, image size, mask behaviour, normalisation, label order and PDF content, and the API is exercised with valid and invalid requests, unknown scan IDs and unavailable dependencies. Second, the whole interface path is run with synthetic or de-identified data: one scan ID is followed from patient selection to patient retrieval, and the PDF is compared with the stored record. If a patient requests another patient's report, the server must deny it, even after a session has expired. Third, we record the build, environment, input identifiers, expected and observed output, status and evidence.

Every reproducible case records the test ID, build, environment, input, expected behaviour, actual behaviour, status and evidence. A screenshot shows how the interface looked, but confirming that data was saved and that access control works may also require API logs and database checks. Inspecting a PDF tells us about its content, not about model accuracy, and a captured image says nothing yet about clinical adequacy or lighting safety. If the hardware or the checkpoint is unavailable, the case is marked Blocked, and we record what was actually observed rather than what was intended. Notifications and hardware transfer stay optional or proposed until they have been demonstrated.

C. *Release Traceability and Review Gate*

A release record names the software build, model checkpoint, ordered class map, database schema version, preprocessing settings and test environment. For each synthetic test we note the scan ID and keep the request, response, database state and generated PDF so that the result can be reproduced later. Before deployment we compare the checkpoint's ordered labels, the preprocessing parameters and the recorded model version. If they differ, the result is not treated as valid, and any changed setting requires a new evaluation record.

After a software change, the affected tests are rerun and the build ID is recorded. A new model needs its own checkpoint identity and evaluation record, and a hardware change needs a fresh capture and image-quality check. The release review examines the path from upload to analysis, the report fields, patient ownership and the failure responses. The software is ready only when authenticated upload, analysis, saving, PDF generation and patient-scoped retrieval all work, and the device is ready only after focus, lighting, power and transfer have been verified. Anything unfinished remains pending, partial tests cannot support a claim about the whole system, and only evidence actually observed counts as Passed.

D. *Hardware Integration and Evaluation Method*

The hardware will be built in stages. The camera, lens, lighting, power source and mount are assembled first and the setup is documented. Focus, field coverage and repeatability are then checked under controlled conditions, and lighting safety is assessed before the device is used on anyone. Once image quality has been reviewed, image transfer is connected to the upload API and the link between patient and scan is confirmed. A successful transfer still says nothing about model performance.

An integration test should then follow a captured image through upload, analysis, database storage, report creation and retrieval by the authorised patient, recording the component models, optical alignment, working distance, lighting, power and network conditions. Running the model on the Raspberry Pi is a separate question: memory, energy and latency must be measured before any claim of edge execution is made.

V. RESULT

The project report provides no measured accuracy, precision, recall, F1-score, confusion matrix, inference time, success rate or clinical validation. It describes software screens and a workflow, and formal test outcomes are still "to be recorded". Raspberry Pi capture awaits the hardware. This section therefore presents the status of the evidence and the planned test cases rather than experimental performance.

A. *Evidence Status*

Table VI lists the existing evidence for each area and what must be recorded next. A case without observed evidence is not Passed.

VI. EVIDENCE AVAILABLE AND WHAT IS STILL NEEDED

Area	Evidence	Next record
Web upload	Screens described	Upload outcomes
API / database	Routes identified	Request and persistence
AI model	Checkpoint identified	Held-out metrics
Patient reports	Workflow described	Ownership tests
Pi hardware	Proposed only	Capture and safety

B. *Planned Functional and Security Test Cases*

The planned cases are valid upload, missing image, unsupported or malformed file, scan analysis, unknown scan ID, report generation, own-report retrieval, cross-patient denial, database outage, missing checkpoint, low-quality input, invalid credentials, missing or expired token, empty report list, report content and camera capture. Each record states the test ID, build, environment, de-identified input, expected and observed response, status and evidence. The environment record also holds the software version, operating system, browser or device, backend and database versions, input identifiers and test date, and the acceptance condition is fixed before the test is run.

