Development of An Ensemble Deep Learning Model for Brain Stroke Detection and Classification in MRI Scans

BY

LEE DING KUAN

A REPORT

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfillment of the requirements

for the degree of

BACHELOR OF INFORMATION TECHNOLOGY (HONOURS) (COMMUNICATIONS AND NETWORKING)

Faculty of Information and Communication Technology

(Kampar Campus)

JAN 2026

COPYRIGHT STATEMENT

This Final Year Project report is submitted in partial fulfillment of the requirements for the degree of Bachelor of Information Technology (Honours) (Communications and Networking) at Universiti Tunku Abdul Rahman (UTAR). This Final Year Project report represents the work of the author, except where due acknowledgment has been made in the text. No part of this Final Year Project report may be reproduced, stored, or transmitted in any form or by any means, whether electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the author or UTAR, in accordance with UTAR's Intellectual Property Policy.

ACKNOWLEDGEMENTS

First, I would like to thank my supervisor, Puan Nur Lyana Shahfiqa Binti Albashah for her guidance, support and encouragement during the development of this project. Her knowledge and feedback have provided invaluable insights and guidance.

I would also like to thank the Faculty of Information and Communication Technology (FICT), Universiti Tunku Abdul Rahman (UTAR) for the support in conducting this project.

And I would like to thank my parents and family for their unconditional love, support and encouragement throughout my journey.

Last but not least, I would like to thank the open-source community and other researchers for their past contributions and research in deep learning and medical imaging that has provided the groundwork for this project.

ABSTRACT

Stroke is leading cause of death and disability globally. More than half of all new strokes are ischemic strokes, which result from blood clotting and obstructing blood supply to the brain. Timely and accurate detection is essential, but interpreting magnetic resonance imaging (MRI) scans is lengthy and often inaccurate. This project created NeuroScan, an ensemble deep learning web application to predict ischemic stroke from brain MRI scans.

Six pre-trained deep learning models were fine-tuned: ResNet50, ResNet101, DenseNet121, DenseNet169, EfficientNet-B3 and Vision Transformer (ViT). The models were fine-tuned using a dataset of 15,120 MRI scans with labels from Hospital Pengajar Universiti Putra Malaysia (HPUPM), with 38 and 34 scans for validation and testing respectively. The ensemble learning methods tested were: simple average, weighted average, hard voting, and stacking. Three ensemble methods and ResNet50 models achieved perfect accuracy on the test set (1.000). Grad-CAM and Attention Rollout were adopted to offer visual explanations of the predictions.

Models were deployed in a Flask web app with Firebase Authentication and Firestore, incorporating three-layer role-based access control for patients, doctors and admins. The platform enabled the full flow for uploading an MRI, review by a doctor, and assisted by an AI chatbot (Claude Haiku), two notifications (in-app and email), export to PDF report, and security features such as content security policy (CSP) headers, rate limiting, session timeout and input validation. Docker was used to deploy the app on Hugging Face Spaces. No critical issues were found in 38 functional tests, 90 cross-browser tests and 10 security tests. The four project goals were completely met.

Area of Study: Deep Learning, Medical Imaging

Keywords: Brain Stroke Detection, MRI, Ensemble Learning, Explainable AI, Grad-CAM, Flask Web Application, Firebase, Role-Based Access Control

TABLE OF CONTENTS

POSTER **A-3**

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

CT	Computed Tomography
AI	Artificial Intelligence
ViT	Vision Transformer
CNN	Convolutional Neural Networks
XAI	Explainable Artificial Intelligence
FNN	Feedforward Neural Network
RNN	Recurrent Neural Network
GAN	Generative Adversarial Network
ML	Machine Learning
DL	Deep Learning
SVM	Support Vector Machine
ANN	Artificial Neural Network
FCN	Fully Convolutional Network
SGD-Net	Segmentation Guided Detector Network
YOLO	You Only Look Once
SSD	Single-shot Detector
DNN	Deeper Neural Network
OAR	Organs at Risk
DSC	Dice Coefficient

Chapter 1
Introduction

This chapter presents the background of the study, the problem statement and motivation behind this project, the research objectives, the project scope and direction, the contributions to the field, and the organisation of this report.

1.1 Background of Study

Stroke is a brain disease in which blood flow to a part of the brain is blocked, resulting in the death of brain cells. Stroke is the third largest cause of death, according to the World Health Organization [1]. It can be caused by either a blood clot in a blood vessel (ischemic stroke) or a ruptured blood vessel (haemorrhagic stroke). Ischemic stroke is more prevalent, comprising more than 62% of incident strokes [2]. Ischemic stroke damage is time critical. Saver [3] calculated the brain loses 1.9 million neurons per minute with an untreated stroke. This highlights the need for timely and accurate stroke diagnosis.

The diagnosis of a stroke is made using medical imaging. In the clinic, there are two main imaging techniques: computed tomography (CT) and magnetic resonance imaging (MRI). CT is often used to screen for haemorrhage. MRI is favoured for diagnosis in the case of ischemic stroke due to its superior soft tissue resolution and ability to detect early signs of ischemia that CT cannot [4]. Sequences like diffusion-weighted imaging (DWI) can reveal ischemic lesions within minutes of stroke onset, so MRI plays a significant role in the acute stroke pathway.

Artificial intelligence (AI) and deep learning (a type of AI) are increasingly used in medical image analysis. Convolutional neural networks (CNNs) like ResNet [5], DenseNet [6], and EfficientNet [7] have demonstrated good performance in medical image classification across a range of applications. Recently, transformer-based models such as the Vision Transformer (ViT) [8] have been used in medical image analysis, where self-attention is used to model long-range spatial interactions within an image. Ensemble learning, where multiple models are combined to make a prediction, has been demonstrated to enhance classification accuracy and stability over any individual model [9].

Explainable AI (XAI) is also a growing focus in medical AI. Methods such as Grad-CAM [10] for CNNs and Attention Rollout for transformers produce heatmaps that highlight the regions of an image the model used for its prediction. This allows physicians to interpret and validate the model's decision making process, which is crucial for ensuring trust in AI-based diagnosis.

1.2 Problem Statement and Motivation

While there have been significant advances in deep learning for medical applications, there are problems with the current methods for diagnosing ischemic stroke from MRI.

First, stroke diagnosis from MRI is currently largely reliant on radiologists. Interpretation of MRI scans for the diagnosis of ischemic stroke requires training, experience, and attention to detail in detecting subtle signs of stroke. Radiologists are in short supply, especially in developing nations where the need for imaging services outstrips supply [4]. Even with trained radiologists, there have been studies where the same radiologist may not agree with another radiologist's interpretation of an MRI scan [11]. Such variability is problematic in stroke diagnosis, as it delays treatment.

Second, the majority of deep learning studies for stroke detection use a single model architecture [12]. Using a single model is less reliable as the model's effectiveness is reliant on how well the single model architecture can capture the features. If it doesn't work on a specific input, there is no redundancy. This can be solved by using ensemble methods with multiple architectures, but few studies have used a combination of CNNs and transformers in an ensemble approach for brain stroke MRI classification.

Third, interpretability of deep learning models is a challenge. Models tend to be "black boxes" that provide a prediction without explaining how it was made. This makes it difficult for doctors to understand how the model reached a certain conclusion, and they are hesitant to rely on the model's predictions for clinical decision-making [4]. Although tools like Grad-CAM can explain the model's reasoning, most research has been focused on offline visualisation, rather than providing an interactive system.

Fourth, most research has only reached proof-of-concept [4]. Models are developed and tested on test data sets, but not integrated into systems doctors or patients can use. There is still a large gap between research-scale model development and deployment.

This project was inspired by these four issues. NeuroScan is an ensemble, deep learning, web-based application that overcomes these challenges: it integrates six model architectures for better generalisation, it uses Grad-CAM and Attention Rollout to explain predictions in the web interface for easy visualisation, and it is deployed as a free, publicly accessible, role-based web system with a full clinical workflow.

1.3 Objectives

The objectives of this project were:

1. **To fine-tune and evaluate multiple state-of-the-art deep learning models**

   In this project, this project fine-tuned pre-trained models including ResNet50, ResNet101, DenseNet121, DenseNet169, EfficientNet-B3 and Vision Transformer (ViT). The models were trained on an MRI dataset from Hospital Pengajar Universiti Putra Malaysia (HPUPM) to classify whether there is a clot or no clot in the MRI scans. The performance of all models was assessed by accuracy, precision, recall, F1-score, confusion matrix, receiver operating characteristic (ROC) area under the curve (AUC), precision recall (PR) AUC and calibration curves.

2. **To develop an ensemble deep learning model**

   An ensemble model was created to harness the synergies between convolutional neural networks (CNNs) and ViT. This project tried four different ensemble methods: averaging, weighted averaging, hard voting, and stacking. The goal of the ensemble model was to improve the performance and generalisation to unseen data over any single model.

3. **To implement explainable artificial intelligence (XAI)**

   Explainable AI was applied to overcome the "black-box" problem in deep learning models. Grad-CAM was used with CNN models, and Attention Rollout for the Vision Transformer. These methods produced heatmaps that indicated the areas of the MRI scan most important for the model's prediction, and promoted trust and transparency for clinicians.

4. **To deploy the ensemble model in a secure, user-friendly web application**

A web application was built using Flask, and integrated with Firebase for user authentication, and Firestore database. A three-level role-based access control with patients, doctors and administrators was set up. The system included several security features such as Flask-Talisman content security policy (CSP) headers, Flask-Limiter rate limiting, session timeout with countdown timer, magic byte file verification, and HTML-safe email templates. The web app was hosted online using Docker on Hugging Face Spaces.

It's important to note that this project did not include multi-class stroke classification. This project limited scope to binary classification, which is identification of the presence of a clot from an MRI and classification as clot or no clot. This project also has not been tested in clinical trials, or in hospitals, and not integrated with electronic health record (EHR) or real-time MRI scanners.

1.4 Project Scope and Direction

This project involved creating an ensemble deep learning model to diagnose ischemic stroke via brain MRI, and a secure web application. The key results of this project were:

- Several fine-tuned deep learning models (ResNet50, ResNet101, DenseNet121, DenseNet169, EfficientNet-B3, and Vision Transformer (ViT)) for detecting ischemic stroke.
- Ensemble model that integrates the predictions of six fine-tuned models using simple averaging, weighted averaging, hard voting and stacking to enhance classification accuracy and reliability.
- In-depth model performance evaluation and comparison using accuracy, precision, recall, F1-score, confusion matrix, ROC-AUC, PR-AUC, and calibration curves.
- Explainable AI visuals via Grad-CAM for CNN models and Attention Rollout for the Vision Transformer, offering visual explanations of a model's predictions.
- Web application with Flask and Firebase integration, role-based dashboards for patients, doctors and administrators, and a full doctor review and notification process.
- An AI chatbot (NeuroScan AI powered by Claude Haiku) to answer users' questions about stroke and the system.

- Export to PDF report functionality for patients to view their complete history of predictions, doctor reviews and clinical disclaimers.
- Internet deployment (using Docker) on Hugging Face Spaces: https://dancinggdogg-neuroscan.hf.space

This system was not deployed clinically or integrated into hospital systems. The project was confined to building a research-only system with proof-of-concept implementation.

1.5 Contributions

This project has contributed to deep learning-based diagnostics in healthcare, namely stroke. The project met the healthcare challenge by offering an easy-to-use system that can accurately and interpretably detect ischemic stroke from MRI images.

This project's main contributions are:

- Helping radiologists detect ischemic stroke with high accuracy and high speed, which can improve radiologists' clinical workflow to make quicker and more reliable decisions.
- An ensemble model, which is rarely used in stroke detection research, with five CNN models and one transformer model. The ensemble model achieved 100% accuracy on the test data.
- Visualisation of the reasoning process of the model using Grad-CAM and Attention Rollout in a web system. This overcomes the explainability issues in many current studies that only provide offline visualisation.
- Closing the gap between research and practice through the deployment of the system as a publicly available web-based service with role-based access, doctor review process, dual notification (in-app and email), and PDF report.
- Adoption of a variety of security techniques, such as content security policy headers, rate limiting, session handling and input sanitisation, suggesting a robust security mindset.

1.6 Report Organisation

This project report has seven chapters. Chapter 1 outlines the project context, problem statement and motivation, project goals, scope and contributions. Chapter 2 reviews the

literature and previous studies on stroke detection using deep learning techniques, highlighting the advantages, disadvantages and research opportunities in this domain. Chapter 3 describes the methodology used in this project, including the system design, system architecture, use case diagrams and sequence diagrams. Chapter 4 provides a description of the system design, application architecture, database design, machine learning process, and interactions between components. Chapter 5 presents the system implementation, including the software installation, configuration, system usage (with screenshots) and challenges in implementation. Chapter 6 details the system evaluation and discussion, including the system functional testing, cross-browser testing, security testing, model performance testing, and how each objective was achieved. The final chapter is the report conclusion, summarising the successes, limitations and future work.

Chapter 2

Literature Review

This chapter provides a literature and technology review, upon which this project was built. Section 2.1 reviews the deep learning architectures and technologies utilised in this project. Section 2.2 describes the existing systems and previous works on brain stroke detection using deep learning. Section 2.3 describes the data collection. Section 2.4 presents a comparison between systems and the proposed system. A summary of the literature is provided in Section 2.5.

2.1 Review of Technologies

This section describes the technologies used in this project, including the deep learning models, explainable AI methods, ensemble learning methods, and web technologies used for the web application.

2.1.1 Residual Networks (ResNet)

He et al. [5] developed ResNet to overcome the degradation issue in deep neural networks. Deeper neural networks typically start to saturate and then rapidly deteriorate in accuracy. ResNet did this by adding skip connections (also called residual connections) which allow the gradient to skip one or more layers, thus making it possible to train very deep networks without vanishing gradients. The idea was to work with the residual function, rather than the desired mapping [5].

There are several variants of ResNet based on the number of layers. ResNet50, with 50 convolutional layers, has become the most popular variant in medical imaging, as it strikes a balance between model depth and computational complexity. ResNet101 is deeper (101 layers) and has more capacity at the cost of additional computation. Both were used in this study. In medical imaging, [13] proposed a ResU-Net, which incorporated ResNet and U-Net for segmentation of organs at risk from CT images, and found that it demonstrated better performance than the traditional U-Net. Also, [14] designed a U-Net with ResNet50 backbone to obtain an accuracy of 97.9% for carcinoma classification. Moreover, [15] implemented a ResNet34 model on a CT imaging dataset of 1,222 patients with lung adenocarcinoma, and attained an accuracy of 87.76% for binary classification.

2.1.2 Densely Connected Networks (DenseNet)

Huang et al. [6] developed DenseNet on the theory that connecting each layer to all other layers in a feed-forward manner would enhance information flow and gradient propagation. While ResNet and DenseNet both use direct connections, there are key differences: ResNet adds feature maps via residual connections but DenseNet concatenates feature maps [16]. This connectivity pattern allows features to be reused and significantly reduces the number of parameters, compared to their plain counterparts.

DenseNet has been successful in medical imaging. [17] developed a 3D DenseNet model to perform multi-organ segmentation in breast cancer and obtained a dice coefficient (DSC) of 86%. Here, DenseNet121 (121 layers) and DenseNet169 (169 layers) were used. DenseNet121 was a lightweight and efficient network, whereas DenseNet169 was deeper to capture more features of MRI scans.

2.1.3 EfficientNet

EfficientNet was introduced by Tan and Le [7] to scale up convolutional neural networks. Traditionally, networks are scaled by independently increasing depth, width or input resolution. Tan and Le showed that scaling in all three dimensions with a compound coefficient achieves substantially better efficiency and accuracy. EfficientNet-B7 achieved 84.4% top-1 accuracy on ImageNet, surpassing all previous accuracy records, and 8.4 times smaller and 6.1 times faster than the best models [7].

EfficientNet models range from B0 (baseline) to B7 (largest). This project used EfficientNet-B3 as it strikes a balance between model size and efficiency, considering the target deployment scenario. EfficientNet-B3 provides a compound-scaled improvement over the baseline model, and is efficient enough for CPU-based inference on cloud services. It is also very efficient in terms of the number of parameters compared to ResNet and DenseNet models of comparable accuracy [7], due to the use of mobile inverted bottleneck convolutions (MBConv) and squeeze-and-excitation blocks.

2.1.4 Vision Transformer (ViT)

Vision Transformer (ViT) was proposed by Dosovitskiy et al. [8] in their research "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale". ViT showed that a pure transformer, originally developed for natural language processing, can be used on a sequence

of image patches to achieve state-of-the-art results on image classification tasks. Instead of using local receptive fields and convolutions, as found in CNNs, ViT uses self-attention to capture global dependencies between all patches of an image [8].

ViT splits an image into patches (e.g. 16×16 pixels) and linearly embeds and adds positional encodings to them, before passing them to a standard transformer encoder. The global representation is obtained by a special classification token ([CLS]) to generate the classification. But ViT generally needs to be pre-trained on large datasets (e.g. ImageNet-21k or JFT-300M) to achieve good performance, because it lacks the inductive biases (translation equivariance and locality) in CNN [8]. In the application of brain stroke diagnosis, [18] used ViT in conjunction with CNN to classify and localise brain stroke in CT images with an accuracy of 87.51%.

This project used the google/vit-base-patch16-224-in21k model, pre-trained on ImageNet-21k, and fine-tuned it for binary classification of MRI scans. This was the only transformer-based architecture used in this project and it provided diversity to the ensemble of CNN-based models.

2.1.5 Ensemble Learning

Ensemble learning is a machine learning technique that leverages the predictions of multiple base models to make a final prediction that is hoped to be more accurate than any of the individual models. The theoretical groundwork for ensemble learning was laid by Breiman [9], who showed that bagging (bootstrap aggregating) decreases variance and enhances generalisation by averaging the predictions of multiple models that have been trained on bootstrap samples of the training data.

Four ensemble methods were considered in this project. Arithmetic averaging is the arithmetic mean of the predicted probabilities from each model. Weighted averaging uses different weights for the predictions according to the performance of individual models, giving more weight to models that perform better. Hard voting takes the majority vote from the predicted labels of the models. Stacking fits a meta-learner (in this case, a logistic regression model) on the concatenated probability predictions of all models to learn how to best combine them [9]. The ensemble strategy was significant for this project as there is no single architecture that works best for different medical imaging datasets, as identified by [12].

2.1.6 Explainable AI (XAI)

The opaque nature of deep learning models makes them difficult to integrate into clinical practice, where it's essential for practitioners to understand how predictions are generated to gain confidence in their use. Explainable AI (XAI) methods provide visual or textual explanations for predictions.

Grad-CAM (gradient-weighted class activation mapping) is a method devised by Selvaraju et al. [10]. It relies on the gradients of the target class flowing into the final convolutional layer to generate a coarse localisation map, showing the important regions in the image for the target class. Grad-CAM can be applied to any CNN without requiring changes to the CNN architecture or training [10]. In this project, Grad-CAM was used with all five CNN models (ResNet50, ResNet101, DenseNet121, DenseNet169 and EfficientNet-B3).

Attention Rollout is a method to visualise attention in transformers. It combines attention weights from all layers of the transformer by multiplying attention matrices, resulting in a single attention map that displays the importance of each input patch in the decision-making process. This project used Attention Rollout with the Vision Transformer to create patch-based attention maps, offering comparable interpretability to Grad-CAM, but with the Vision Transformer.

2.1.7 Web Application Technologies

Technologies used for the web application. Flask is a simple Python web framework for building web applications, offering routing, request handling and templating with Jinja2. This framework was selected for its ease of use and ability to integrate with PyTorch model inferencing in the backend. Firebase services included Firebase Authentication for user login and registration (using email/password or Google OAuth), Cloud Firestore as the NoSQL database to store users, predictions, reviews, notifications and audit logs, and Firebase service account for server-side tasks. Jinja2 was the template engine for generating dynamic HTML pages with user-specific content.

The project also utilised DataTables.js for dynamic table pagination and search on the dashboards, Chart.js for creating doughnut and bar charts in the admin dashboard, ReportLab for creating PDF reports of predicted patient data, and Resend API for sending email

notifications to doctors for reviews. An AI chatbot was integrated using the Anthropic API to interact with Anthropic's Claude Haiku model to provide educational information about stroke.

2.1.8 Summary of Technologies Review

Table 2.1 Summary of Technologies Used in This Project

Technology	Role in This Project
ResNet50 / ResNet101	CNN-based base models for stroke classification; residual connections enable deep feature extraction
DenseNet121 / DenseNet169	CNN-based base models; dense connectivity promotes feature reuse with fewer parameters
EfficientNet-B3	CNN-based base model; compound scaling balances depth, width, and resolution efficiently
Vision Transformer (ViT)	Transformer-based base model; self-attention captures global spatial relationships in MRI scans
Ensemble Methods	Simple averaging, weighted averaging, hard voting, and stacking to combine base model predictions
Grad-CAM	XAI technique for CNN models; generates heatmaps highlighting prediction-relevant regions
Attention Rollout	XAI technique for ViT; aggregates multi-layer attention to show patch-level importance
Flask + Jinja2	Backend framework and template engine for role-based web application
Firebase	Authentication, Firestore database, and cloud service integration
Claude Haiku (Anthropic)	AI chatbot for stroke education and system guidance

2.2 Review of Existing Systems and Previous Works

This section surveys existing research on the application of deep learning to medical image interpretation (especially stroke diagnosis) and highlights the gap this project seeks to fill.

2.2.1 Deep Learning Methodologies in Brain Stroke Diagnosis

There are mainly three approaches to diagnose brain stroke using deep learning: classification, segmentation, and object detection [4]. Classification involves classifying MRI or CT scans

into various categories (such as stroke or no stroke) using algorithms that learn from labelled data to recognise patterns. Segmentation locates and outlines structures or lesions in the brain on a pixel-by-pixel basis using models such as U-Net, SegNet and Mask R-CNN [4]. Object detection identifies and classifies structures (or lesions) within the brain using bounding boxes, such as Faster R-CNN, YOLO and SSD [4].

This research project used the classification technique, in particular binary classification of brain MRI into clot (ischemic stroke) and no clot. Binary classification has been selected because it directly meets the clinical requirement for quick triage, which addresses the question whether there is a clot or not in the brain MRI.

2.2.2 CNN-Based Approaches in Medical Imaging

CNNs have been the most popular architecture in medical image analysis because they can learn hierarchical features automatically from images [4]. Popular CNN architectures include ResNet [5], DenseNet [6] and EfficientNet [7] which have all shown promising results in medical image analysis. For the particular task of brain stroke detection, [19] used CNN-based approaches and showed the feasibility of their approach to classify MRI images, laying the foundation for this work.

[13] developed a ResU-Net to segment organs at risk from CT scans and showed improved performance compared to U-Net. [14] proposed a U-Net with ResNet50 as backbone for semantic segmentation and classification of carcinomas with 97.9% classification accuracy. [15] used ResNet34 to train on CT images of 1,222 lung adenocarcinoma patients, and obtained 87.76% accuracy for binary classification and 80.61% for three-category classification.

[17] used a DenseNet 3D for multi-organ segmentation in breast cancer with a DSC of 86%. With ensemble methods, [20] combined VGG-19, ResNet152v2 and other models to build an ensemble approach for distinguishing between COVID-19, pneumonia and lung cancer with X-ray and CT imaging, reporting an accuracy of 98% and an F1-score of 98.24. Moreover, [21] used an ensemble of VGG, ResNet and DenseNet to classify breast ultrasound tumours, reaching an accuracy of 95.48%.

2.2.3 Transformer-Based Approaches

The transformer architecture, first introduced by Vaswani et al. [22] in the paper "Attention Is All You Need", has had an impact on the world of natural language processing (NLP) with its

attention mechanism. The transformer architecture was adapted to computer vision by Dosovitskiy et al. [8] in their paper "An Image Is Worth 16x16 Words: Transformers for Image Recognition at Scale", to demonstrate that a pure transformer directly applied to sequences of image patches can perform on par with CNNs for image classification.

In medical imaging domain, [18] combined ViT with CNN to classify and localise brain stroke in CT images with 87.51% accuracy. Their approach achieved better results than baseline models (VGG-19 and ResNet). However, [8] pointed out that ViT needs large-scale datasets for pre-training, which may not be suitable for small-scale medical image datasets. This was in line with the findings of this project, where ViT had a lower accuracy (88.24%) than CNN models when fine-tuned on the small-size HPUPM dataset.

2.2.4 Explainable AI in Medical Imaging

The importance of explainability in medical AI has been acknowledged. [4] noted that the lack of explainability in deep learning models is a challenge to their adoption in clinical practice. Selvaraju et al. [10] introduced Grad-CAM, which generates visual explanations from CNN models by computing localisations using the gradients flowing into the last convolutional layer. Grad-CAM has gained popularity in the medical imaging space as it can be used with any CNN model without retraining [10].

In the case of transformer-based models, Attention Rollout offers a similar explainability approach by aggregating attention from each transformer layer. [23] also made an advancement in explainability by extending ProtoPNet, which explains predictions by identifying prototypical parts of the input image. Unfortunately, many of the previous works focus on offline visualisations of Grad-CAM or attention maps, but do not provide them in an interactive platform. This research overcame this limitation by incorporating Grad-CAM and Attention Rollout directly into the web platform to enable doctors and patients to see the heatmaps together with the prediction outcome in real time.

2.2.5 Summary of Existing Systems

Through the literature review, it was shown that deep learning, particularly CNNs, has delivered impressive results in medical image analysis. But there were several issues found in the literature:

- Numerous studies used a single model architecture, which may not be optimal [12].

- Explainable AI was commonly offline and not in an interactive system.
- Research-level evaluation was common with a lack of deployment as a web application.
- Very few studies used CNNs and transformers in an ensemble approach for brain stroke classification from MRI.
- The problem of small data and absence of external validations continued [12], [24], [25].

2.3 Data Collection

The data set in this study was obtained from Hospital Pengajar Universiti Putra Malaysia (HPUPM). A total of 15,120 training images (7,560 clot and 7,560 no clot, balanced), 38 validation images and 34 testing images were included. The images were JPG files and were categorised into two labels: clot (presence of ischemic stroke) and no clot.

In the preprocessing step, all MRI images were resized to 224×224 pixels to meet the input size of the pre-trained deep learning models this project utilised. Standard ImageNet normalisation was applied using mean values of [0.485, 0.456, 0.406] and standard deviation values of [0.229, 0.224, 0.225]. Augmentation strategies were also applied to improve model generalisation and reduce overfitting, such as random horizontal flips, random rotations by ±10°, and colour jittering which randomly changed the brightness, contrast, saturation and hue. Data preprocessing was done with the torchvision.transforms library in PyTorch.

2.4 Comparison between Existing Systems and Proposed System

Although previous work has made many advances in deep learning for medical imaging, there were several limitations to previous systems that this project overcame.

First, previous research often used a single model, which is less robust. This work investigated six models (ResNet50, ResNet101, DenseNet121, DenseNet169, EfficientNet-B3, and ViT) and used four ensemble strategies to merge the predictions of different models, enhancing robustness and generalisation. Using an ensemble of five CNN models and one transformer model for brain stroke prediction from MRI scans was not reported in the literature.

Second, many of the previous studies adopted explainable AI approaches like Grad-CAM but limited their use to offline visualisations instead of an interactive system. However, this project integrated Grad-CAM and Attention Rollout into the web app, allowing users to see heatmaps along with predictions.

Third, many research efforts stopped at offline evaluations at the research level. This project went further to develop a Flask-Firebase-based web application with role-based access control (patient, doctor, administrator) to simulate a clinical scenario. The project also offered features like an AI chatbot, PDF report download, email alert system and security measures not found in other research prototypes.

Fourth, one of the shortcomings of models in the literature is their inability to generalise, especially for those trained on single-source data [12]. [24] investigated the impact of domain shift on medical imaging, and observed that CNN models trained on individual hospital systems do not generalise well to other data. This project recognised this limitation, since the dataset was obtained from a single hospital (HPUPM). But the use of an ensemble model was specifically designed to overcome such a challenge as ensemble models are known to be more resilient to noise, artefacts and variability in data compared to single models [9].

2.5 Summary

The review of literature indicated that deep learning is the current state-of-the-art for medical image analysis, including brain stroke diagnosis. CNN-based models like ResNet, DenseNet and EfficientNet have excelled in feature extraction, while emerging transformer-based models such as ViT have been shown to model global spatial information via the self-attention mechanism. Ensemble learning has been demonstrated to boost accuracy and reliability through the complementary nature of multiple models.

Despite these achievements, the literature review identified some gaps: single model approaches, lack of interactive XAI systems, lack of deployment for real-life applications, and the inability of models to generalise across datasets. This project overcame these limitations by ensemble five CNN and one transformer models, integration of Grad-CAM and Attention Rollout into a real-time web application, and the deployment of the entire system online with role-based security, two-factor authentication and other features such as an AI chatbot, notification system and export to PDF report.

CHAPTER 3

System Methodology/Approach

This chapter outlines the methodology used in this project, including the development model, user requirements, system requirements, system design and system architecture.

3.1 Methodology

The project had two main stages: the model development stage and the system application development stage. Each stage adopted the Waterfall software development model, which is divided into five phases: requirements, design, implementation, verification and maintenance. Each of the stages was fully completed before moving on to the other.

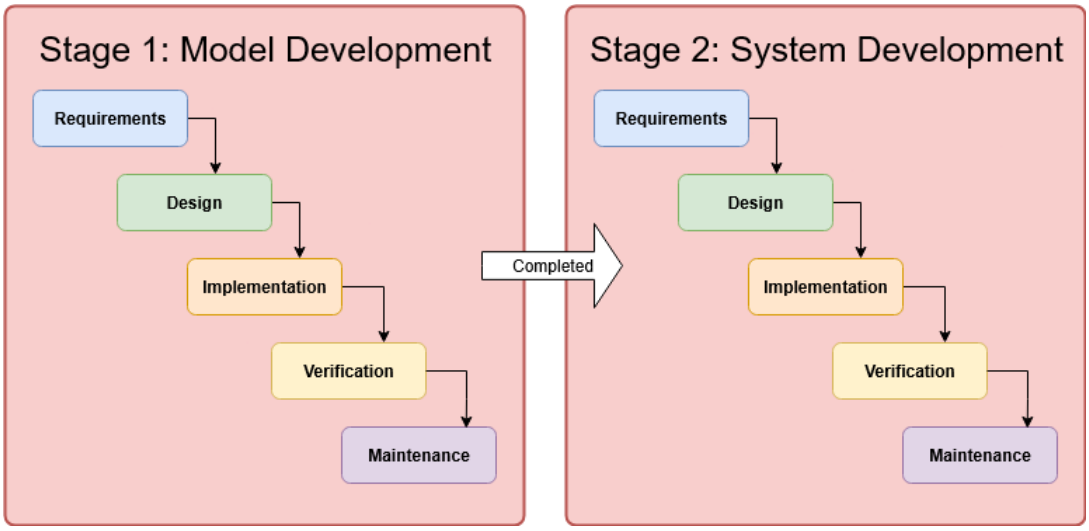

Figure 3.1 Two Stages of Waterfall Software Development Model

3.1.1 Model Development Stage

The phases of the model development stage are described in Table 3.1.

Table 3.1 Five Phases of Waterfall Model for Model Development Stage

Phase	Tasks
1. Requirement	• Defined the problem of detecting ischemic stroke from MRI scans using deep learning.

	• Collected labelled MRI datasets from HPUPM for training (15,120 images), validation (38 images), and testing (34 images). • Established model performance requirements. • Defined the need for explainable AI. • Set up Google Colab, PyTorch, and Visual Studio Code.
2. Design	• Selected pre-trained architectures: ResNet50, ResNet101, DenseNet121, DenseNet169, EfficientNet-B3, and Vision Transformer (ViT). • Planned the preprocessing pipeline (resizing to 224×224, normalisation, augmentation). • Decided training strategies including Adam optimiser, cross-entropy loss, and hyperparameter tuning. • Planned four ensemble methods: simple averaging, weighted averaging, hard voting, and stacking. • Selected evaluation metrics: accuracy, precision, recall, F1-score, ROC-AUC, PR-AUC, confusion matrix, and calibration curves.
3. Implementation	• Built the preprocessing pipeline using torchvision.transforms. • Implemented training code for each architecture. • Trained all six models on Google Colab using A100 and L4 GPUs. • Implemented Grad-CAM for CNN-based models and Attention Rollout for ViT. • Saved the best model checkpoints (.pth files) for each architecture.
4. Verification	• Evaluated all models on the held-out test set of 34 MRI scans. Compared results across all metrics. • Validated ensemble model performance against individual models. • Generated classification reports, confusion matrices, ROC curves, PR curves, and calibration plots.
5. Maintenance	• Organised model checkpoints into a dedicated folder structure. • Stored results and evaluation reports in Google Drive. • Froze best-performing models for deployment. • Uploaded model weights (~697 MB across 7 .pth files) to a separate Hugging Face model repository (DancinggDogg/NeuroScan-models) for runtime download during deployment.

3.1.2 System Development Stage

Once the model development stage was complete, the project moved on to the system development stage. The five phases are defined in Table 3.2.

Table 3.2 Five Phases of Waterfall Model for System Development Stage

Phase	Tasks
1. Requirement	• Defined three system roles: patient, doctor, and admin. • Defined functional requirements for each role (MRI upload, prediction viewing, doctor review workflow, admin user management, audit logging). • Defined non-functional requirements: role-based access control, secure data handling, session management, rate limiting. Set up Flask, Jinja2, Firebase Admin SDK, and all required dependencies.
2. Design	• Designed system architecture with Flask backend, Jinja2 templates, and Firebase services. • Designed role-specific dashboards for patient, doctor, and admin. • Planned Firestore database collections: users, predictions, reviews, notifications, and logs. • Designed the prediction workflow including model selection, inference, XAI heatmap generation, and result storage. • Planned security measures: Flask-Talisman CSP headers, Flask-Limiter rate limiting, session timeout, magic byte file validation. • Planned notification system (in-app bell and email via Resend API). Planned AI chatbot integration and PDF report export.
3. Implementation	• Developed Flask routes for authentication, prediction, review, role management, notifications, chatbot, PDF export, and settings. • Implemented Firebase Authentication supporting email/password and Google OAuth. Built role-based dashboards with Jinja2 templates, custom CSS (DM Sans font, CSS variables design system), DataTables.js, and Chart.js. • Integrated all six trained models and the ensemble into the Flask backend for real-time inference. • Implemented Grad-CAM and Attention Rollout visualisation in the prediction workflow. • Integrated Anthropic Claude Haiku API for the AI chatbot. • Implemented ReportLab-based PDF report generation.