C. *Limitations*

- The capture hardware is only proposed and unvalidated, and edge inference on the Raspberry Pi has not been shown to work. The model path is described in the backend.
- Training data, class balance, checkpoint origin, preprocessing parameters and held-out metrics are undocumented.

- Functional and security test outcomes are still pending.
- Given these limits, no claim can be made about clinical usefulness, accuracy, speed, portability or safety.

Until independent technical evaluation and clinical validation are complete, the classifier serves only as preliminary support. A trained eye-care professional must interpret the image and the result, and the reports must not recommend treatment. Any later clinical study should first define the intended population, workflow, oversight and intended use before collecting evidence.

D. Reporting of Future Results

A future results section should state the exact software build, model checkpoint, data split, class counts and test conditions, and should keep functional success, model performance and image-acquisition safety separate. When a metric is unavailable, the paper should say so plainly instead of estimating it from screenshots or from results published for other systems.

VI. CONCLUSION

VisionCare connects a healthcare-worker interface, FastAPI services, image preprocessing, a five-class ViT, PostgreSQL records, PDF reports and Flutter patient access. This paper has set out its architecture, diagrams, algorithms, module responsibilities and evaluation plan. The Raspberry Pi device is not yet integrated, no test outcomes or model metrics exist, and clinical validity has not been shown. VisionCare is a screening-support prototype and not a diagnostic system.

The next milestone is evaluation that can be traced across every software and hardware boundary. Future work will build and test the acquisition unit, verify the class mapping and preprocessing against the training code, run the software tests listed here and evaluate the model independently. Results must be reported separately: a web upload test does not validate camera capture, a model metric does not show that image acquisition is safe, and a working report does not prove clinical benefit. Any later clinical use requires qualified professional oversight and proper clinical validation. Until then, results remain preliminary and should direct patients to professional eye care.

REFERENCES

1. V. Gulshan, L. Peng, M. Coram et al., “Development and Validation of a Deep Learning Algorithm for Detection of Diabetic Retinopathy in Retinal Fundus Photographs,” *JAMA*, vol. 316, no. 22, pp. 2402–2410, 2016.
2. D. S. W. Ting, L. R. Pasquale, L. Peng et al., “Artificial Intelligence and Deep Learning in Ophthalmology,” *Br. J. Ophthalmol.*, vol. 103, no. 2, pp. 167–175, 2019.
3. Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang, “Edge Intelligence: Paving the Last Mile of Artificial Intelligence with Edge Computing,” *Proc. IEEE*, vol. 107, no. 8, 2019.
4. J. De Fauw, J. R. Ledsam, B. Romera-Paredes et al., “Clinically Applicable Deep Learning for Diagnosis and Referral in Retinal Disease,” *Nat. Med.*, vol. 24, pp. 1342–1350, 2018.
5. P. Wankhede, T. Bhusari, V. Tammewar, R. Agrawal, C. Dhule, and N. C. Morris, “IoT and AI-ML Based Portable Non-Contact Retinal Imaging for Glaucoma and Eye Disease Monitoring at Home,” in *Proc. 2025 12th Int. Conf. Emerging Trends in Engineering and Technology – Signal and Information Processing*, Nagpur, India, 2025.
6. A. Dosovitskiy et al., “An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale,” in *Proc. ICLR*, 2021.
7. R. T. Fielding, “Architectural Styles and the Design of Network-Based Software Architectures,” Ph.D. dissertation, University of California, Irvine, 2000.
8. M. Jones, J. Bradley, and N. Sakimura, “JSON Web Token (JWT),” IETF RFC 7519, May 2015.
9. A. Paszke et al., “PyTorch: An Imperative Style, High-Performance Deep Learning Library,” in *Advances in Neural Information Processing Systems*, vol. 32, 2019.
10. React documentation. [Online]. Available: https://react.dev
11. FastAPI documentation. [Online]. Available: https://fastapi.tiangolo.com
12. G. Bradski, “The OpenCV Library,” *Dr. Dobb’s Journal of Software Tools*, 2000.
13. Flutter documentation. [Online]. Available: https://docs.flutter.dev