	• Implemented Resend API for email notifications. • Deployed via Docker on Hugging Face Spaces.
4. Verification	• Tested role-based access control and role-specific restrictions. • Conducted functional testing of all features (prediction, review, notifications, chatbot, PDF export, settings). • Performed cross-browser compatibility testing across Chrome, Firefox, and Edge (90 test cases). • Tested security measures: rate limiting, file validation, session timeout, CSP headers. • Verified email notification delivery via Resend API.
5. Maintenance	• Performed UI/UX refinements based on testing feedback. • Applied security hardening: rate-limited login endpoint, HTML-sanitised email content. • Optimised Firestore queries by eliminating N+1 query patterns through batch fetching and user caching. • Cached ViT processor and GradCAM objects at startup for faster inference. • Parallelised ensemble prediction using ThreadPoolExecutor. • Maintained requirements.txt and Dockerfile for reproducible deployment.

3.2 User Requirements

There were three user roles in this project. The system's requirements for each category are outlined below.

Patients:

- Submit MRI scans in a secure manner in JPG, JPEG, or PNG format with magic byte verification.
- Choose between seven models (six single and one ensemble) to make predictions.
- See prediction results with confidence levels and Explainable AI heatmaps.
- View history of predictions with DataTables search and pagination in the patient dashboard.
- View doctor review results and comments in popups.
- Review the history of predictions in a PDF report.

- Get push notifications (bell icon with badge) and email notifications when a doctor reviews.
- Use the AI chatbot (NeuroScan AI, using Claude Haiku) to ask questions about stroke.
- Rename and reset password in the settings page.

Doctors:

- See a list of cases to be predicted in the doctor dashboard.
- View details of individual cases, including original MRI and explainable AI heatmaps.
- Provide review decisions (agree or disagree) with clinical comments.
- Ask the AI chatbot for stroke-related information.
- Upload MRI and run predictions.

Administrators:

- See all users with roles and dates of joining.
- Change user roles (patient, doctor and admin), and delete users (Firebase Auth and Firestore records).
- View all predictions made across the system, and delete prediction records.
- View system audit logs for user activities.
- View user and prediction statistics on Chart.js (doughnut and bar charts).
- Upload MRI scans and make predictions.

3.3 System Performance Definition

The following were the performance goals set at the beginning of the project. The results obtained are presented in Chapter 6.

- Accuracy: Single models to achieve an accuracy greater than 85%, and an ensemble of models to achieve an accuracy greater than 90%.
- F1-score: All models were targeted to achieve F1-score greater than 0.90 for sensitivity and specificity.
- ROC-AUC: All models were targeted to have an ROC-AUC greater than 0.95 for successful binary classification.

- Explainability: The heatmaps were expected to clearly localise the clot regions for CNN-based models (Grad-CAM) and the transformer model (Attention Rollout).
- Inference time: Prediction time was expected to be less than 10 seconds per image for prediction results including the explainable AI heatmap.

3.4 System Requirements

3.4.1 Hardware

Table 3.3 shows the hardware used for model development and system development.

Table 3.3 Hardware Specifications

Specification	Details
Device	Laptop
Model	Acer Nitro AN515-46
System Type	64-bit operating system, x64-based processor
Processor	AMD Ryzen 7 6800H with Radeon Graphics (3.20 GHz)
Graphics Card	AMD Radeon Graphics / NVIDIA GeForce RTX 3050 Laptop GPU
RAM	16 GB (4800 MT/s)
Operating System	Windows 11 Home Single Language

This project also used two types of GPUs in Google Colab for model training: the NVIDIA A100 (Ampere architecture, 40 GB HBM2e, 6,912 CUDA cores) and the NVIDIA L4 (Ada Lovelace architecture, 24 GB GDDR6, 7,424 CUDA cores). These GPUs allowed for much faster training than the laptop's CPU.

3.4.2 Software

This project used various tools and technologies in both stages, which are listed in Table 3.4.

Table 3.4 Software Specifications

Stage	Technology	Version	Usage
Model Dev	Google Colab	-	Cloud-based training with GPU acceleration
	Python	3.11	Programming language for model training and web application

	PyTorch	≥2.2.0	Deep learning framework for model training and inference
	torchvision	≥0.17.0	Pre-trained models and image transforms
	timm	≥0.9.0	EfficientNet-B3 model loading
	transformers	≥4.38.0	HuggingFace ViT model and processor
System Dev	Flask	≥3.0.0	Lightweight web framework for backend
	Jinja2	≥3.1.2	Template engine for dynamic HTML rendering
	Firebase Admin SDK	≥6.3.0	Server-side Auth, Firestore, and admin ops
	Flask-Login	≥0.6.3	User session management
	Flask-Talisman	≥1.1.0	CSP headers and security hardening
	Flask-Limiter	≥3.5.0	Rate limiting for endpoints
	DataTables.js	1.13.5	Interactive table pagination and search
	Chart.js	4.4.0	Dashboard charts (doughnut, bar)
	ReportLab	≥4.0.0	PDF report generation
	Anthropic SDK	≥0.40.0	Claude Haiku API for AI chatbot
	Resend	≥2.0.0	Email notification delivery
	Docker	-	Containerised deployment on HF Spaces

3.5 System Design

3.5.1 Model Development Flowchart

The model development stage process is shown in Figure 3.2.

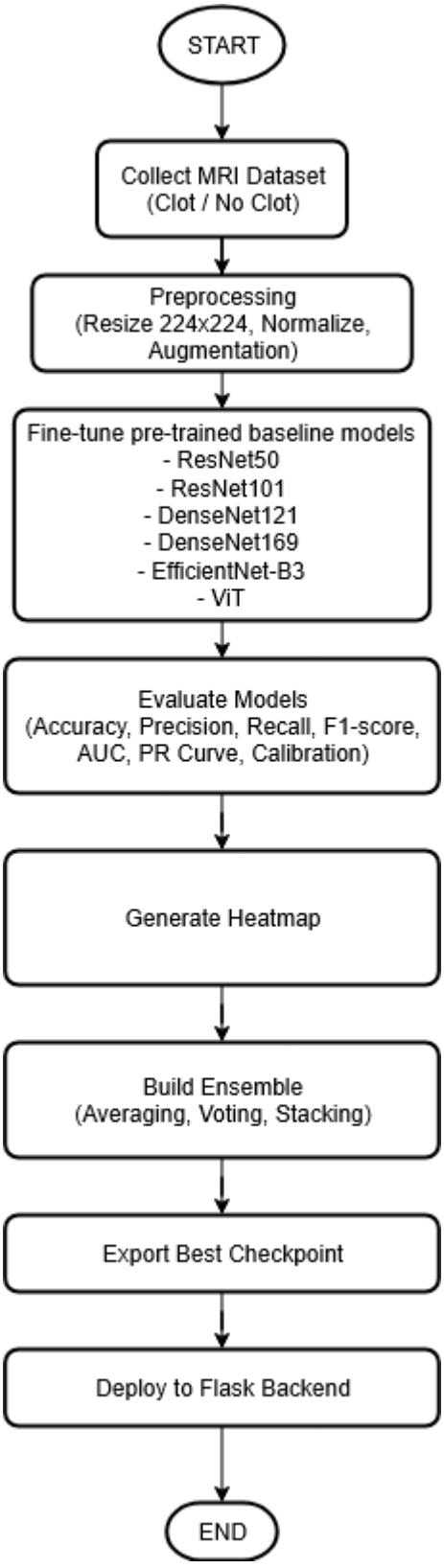

Figure 3.2 Model Development Flowchart

During the model development phase, MRI images of clot or no clot were obtained from HPUPM. The data was preprocessed by resizing the images to 224×224 pixels, normalisation

and augmentation. The training set was used to fine-tune six pre-trained models (ResNet50, ResNet101, DenseNet121, DenseNet169, EfficientNet-B3 and ViT). Performance metrics included accuracy, precision, recall, F1-score, Receiver Operating Characteristic (ROC) and Precision-Recall (PR) Area Under the Curve (AUC), confusion matrices, and calibration curves. Explainable AI techniques were used: Grad-CAM for the five CNN models and Attention Rollout for ViT. Additionally, ensemble models were built using four approaches (average, weighted average, hard voting, and stacking) to ensemble the base models. The best checkpoints were exported and deployed in the Flask backend.

3.5.2 Web Application Workflow

The system flow is shown in Figure 3.3. The flow chart demonstrated the full user flow from entering the website, then authenticating, accessing the role-based dashboards and common features. The main pages were linked to the detailed workflow flowcharts (Figures 3.4 - 3.9).

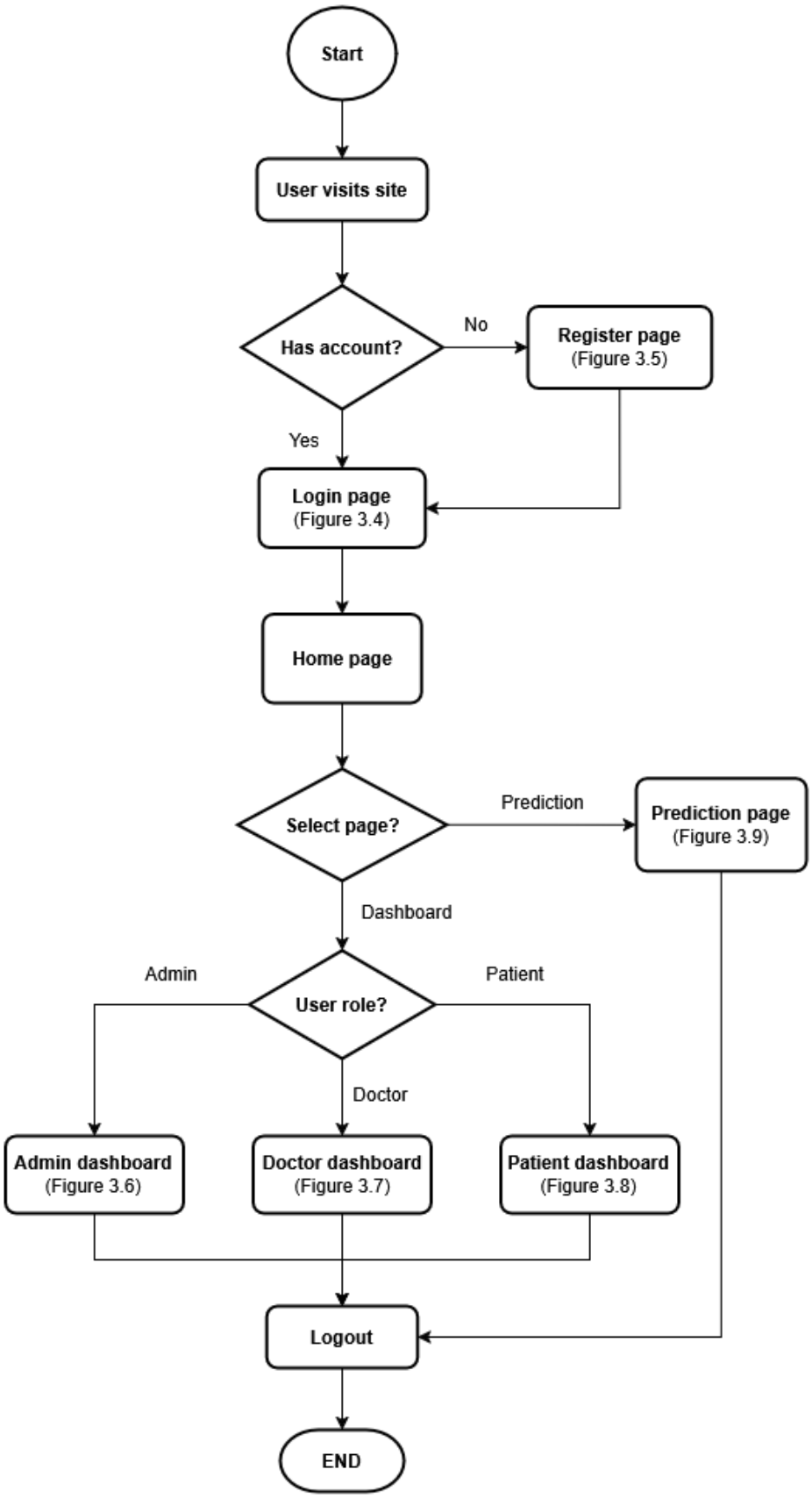

Figure 3.3 Main System Flowchart

The web application pages with role-based authentication included the following: Login, Register, Home, Prediction, Patient Dashboard, Doctor Dashboard, Admin Dashboard, Settings, About and Model Comparison. The workflow of each major page is shown in the following flowcharts.

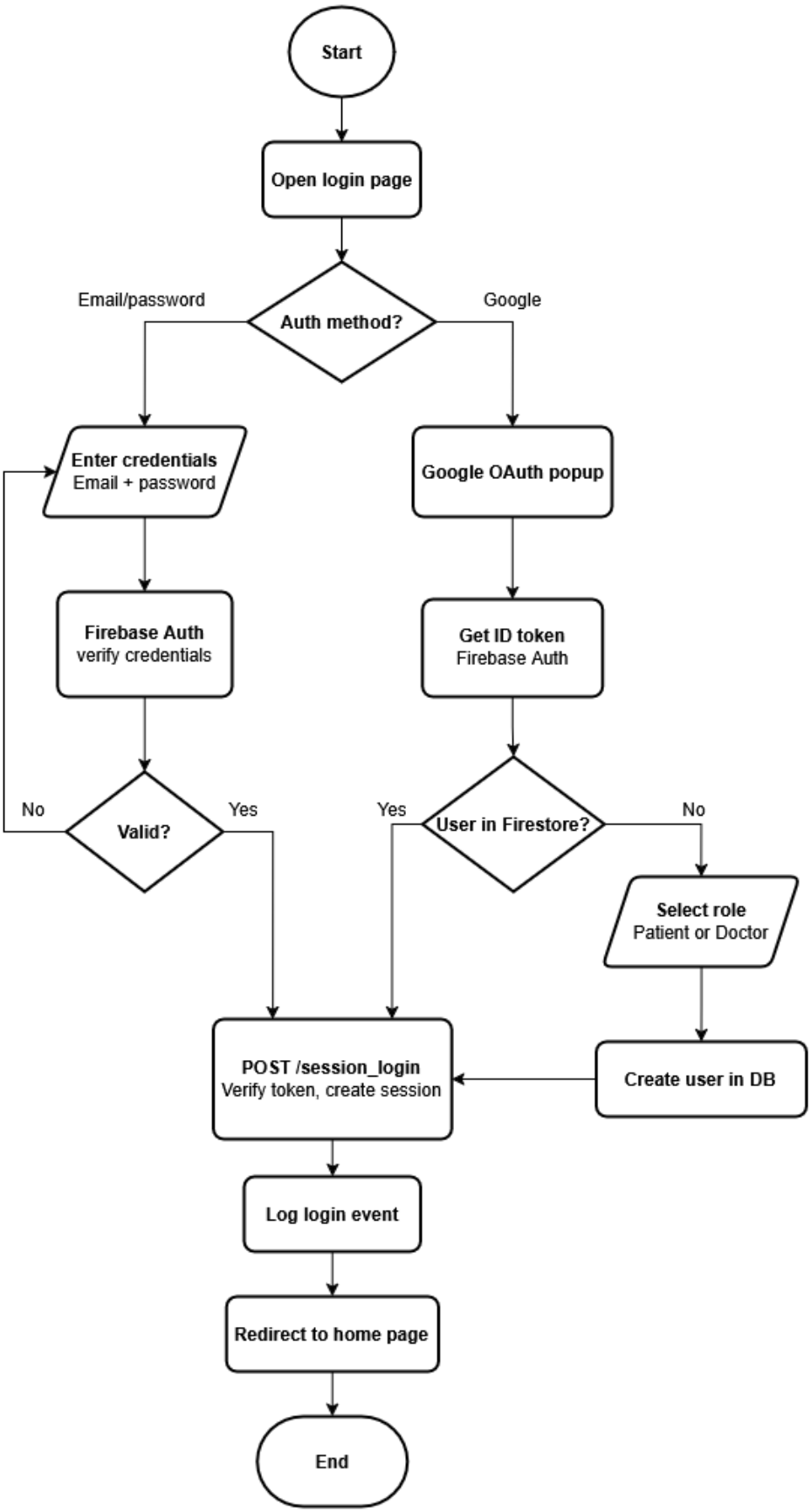

Figure 3.4 Login Page Workflow Chart

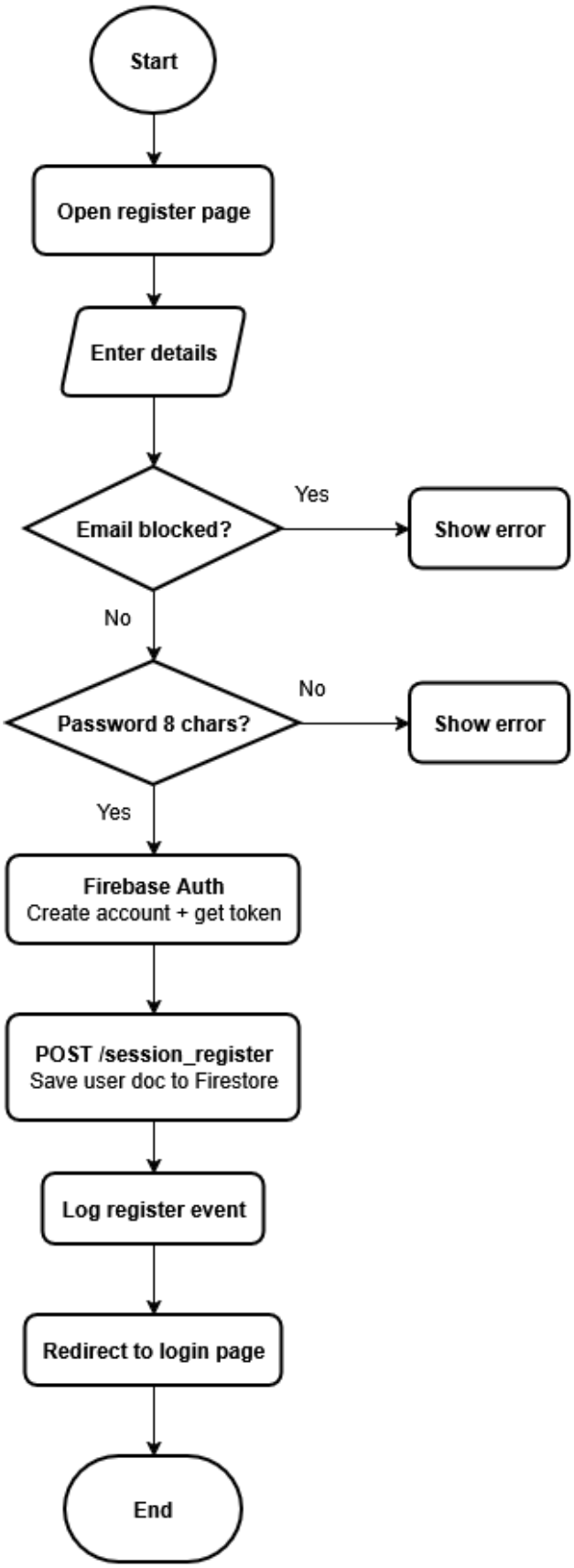

Figure 3.5 Register Page Workflow Chart

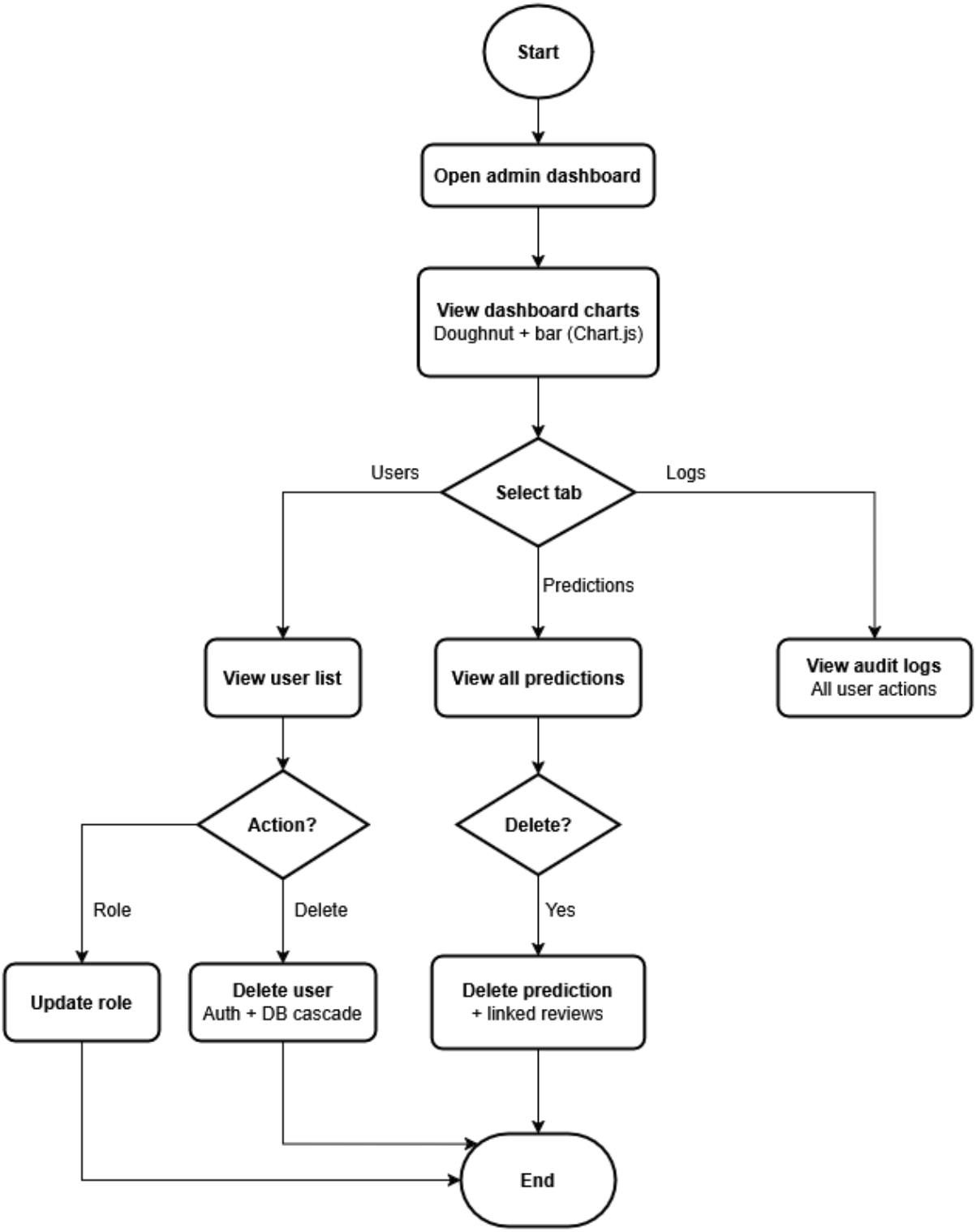

Figure 3.6 Admin Site Workflow Chart

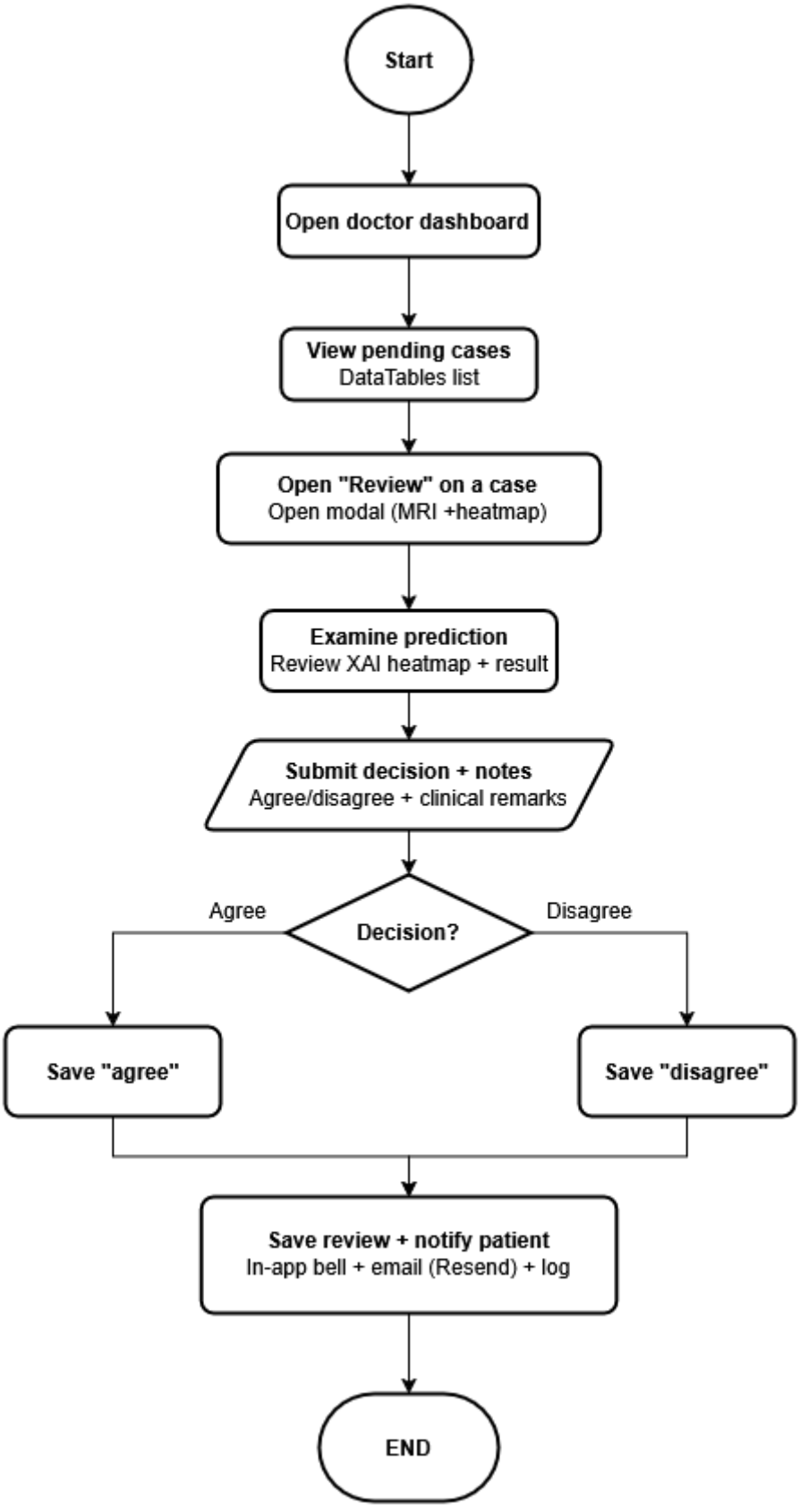

Figure 3.7 Doctor Site Workflow Chart

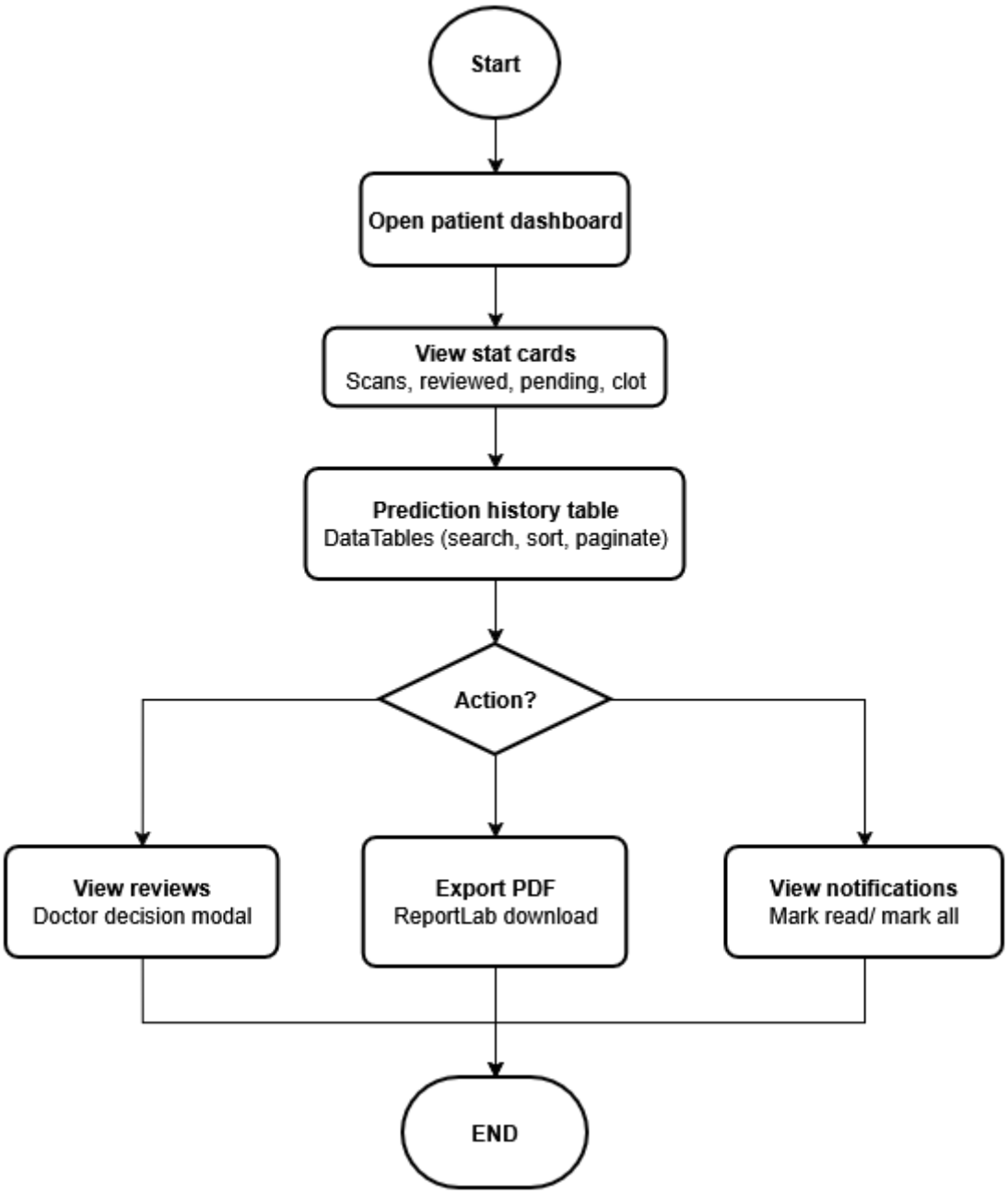

Figure 3.8 Patient Site Workflow Chart

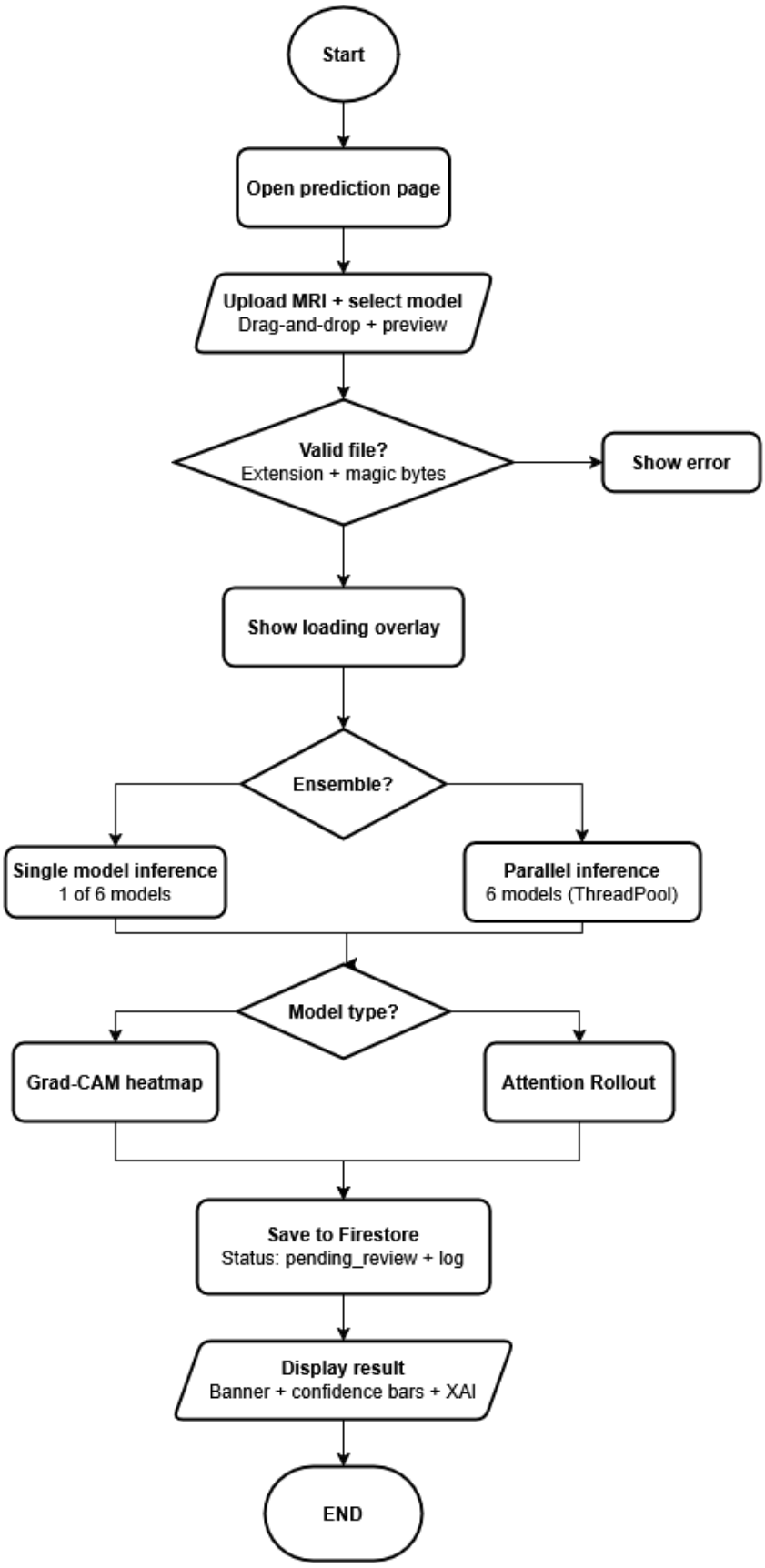

Figure 3.9 Prediction Page Workflow Chart

The pages' detailed workflows and role-based features were depicted in Figures 3.4 to 3.9. The login page (Figure 3.4) allowed email/password login and Google OAuth login. A modal dialog was used to require new Google users to choose a role (patient or doctor). The register page (Figure 3.5) checked email domains against a list of blocked domains and the password length against a minimum of 8 characters, before the Firebase Auth account and Firestore user document were created. The admin portal (Figure 3.6) had three tabs to manage users, predictions and audit logs, with cascade deletion for users and predictions. The doctor site (Figure 3.7) showed cases for review and informed the patient via in-app and email notifications when a review was submitted. The patient site (Figure 3.8) showed the prediction history, with the ability to view reviews, generate PDF reports and manage notifications. The prediction page (Figure 3.9) managed MRI upload, file verification (extension and magic bytes), predictions (single or ensemble using ThreadPoolExecutor), XAI heatmap (Grad-CAM or Attention Rollout), and the display of prediction results with confidence bars.

3.6 Use Case Diagram

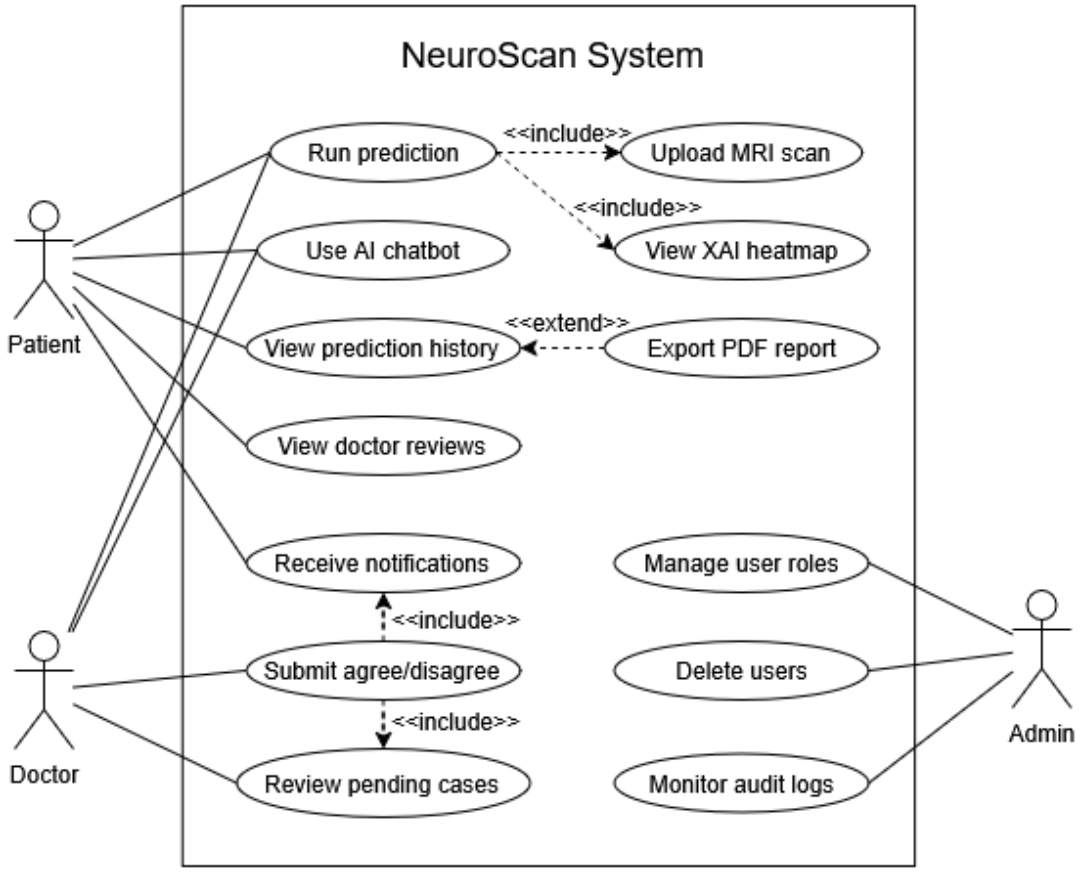

Figure 3.10 Use Case Diagram

The application had three main actors: patient, doctor, and admin. Certain features were common to all (such as MRI upload and prediction), while others were limited to certain actors. This allowed an efficient clinical process with privacy and privilege management.

The role definitions are outlined in Table 3.5.

Table 3.5 Actors' Role Specifications

Actor	Role Specification
Patient	The primary end-user who uploaded MRI scans for ischemic stroke detection. Received prediction results from various deep learning models with explainable AI heatmaps. Viewed doctor review decisions and clinical remarks. Received in-app and email notifications upon doctor review. Exported prediction history as PDF. Accessed the AI chatbot for stroke-related queries.
Doctor	The medical professional who reviewed AI-generated predictions. Examined original MRI scans alongside Grad-CAM or Attention Rollout heatmaps. Submitted review decisions (agree or disagree) with optional clinical notes. The review triggered notifications to the patient.
Admin	The system administrator responsible for managing user roles, deleting users and predictions, and monitoring system audit logs. Viewed user distribution and prediction statistics through dashboard charts.

The prediction module and the dashboard were mainly used by patients. And doctors functioned as verifiers to validate the model predictions to avoid sole reliance on the AI model. System administrators managed system operations, including user roles and logs. This structured interaction demonstrated the collaborative interactions in the system, with models making predictions, users receiving the initial results, and the doctors verifying the results to provide an additional level of assurance

3.7 Sequence Diagrams

To better explain the interactions between the actors and system, three sequence diagrams were created.

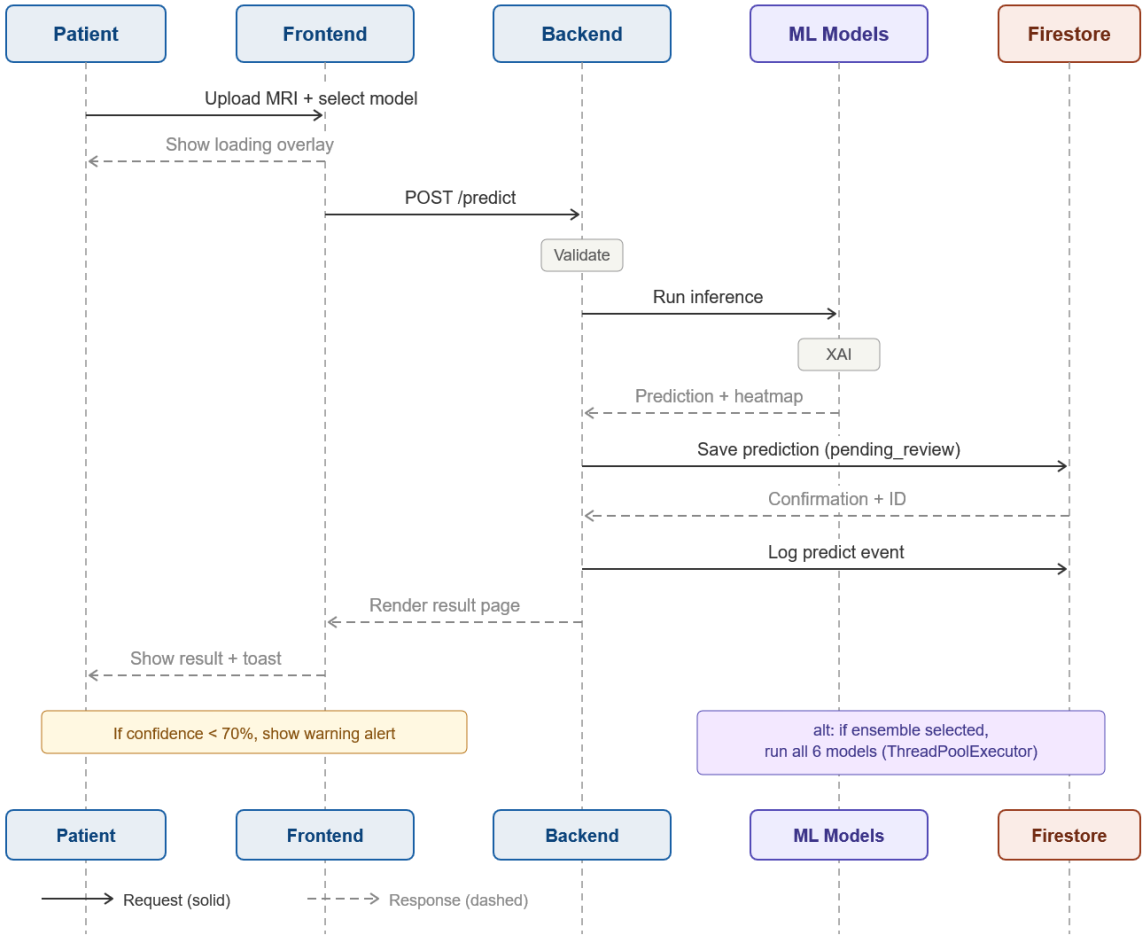

Figure 3.11 Sequence Diagram of MRI Prediction Flow

Figure 3.11 showed the interaction for a user to upload an MRI image. The MRI scan was processed by the Flask frontend, checked for file format and magic bytes, and sent to the Flask backend. The backend predicted the MRI scan using the selected deep learning model, produced an explainable AI heatmap (Grad-CAM or Attention Rollout) and saved the prediction result in Firestore with a status of "pending_review". The results of the prediction (class label, confidence scores, heatmaps) were returned to the patient. A message toast showed the prediction was saved.

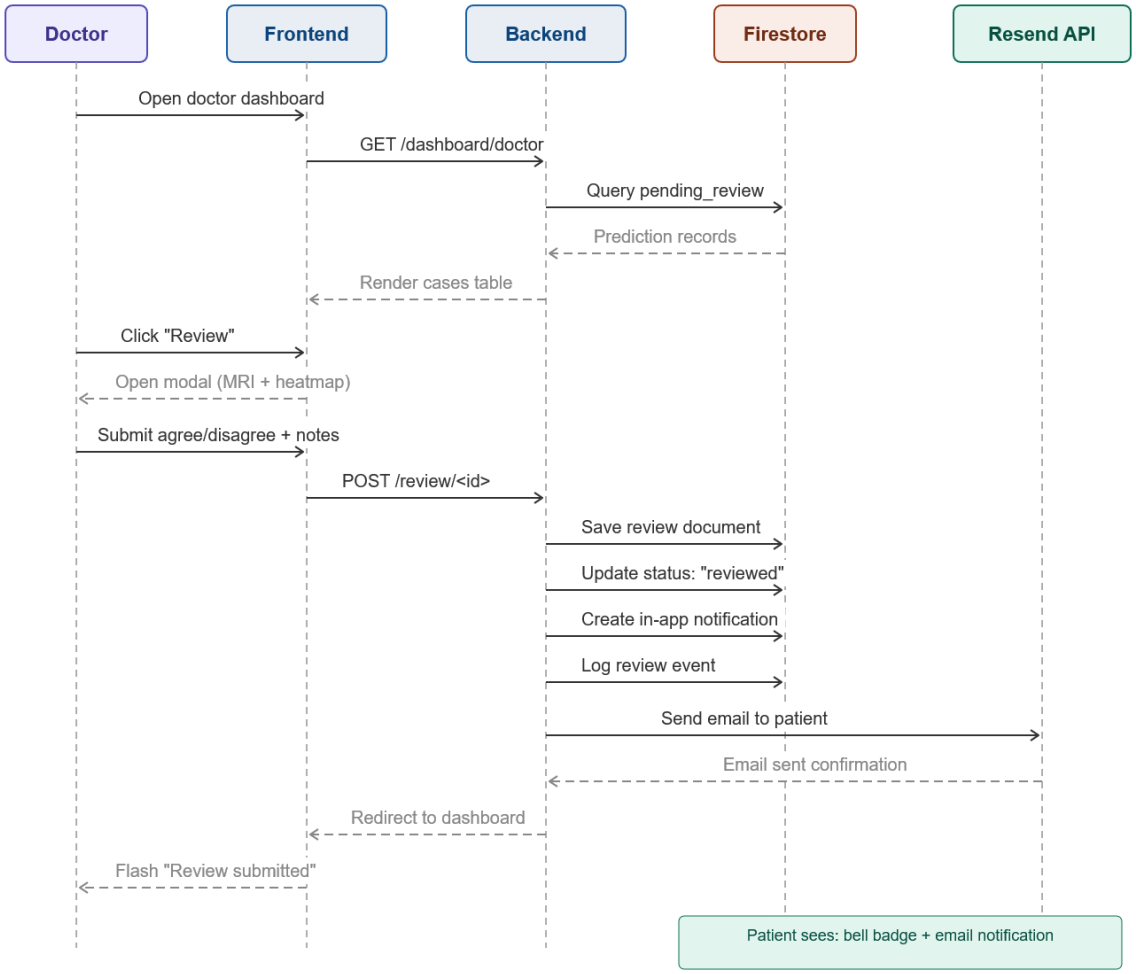

Figure 3.12 Sequence Diagram of Doctor Review Process

Figure 3.12 depicted the process when a doctor logged in and viewed the pending list in the doctor dashboard. This dashboard searched the Firestore predictions collection for status “pending_review”. The doctor could review the prediction details, such as the original MRI scan and explainable AI heatmap, in a review modal. Once the decision (agree or disagree) was made and any clinical notes were entered, the Flask backend recorded the review in the reviews collection, changed the status of the prediction to "reviewed", inserted an in-app notification into the notifications collection, sent an email notification to the patient using the Resend API and logged the action in the logs collection.

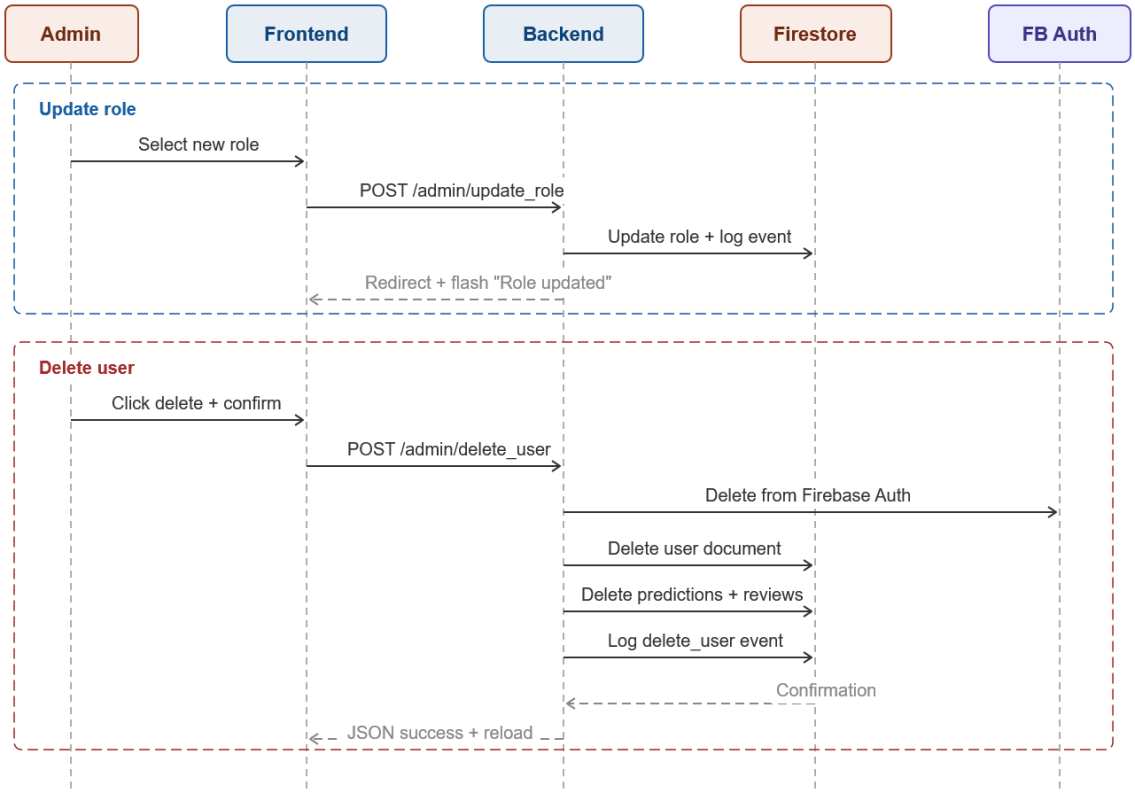

Figure 3.13 Sequence Diagram of Admin Role Management

Figure 3.13 illustrated the admin's actions when assigning roles. The admin could select a user and change their role from the admin dashboard. The backend updated the user document in Firestore, and also stored the change in the logs collection. The admin could also remove users (deleting the user from Firebase Auth and all the associated Firestore documents) and remove predictions and reviews.

3.8 System Architecture

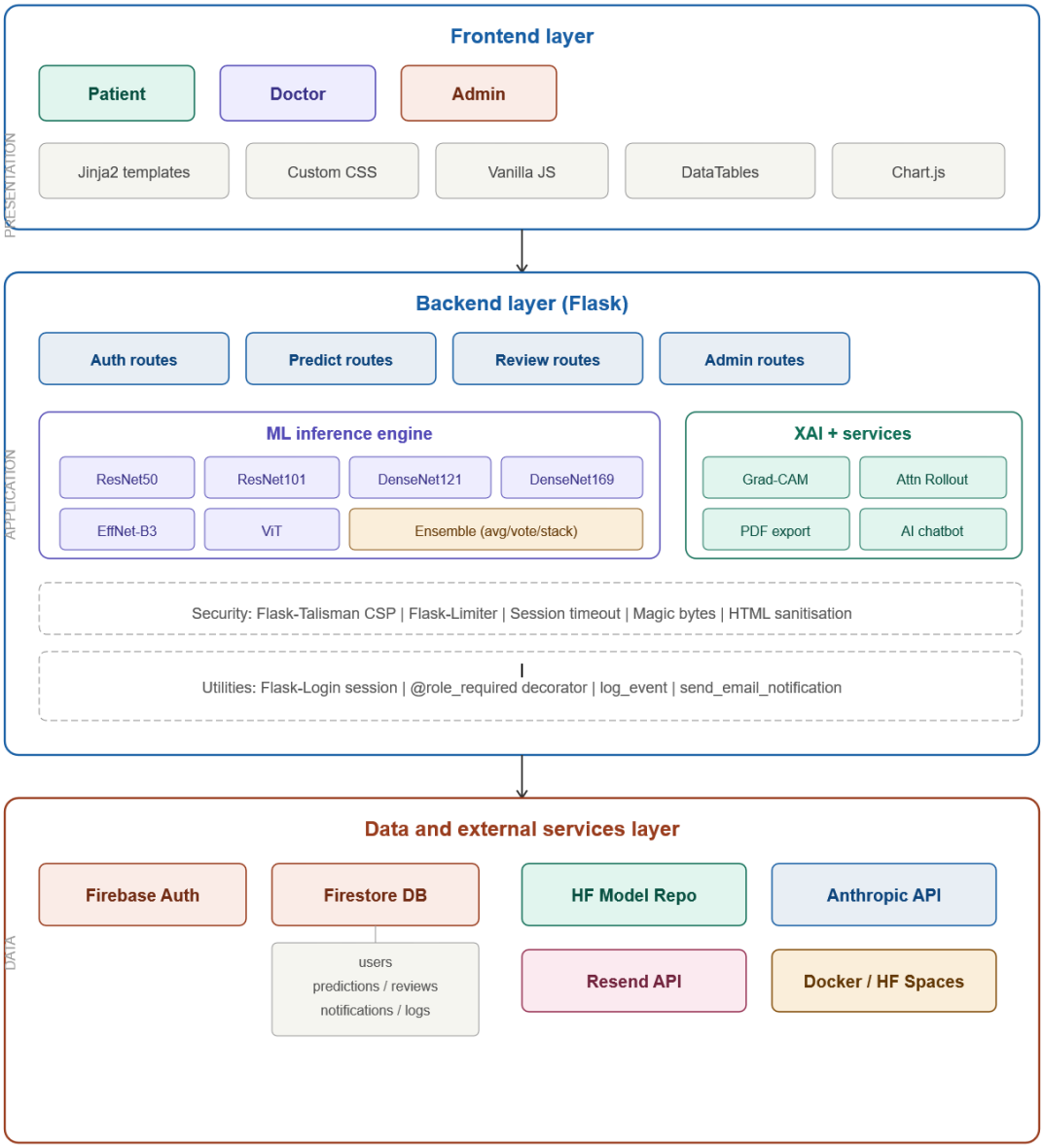

Figure 3.14 System Architecture Diagram

The application was structured into three layers: frontend, backend and data/services. The frontend used Flask with Jinja2 templates, custom-made CSS (using the DM Sans typography and a CSS variables-based design system) and vanilla JavaScript. Users interacted with role-based dashboards in Jinja2 templates, tailored for each user role.

The back-end handled user authentication (using Firebase Admin SDK), MRI scan prediction, review process management, notifications, chatbot, generation of PDFs and monitoring. Inference was done by a set of six individually loaded models (ResNet50, ResNet101, DenseNet121, DenseNet169, EfficientNet-B3, and ViT) and the ensemble model. The model

weights were hosted in a separate Hugging Face model repository and loaded on the fly using the huggingface_hub library. Grad-CAM objects were pre-emptively stored to avoid hook registration on each prediction, and the ViT processor was downloaded at start-up to remove per-prediction download times.

Firebase handled authentication (email/password, Google OAuth), Firestore (NoSQL database) for storing users, predictions, reviews, notifications and audit logs, and the Firebase Admin SDK on the server for administrative tasks. The app was Dockerised and hosted on Hugging Face Spaces with 10 environment secrets for API keys and credentials.

3.9 Implementation Issues and Challenges

There were some challenges faced in this project.

1. Dataset limitation: Despite having 15,120 training images, the dataset was sourced from one hospital (HPUPM). The validation set (38 images) and test set (34 images) were especially small. This risked overfitting and reduced trust in generalisation to new data. Data augmentations (flips, rotations, jittering) were used to reduce this effect.
2. Integration complexity: Syncing the Flask web server, six PyTorch models, Firebase services and several third-party APIs (Anthropic, Resend) was complex. Real-time explanations (heatmaps) were especially difficult to implement, with Grad-CAM requiring hooking convolutional layers and Attention Rollout requiring access to attention weights.
3. Deployment constraints: The Hugging Face Spaces free plan had limitations: no GPU (only CPU inference), temporary storage (Grad-CAM images lost when the container restarted), blocked outbound SMTP (hence the use of Resend API for email notifications), and cold starts (5-10 minutes to download model weights, ~697 MB).
4. Firebase service account key incident: The Firebase service account key was accidentally deleted during development and had to be recreated via Google Cloud Console with certain IAM roles (Firebase Admin SDK Administrator Service Agent and Cloud Datastore User).
5. CSRF incompatibility: CSRF prevention was turned off because it was incompatible with the Firebase JavaScript SDK authentication process (which posts ID tokens in JSON). This issue was resolved by SameSite=Lax cookies and Firebase ID token verification in every session login.

Chapter 4
System Design

This chapter discuss the design of the NeuroScan system in detail, including the system architecture and component specifications, database schema, machine learning pipeline design, and component interaction operations.

4.1 System Block Diagram

The design of the NeuroScan system was a three-layer modular architecture, with the frontend layer, the backend layer, and the data and external services layer. These layers performed different functions and interacted with each other through interfaces.

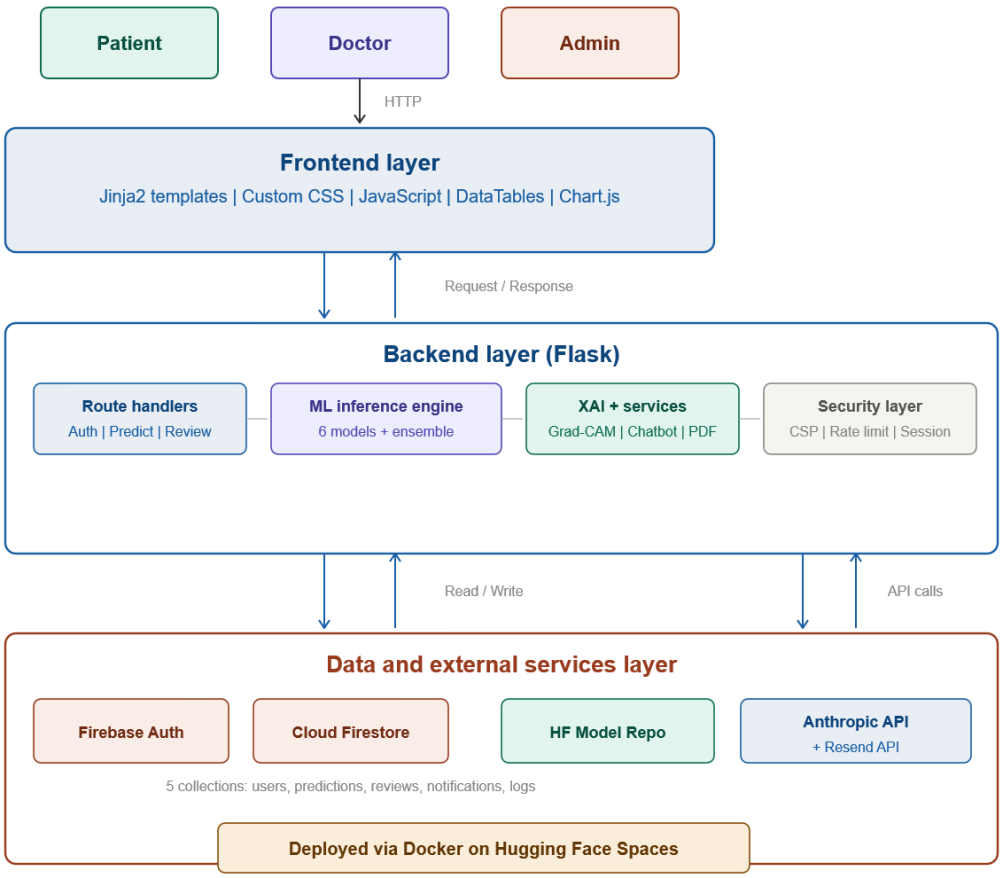

Figure 4.1 System Block Diagram

As shown in Figure 4.1, the frontend layer was responsible for the user interface and client-side interactions. The frontend layer was made up of Jinja2 templates that dynamically generated HTML content, a custom CSS stylesheet with a DM Sans-based design system using

CSS variables for themes, vanilla JavaScript for interacting with the DOM and making asynchronous HTTP requests, the DataTables.js library for dynamic pagination and search functionality on dashboards, and the Chart.js library for rendering bar and doughnut charts on the admin dashboard.

The backend layer was the heart of the system, built with the Flask framework. The core system was responsible for four main functions: (1) user authentication and session management using Firebase Admin SDK and Flask-Login, (2) MRI scan processing and deep learning model inference by the ML inference engine, (3) business logic for the review workflow, sending notifications, generating PDFs and interacting with the AI chatbot, and (4) security implementation through Flask-Talisman, Flask-Limiter, session timeouts, and input validation.

The data and external services layer offered data persistence and third party services. Firebase Authentication provided user authentication for email/password and Google OAuth. The NoSQL database (Cloud Firestore) contained five collections: users, predictions, reviews, notifications, and logs. The model weights (7 .pth files, totalling ~697 MB) were stored on the Hugging Face Model Repository and downloaded on-the-fly. The AI chatbot utilised the Anthropic API's Claude Haiku model, and the Resend API was used for sending notifications. The whole project was containerised with Docker and hosted on Hugging Face Spaces.

4.2 System Components Specifications

This section outlines the configurations of the main components of the system.

4.2.1 Application Structure

The Flask application was structured with the application factory pattern. The project structure was as follows:

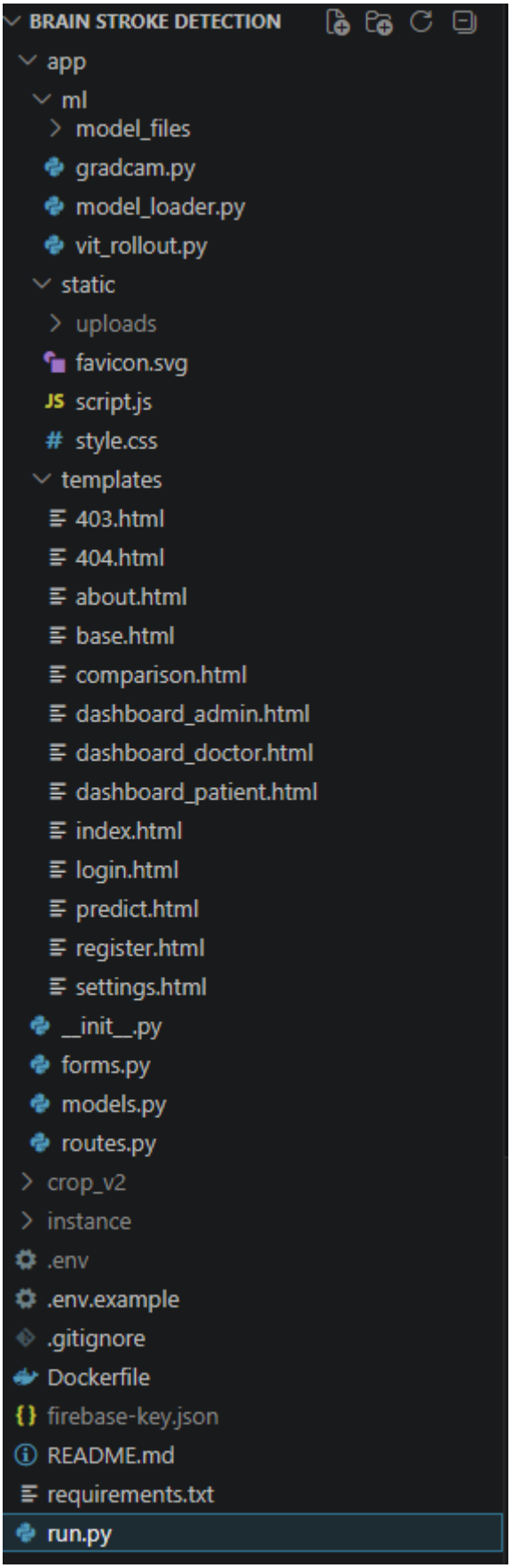

Figure 4.2 Project Directory Structure

The components of the app are summarised in Table 4.1.

Table 4.1 Application Structure Components

Component	Description
run.py	Entry point that invokes the application factory and starts the Flask development server.
app/__init__.py	Application factory function (create_app). Initialises Flask, Firebase Admin SDK, Flask-Login, Flask-Talisman, Flask-Limiter, session configuration, and registers the blueprint.
app/routes.py	Contains all route handlers: authentication (login, register, session_login, session_register), prediction, review, dashboards (patient, doctor, admin), notifications, chatbot, PDF export, settings, and admin operations. Also contains the log_event and send_email_notification utility functions.
app/models.py	Defines the User class implementing Flask-Login's UserMixin. Provides get_by_id(uid) class method to retrieve user data from Firestore and construct User objects with id, email, name, role, and provider attributes.
app/forms.py	Defines the UploadForm using Flask-WTF with a FileField for MRI image upload and a SelectField for model selection (7 options: 6 individual models + ensemble).
app/ml/model_loader.py	Handles model weight downloading from HuggingFace (ensure_models_downloaded), model loading (load_model), single-model prediction (predict_stroke_risk), and ensemble prediction (predict_ensemble) with ThreadPoolExecutor for parallel inference.
app/ml/gradcam.py	Implements Grad-CAM for CNN-based models. Registers forward and backward hooks on the target convolutional layer, computes gradient-weighted activation maps, and generates colour-mapped heatmap overlays on the original MRI image.
app/ml/vit_rollout.py	Implements Attention Rollout for the Vision Transformer. Extracts attention weights from all transformer layers, recursively multiplies attention matrices, and generates a patch-level attention heatmap.
app/templates/	Contains 13 Jinja2 HTML templates: base.html (layout with navbar, session timeout, chatbot panel, notification bell), index.html (home), login.html, register.html, predict.html, dashboard_patient.html, dashboard_doctor.html, dashboard_admin.html, settings.html, about.html, comparison.html, 403.html, and 404.html.

app/static/	Contains style.css (custom CSS with CSS variables design system), script.js (client-side JavaScript), favicon.svg (brain icon), and the uploads/ directory for temporary storage of uploaded MRI scans and generated heatmap images.

4.2.2 User Model

The User model (app/models.py) was an implementation of the Flask-Login's UserMixin interface for session-based authentication. A User object had five attributes: id (Firebase UID), email, name (user display name), role (patient, doctor, or admin), and provider (email or google, to decide if the user can reset their password). The get_by_id(uid) class method read the user document from Firestore and created the User. It was also used as the user_loader callback for Flask-Login to ensure the user was fetched from Firestore for each authenticated request.

4.2.3 Route Design

All route handlers were implemented in a single Flask blueprint (app/routes.py). Routes were grouped by their functionality and access was controlled using decorators. Table 4.2 lists the groups and their associated access controls.

Table 4.2 Route Groups, Access Controls, and Rate Limits

Route Group	Endpoint(s)	Access Control	Rate Limit
Authentication	/login, /register, /session_login, /session_register, /logout	Public (login, register) / @login_required (logout)	5/min (session_login)
General Pages	/, /home, /about, /comparison	@login_required (home) / Public (about, comparison)	None
Prediction	/predict	@login_required	10/min
Patient Dashboard	/dashboard/patient	@role_required("patient")	None
Doctor Dashboard	/dashboard/doctor	@role_required("doctor")	None
Admin Dashboard	/dashboard/admin	@role_required("admin")	None
Review	/review/<prediction_id>	@role_required("doctor")	None
PDF Export	/export-pdf	@role_required("patient")	None

AI Chatbot	/chat	@login_required	20/min
Notifications	/notification/<id>/read, /notifications/read-all	@login_required	None
Settings	/settings, /update-display-name, /update-password	@login_required	None
Admin Operations	/admin/update_role/<id>, /admin/delete_user/<id>, /admin/delete_prediction/<id>	@role_required("admin")	None

The @role_required decorator was a custom decorator that compared the logged-in user's role to the allowed roles. It raised an HTTP 403 Forbidden error if the user's role did not match. This decorator was used in combination with Flask-Login's @login_required decorator to provide authentication and authorisation.

4.3 Database Schema Design

The data was stored in a NoSQL document database called Cloud Firestore. Firestore's data model is based on collections of documents, with each document being a collection of key-value pairs. A total of five collections were defined.

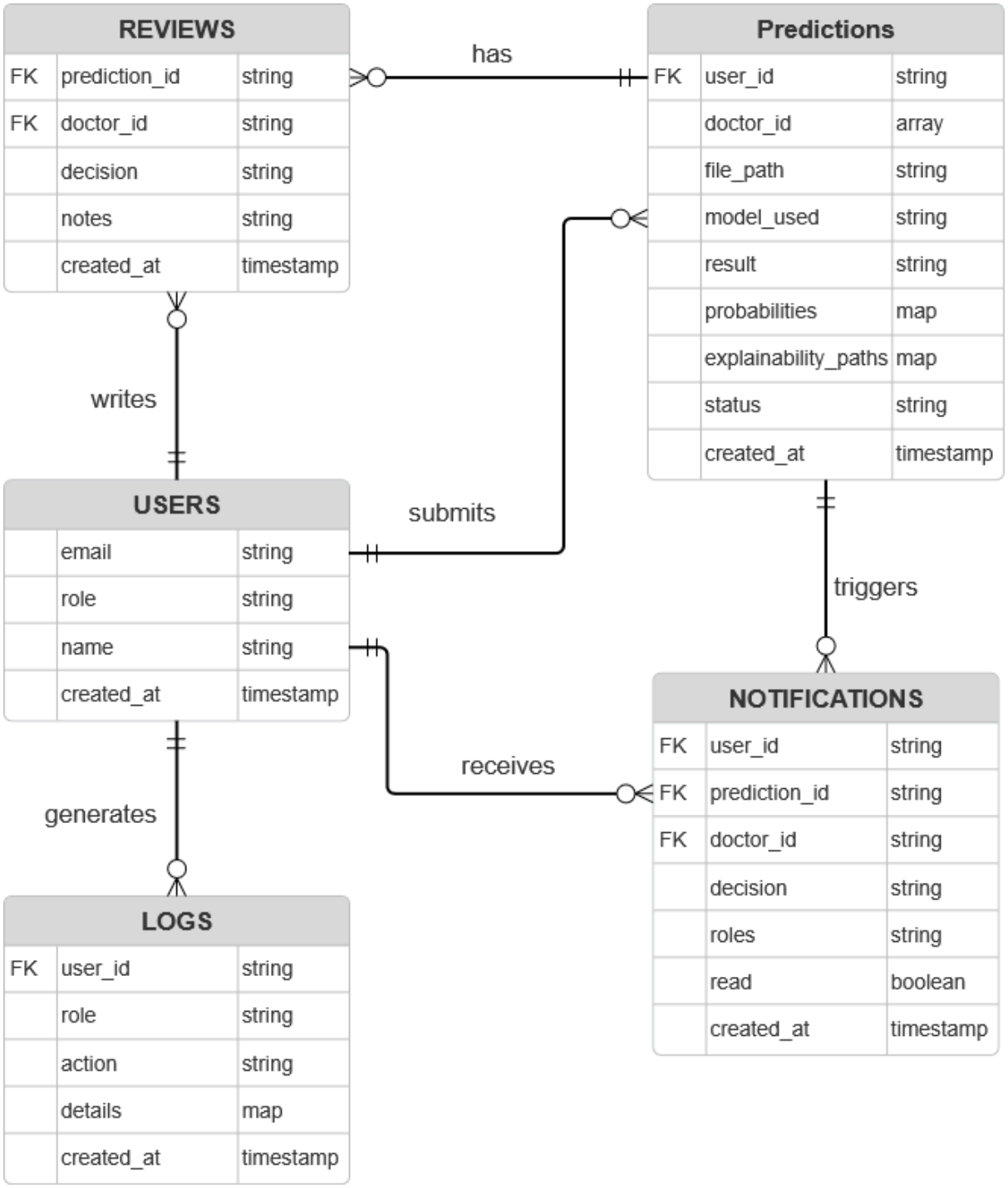

Figure 4.3 Firestore Database Schema (ERD)

4.3.1 Users Collection

The users collection held user profile data. The documents were indexed by the Firebase Authentication UID.

Table 4.3 Users Collection Schema

Field	Type	Description

email	string	User’s email address (from Firebase Auth)
role	string	User role: "patient", "doctor", or "admin"
name	string	Display name (optional, set via settings page)
created_at	timestamp	Account creation timestamp (Firestore SERVER_TIMESTAMP)

4.3.2 Predictions Collection

The predictions collection stored all MRI scans predictions. The documents were automatically generated with a Firestore ID.

Table 4.4 Predictions Collection Schema

Field	Type	Description
user_id	string	Firebase UID of the user who submitted the prediction
doctor_ids	array	List of doctor UIDs who reviewed the prediction (initially empty)
file_path	string	Filename of the uploaded MRI scan in app/static/uploads/
model_used	string	Selected model name (e.g. "resnet50", "ensemble")
result	string	Predicted class label: "Clot" or "No Clot"
probabilities	map	Dictionary of class probabilities (e.g. {"Clot": 0.92, "No Clot": 0.08})
explainability_paths	map	Dictionary containing "gradcam" and "original" image file paths for XAI heatmaps
status	string	Review status: "pending_review" or "reviewed"
created_at	timestamp	Prediction submission timestamp

4.3.3 Reviews Collection

The reviews collection stored reviews by doctors. The prediction_id field associated the review with a prediction.

Table 4.5 Reviews Collection Schema

Field	Type	Description

prediction_id	string	Firestore document ID of the associated prediction
doctor_id	string	Firebase UID of the reviewing doctor
decision	string	Doctor’s decision: "agree" or "disagree"
notes	string	Optional clinical remarks from the doctor (HTML-sanitised before email inclusion)
created_at	timestamp	Review submission timestamp

4.3.4 Notifications Collection

The notifications collection stored in-app notifications for the patient when a doctor reviewed a patient.

Table 4.6 Notifications Collection Schema

Field	Type	Description
user_id	string	Firebase UID of the patient to be notified
prediction_id	string	Firestore document ID of the reviewed prediction
doctor_id	string	Firebase UID of the doctor who submitted the review
decision	string	Doctor’s decision: "agree" or "disagree"
notes	string	Doctor’s clinical remarks (if any)
read	boolean	Whether the patient has read the notification (default: false)
created_at	timestamp	Notification creation timestamp

4.3.5 Logs Collection

The logs collection contained audit logs of all major user interactions with the system.

Table 4.7 Logs Collection Schema

Field	Type	Description
user_id	string	Firebase UID of the user who performed the action
role	string	Role of the user at the time of the action

action	string	Action type (e.g. "login", "register", "predict", "review", "update_role", "delete_user", "delete_prediction", "update_name")
details	map	Additional context specific to the action (e.g. {"model": "ensemble", "result": "Clot"})
created_at	timestamp	Event timestamp

The connections between the collections used document IDs for references, rather than foreign key constraints (Firestore is a NoSQL database). The user_id in predictions, notifications and logs collection was the users collection document ID. The prediction_id field in reviews and notifications referenced the predictions collection document ID. The doctor_id field in reviews and notifications referred to the users collection ID. These links were implemented in the application rather than the database.

4.4 Machine Learning Pipeline Design

The machine learning inference pipeline was capable of making predictions using individual models or an ensemble of models. The pipeline had four components: model weight storage, model loading, inference, and explainability.

4.4.1 Model Weight Management

The model weights (7 .pth files, totalling about 697 MB) were hosted separately in another Hugging Face model repository (DancinggDogg/NeuroScan-models) instead of in the application's Git repository. The reason for this decision is that Git is not an effective way to store large binary files, and Hugging Face Spaces has a repository size limit. At application startup, the ensure_models_downloaded() function (located in model_loader.py) verified that the model files were present in the local app/ml/model_files/ directory. If a file was not present it was downloaded from the Hugging Face model repository using the huggingface_hub library. This allowed zero-downtime deployments and a small application repository.

4.4.2 Model Loading

The load_model() function, which took model name and checkpoint filename as arguments, loaded all six base models. It built the model architecture (ResNet50, ResNet101, DenseNet121, DenseNet169, EfficientNet-B3, or ViT), replaced the final classification layer with two output classes (clot, no clot), loaded the model's state dictionary from the .pth checkpoint, and put the

model in evaluation mode. Models were loaded at startup and stored in memory to eliminate the load times.

Two other optimisations were performed at startup:

- HuggingFace ViT image processor caching: The HuggingFace ViT image processor (google/vit-base-patch16-224-in21k) was downloaded and stored at startup to avoid the download on each prediction request.
- Grad-CAM object caching: GradCAM objects for the five CNN models were created at startup and added to a GRAD_CAM_CACHE dictionary. This eliminated the need to register hooks on every prediction.

4.4.3 Inference Pipeline

For individual model predictions, the predict_stroke_risk() function took in the image path and model name, applied some pre-processing (resize to 224x224, normalise with ImageNet statistics), performed forward inference, applied softmax to get class probabilities and produced the XAI heatmap. For CNN models, Grad-CAM was calculated on the last convolutional layer. For the ViT, Attention Rollout was applied with the cached processor.

To calculate the ensemble prediction, the predict_ensemble() function ran the six models in parallel using Python's ThreadPoolExecutor with max_workers set to the number of models. This was possible because PyTorch releases the GIL (Global Interpreter Lock) when it operates on tensors, enabling parallel processing on the CPU. The pooling method was simple averaging of the individual probabilities. For the deployed system, simple averaging was used as the ensemble method, based on the results in Chapter 6.

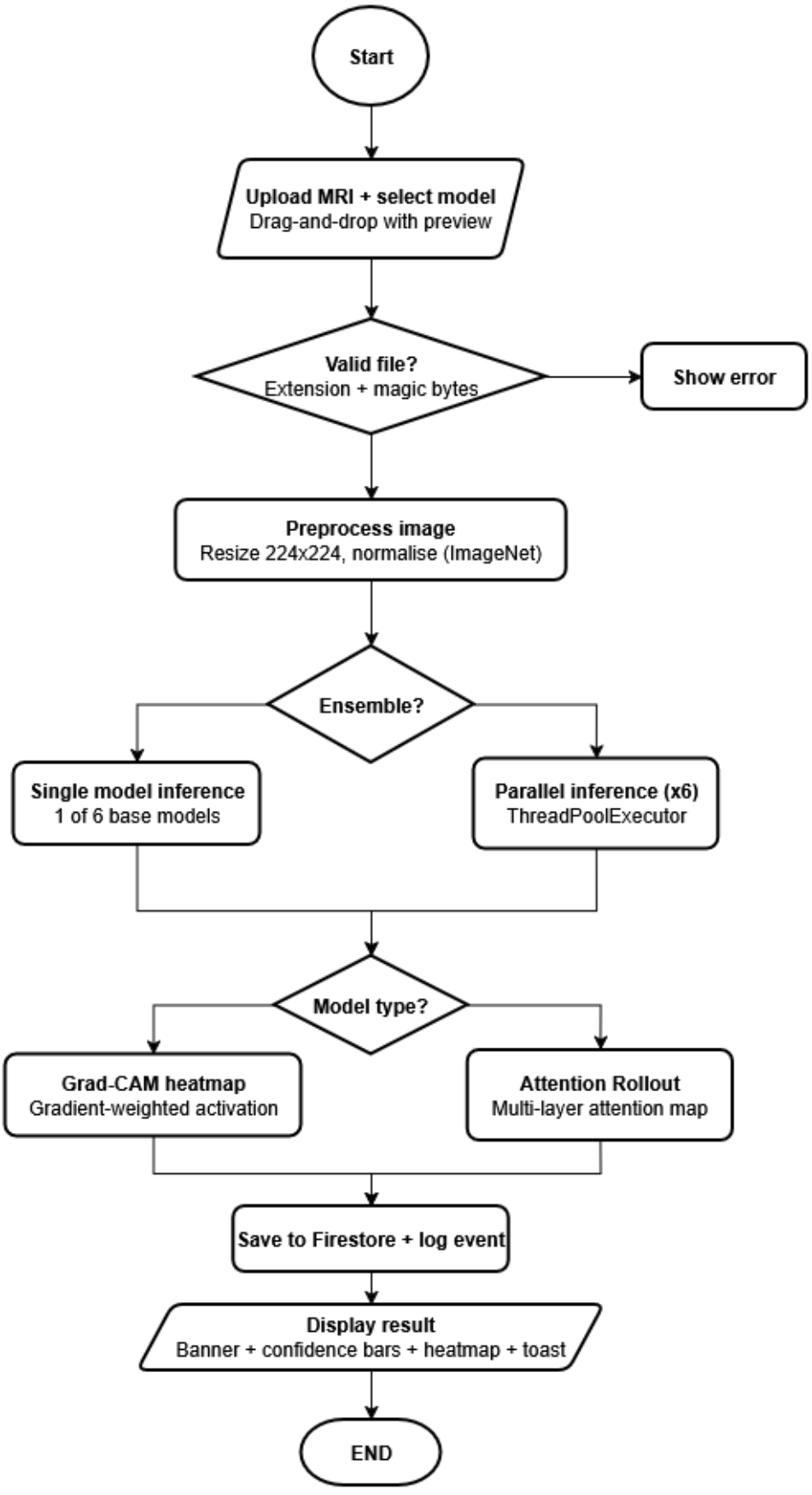

Figure 4.4 ML Inference Pipeline Flowchart

4.4.4 Explainability Generation

Grad-CAM was used to explain the five CNN models (ResNet50, ResNet101, DenseNet121, DenseNet169, EfficientNet-B3). The code in gradcam.py registered forward hooks to get the activation maps and backward hooks to get the gradients at the chosen convolutional layer. Following a forward and backward pass, the gradient-weighted activation map was calculated by averaging the gradients across the spatial dimensions and weighting the activation channels with these gradients. The heat map was re-sampled to the original image size by bilinear interpolation, normalised to [0, 1], and then overlaid on the original MRI image with a colour map. The result was saved in the app/static/uploads/ folder.

Attention Rollout was used with the Vision Transformer. The code in vit_rollout.py obtained the attention weights for the 12 attention heads in the 12 encoder layers of the transformer. Attention maps were averaged over heads within each layer, then an identity matrix was added to account for the residual connection, re-normalised, and then multiplied with the attention map of the previous layer to obtain the cumulative attention map. The attention corresponding to the first token (the [CLS] token) was taken, reshaped to the grid of patches (14×14 for 224×224 image size with 16×16 patches) and then upsampled to the original resolution. This heatmap was then visualised on the original MRI image and written to the uploads folder.

4.5 Security Design

Security was an important aspect of the project, as it involved medical data. A range of security measures were put in place.

Table 4.8 shows the security features designed for this project.

Table 4.8 Security Measures

Security Measure	Design and Implementation
Flask-Talisman CSP	Content Security Policy headers were configured to restrict the sources from which scripts, styles, images, fonts, and frames could be loaded. Inline scripts and styles were permitted (‘unsafe-inline’) due to Jinja2 template requirements. CSP mitigated cross-site scripting (XSS) and data injection attacks.
Flask-Limiter	Rate limiting was applied to three endpoints: /session_login (5 requests per minute), /predict (10 requests per minute), and /chat (20 requests per minute). This prevented brute-force login attempts, prediction abuse, and chatbot API cost overruns.

Session timeout	User sessions were configured with a 30-minute inactivity timeout. A countdown warning modal appeared after 29 minutes with a 60-second timer. Users could click “Stay logged in” to reset the session via an AJAX ping to /ping. If the countdown reached zero, the user was automatically logged out.
Magic byte validation	Uploaded files were validated not only by file extension but also by their magic bytes (file signature). The system checked the first few bytes of the uploaded file against known signatures for JPEG (0xFFD8FF) and PNG (0x89504E47) formats. Files with mismatched extensions and magic bytes were rejected.
Role-based access control	A custom @role_required decorator checked the authenticated user’s role against the permitted roles for each route. Unauthorised access returned a 403 Forbidden response. This was layered on top of Flask-Login’s @login_required decorator.
Prediction ownership	Patient dashboard queries and PDF export queries filtered predictions by user_id == current_user.id, ensuring patients could only view and export their own prediction records.
HTML sanitisation	Doctor’s clinical notes were HTML-escaped using markupsafe.escape() before injection into email notification HTML templates, preventing HTML injection attacks in emails.
Blocked email domains	Registration was blocked for disposable and temporary email domains (e.g. mailinator.com, tempmail.com) through both client-side and server-side (defence in depth) domain validation.
HttpOnly cookies	Session cookies were configured with HttpOnly and SameSite=Lax attributes to prevent client-side JavaScript access and mitigate cross-site request forgery (CSRF) risks.

It's important to note that CSRF protection was deliberately turned off in this project because it would not work with the Firebase JavaScript SDK authentication. CSRF cannot be protected by a token when Firebase Authentication communicates ID tokens to the server via client-side JSON POST requests. This was compensated by the SameSite=Lax cookie policy and Firebase ID token verification on session login.

4.6 System Components Interaction Operations

This section will provide an overview of how the core components interact.

4.6.1 Authentication Flow

The login process involved interactions between the client-side Firebase JavaScript SDK and the Flask server. For email/password login, the Firebase JS SDK logged in the user and received an ID token. This token was passed to the /session_login endpoint where the Flask server verified it with the Firebase Admin SDK, fetched the user document from Firestore and stored it in a Flask-Login session. For Google OAuth, the process was the same, except that a role selection modal was presented for first-time Google users, to select a patient or doctor role, before the Flask session was created.

4.6.2 Prediction Flow

The MRI submission included file validation (extension and magic bytes). The Flask backend called the predict function of the model (predict_ensemble for ensemble predictions). The model predicted the class, probabilities and the XAI heatmap image file path. The result was then stored in Firestore with a status of "pending_review", and the result was logged to the logs collection. The result page showed the prediction banner (green for no clot, red for clot), confidence bars with an animation and the XAI heatmap image with a button to download the image.

4.6.3 Review and Notification Flow

When a doctor added a review, the Flask backend did the following: (1) created a review document in the reviews collection, (2) updated the prediction status from “pending_review” to “reviewed”, (3) created an in-app notification document in the notifications collection, (4) logged the review to the logs collection and (5) sent an email notification to the patient using the Resend API. The patient then received a badge on the bell icon in the navbar and an email with the doctor's decision and notes.

4.6.4 Admin Operations Flow

The admin also performed role updates and user/prediction deletion. When the admin changed a user's role, the Firestore user was updated and the action was logged. When an admin deleted a user, the system cascaded the deletion: deleted the user from Firebase Authentication, then the user document in Firestore, along with all the user’s prediction documents and the reviews associated with those predictions. This ensured application-level referential integrity.

4.7 Concluding Remark

This chapter has outlined the design of the NeuroScan system. The three-layered architecture with distinct user interface, business, and data layers allowed for parallel development and maintenance. The Firestore database structure was tailored to support the full clinical workflow from prediction to review, with a full audit history. The machine learning pipeline used parallel inference, caching at startup and downloading models during runtime to balance performance and deployment considerations. The system design incorporated several security features including rate limiting, content security policies, session management, input validation and role-based access control. The following chapter provides details of the implementation.

Chapter 5

System Implementation

This chapter details the implementation of the NeuroScan system, including the hardware and software setup, the configuration, the operation of the system including screenshots, and any implementation issues and challenges faced.

5.1 Hardware Setup

This project involved two hardware environments: the local development environment and the cloud training environment.

1. Local development environment: Web application development, testing and build for deployment was done on an Acer Nitro AN515-46 laptop with AMD Ryzen 7 6800H processor (3.20 GHz), 16 GB RAM, NVIDIA GeForce RTX 3050 Laptop GPU, Windows 11. Coding, debugging and version control (Git) was done in the integrated development environment (IDE) Visual Studio Code.
2. Cloud training environment: Models were trained in Google Colab, which made available NVIDIA A100 (40 GB HBM2e) and NVIDIA L4 (24 GB GDDR6) GPUs for training. Google Colab was used as the local laptop GPU didn't have enough VRAM to train deep learning models on the MRI dataset. Training Jupyter notebooks were stored on Google Drive for version management.

For deployment environment, the model was deployed using Hugging Face Spaces, which provided a shared CPU with a constrained memory. The free plan did not provide access to GPU, hence inference was done on the CPU. The hosting environment was containerised using Docker and the application was served on port 7860.

5.2 Software Setup

The software environment included the installation and setup of all necessary tools and libraries for the development of both the model and web application.

5.2.1 Python Environment

Programming for both stages of the project was done with Python 3.11. A local virtual environment (.venv) was set up using the venv module to separate the project dependencies from the system Python installation.

5.2.2 Dependencies Installation

The project dependencies were listed in a requirements.txt file and grouped. The dependencies were installed using pip. The main dependency groups are summarised in Table 5.1.

Table 5.1 Dependency Groups

Group	Key Packages	Purpose
Core ML	torch, torchvision, Pillow, timm, opencv-python-headless	Deep learning framework, pre-trained models, image processing, EfficientNet loading, OpenCV for image manipulation
Web (Flask stack)	Flask, Werkzeug, Jinja2, python-dotenv	Web framework, WSGI utility, template engine, environment variable loading
Security	Flask-Login, Flask-Limiter, Flask-Talisman, Flask-WTF, Flask-Bcrypt, cryptography	Session management, rate limiting, CSP headers, form handling, password hashing, cryptographic utilities
Firebase	firebase-admin	Server-side Firebase Authentication, Firestore database, and administrative operations
ML Utilities	numpy, matplotlib, transformers	Numerical operations, visualisation, HuggingFace ViT model and processor
AI Chatbot	anthropic	Anthropic API SDK for Claude Haiku integration
PDF Export	reportlab	PDF document generation for patient prediction reports
Email	resend	Resend API for email notification delivery

5.2.3 Firebase Setup

Firebase was configured in the Firebase Console (console.firebase.google.com). The following steps were performed:

- Created a new project "NeuroScan".

- Configured Firebase Authentication with two providers: Email/Password and Google.
- Enabled a production mode Cloud Firestore database.
- Created a service account key (JSON) for Firebase in the Google Cloud Console with two IAM roles: Firebase Admin SDK Administrator Service Agent and Cloud Datastore User.
- Stored the service account key in firebase-key.json for development.
- For deployment, the service account key, as a JSON string, was stored as the FIREBASE_KEY_JSON environment secret on Hugging Face Spaces.

5.2.4 Anthropic API Setup

A console.anthropic.com account was set up. An API key was created and saved as the ANTHROPIC_API_KEY secret. The AI chatbot was set up to use claude-haiku-4-5-20251001 model with a stroke-focused system prompt that limited chatbot responses to stroke medical information and how to use the system.

5.2.5 Resend API Setup

An account was set up with Resend (resend.com). An API key was created and added to the RESEND_API_KEY environment secret. Because of Resend's free tier restrictions, the emails were sent from Resend's default address and the RESEND_TO_EMAIL environment variable was set to send all notification emails to the admin's email address with the patient's email address in the subject. This was a known way to bypass the Resend free tier restriction that you can only send emails to arbitrary email addresses if you verify the domain.

5.3 Setting and Configuration

5.3.1 Application Factory Configuration

The Flask application was set up using the application factory pattern in app/__init__.py. The important configuration settings are summarised in Table 5.2

Table 5.2 Flask Application Configuration

Configuration	Value and Purpose
SECRET_KEY	Loaded from SECRET_KEY environment variable. Used for session signing and CSRF token generation.

PERMANENT_SESSION_LIFETIME	30 minutes. Sessions expire after 30 minutes of inactivity.
SESSION_REFRESH_EACH_REQUEST	True. Session lifetime resets on every request, ensuring the timeout is based on inactivity rather than absolute time.
MAX_CONTENT_LENGTH	16 MB. Maximum upload file size to prevent oversized file uploads.
WTF_CSRF_ENABLED	False. CSRF protection disabled due to Firebase JS SDK incompatibility (documented limitation).

5.3.2 Content Security Policy Configuration

Flask-Talisman was used to implement a Content Security Policy (CSP) which rigorously controls the sources of resources loaded. The CSP directives allowed resources to be loaded from trusted CDNs (jQuery, DataTables, Chart.js, Google Fonts, Firebase SDKs) and disallowed all other origins. Inline scripts and styles were allowed ('unsafe-inline') as the Jinja2 templating engine was used to inject dynamic JavaScript and CSS into HTML templates.

5.3.3 Hugging Face Spaces Configuration

The web app was deployed on Hugging Face Spaces using Docker SDK. The Spaces settings were specified in the README.md metadata

The Hugging Face Spaces environment defined ten environment secrets, as listed in Table 5.3.

Table 5.3 Hugging Face Spaces Environment Secrets

Secret Name	**Purpose**
SECRET_KEY	Flask session signing secret
FIREBASE_API_KEY	Firebase Web API key for client-side SDK initialisation
FIREBASE_AUTH_DOMAIN	Firebase Authentication domain (projectid.firebaseapp.com)
FIREBASE_PROJECT_ID	Firebase project identifier
FIREBASE_KEY_JSON	Full Firebase service account key JSON (base64 or raw JSON string)
ANTHROPIC_API_KEY	Anthropic API key for Claude Haiku chatbot
RESEND_API_KEY	Resend API key for email notifications

RESEND_TO_EMAIL	Admin email address for receiving notification emails (free tier workaround)
HF_TOKEN	Hugging Face token for downloading model weights from the private model repository
FLASK_DEBUG	Debug mode flag (set to false in production)

5.3.4 Docker Configuration

The Dockerfile was used to specify the container environment. It was based on Python 3.11 slim, installed system packages needed for OpenCV (libglib2.0-0, libgl1), installed Python packages from requirements.txt, copied application code, exposed port 7860 (Hugging Face Spaces default) and launched the Flask development server (run.py).

5.3.5 Model Weight Repository

The model weights were kept in a separate model repository on Hugging Face (DancinggDogg/NeuroScan-models), instead of the app's Git repository. This repository contained seven checkpoint files (with the extension .pth) totalling 697 MB. On start, the ensure_models_downloaded() function checked if each file existed and downloaded any missing files with the huggingface_hub library using the environment secret HF_TOKEN for authentication.

5.4 System Operation

This section provides screenshots of the important pages and features of the NeuroScan web application.

5.4.1 Login Page

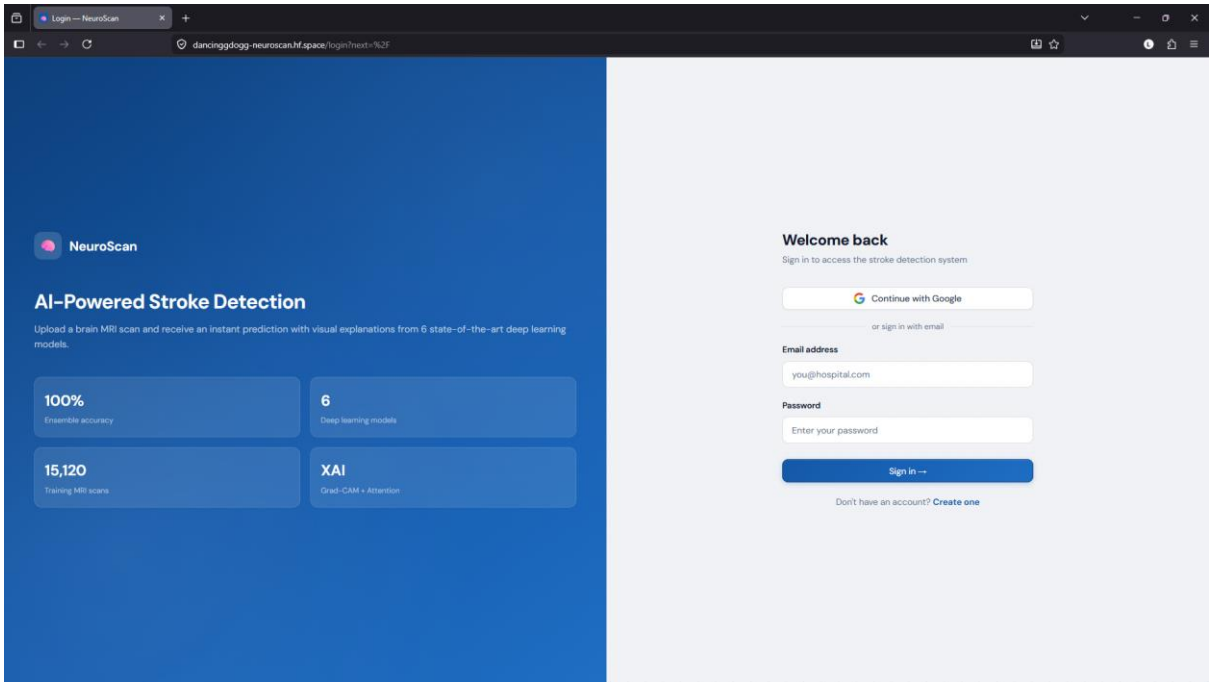

Figure 5.1 Login Page (Split-Panel Layout)

The login page had a two-panel layout with the NeuroScan logo and tagline on the left, and the login form on the right. Authentication was via email and password or Google OAuth. This design aimed to make a good first impression.

5.4.2 Register Page

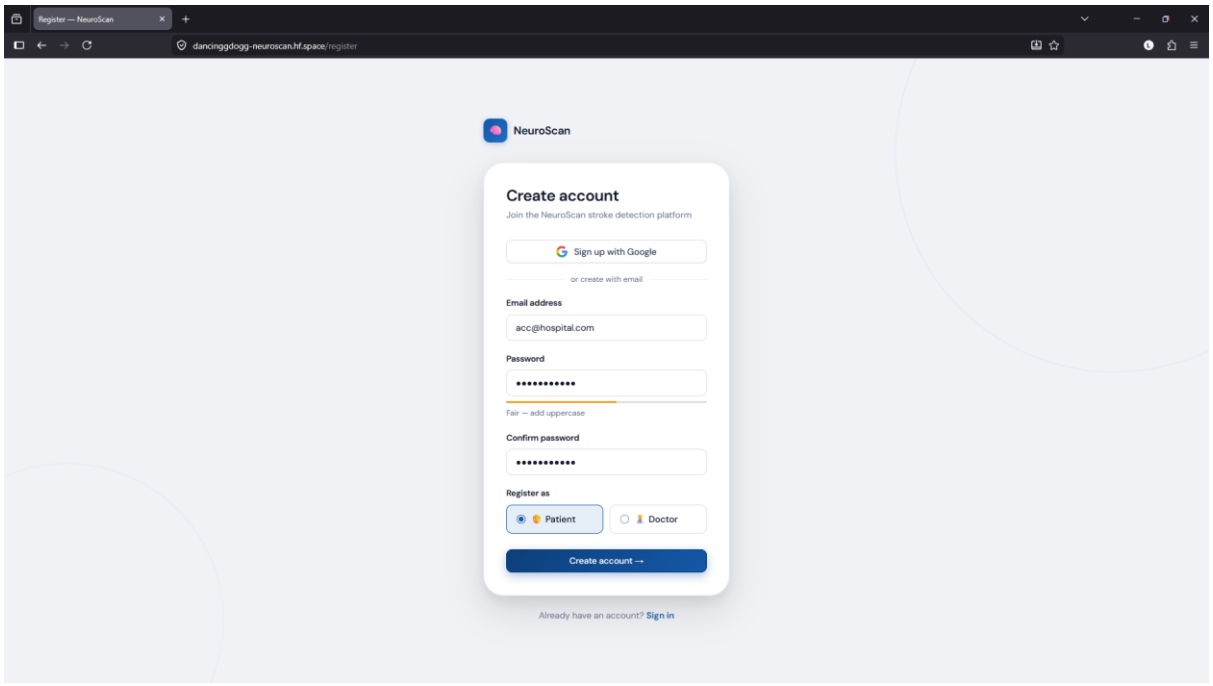

Figure 5.2 Register Page with Role Selection

The register page enabled users to sign up by entering their email, password and choosing a user type (patient or doctor). The password strength bar gave real-time feedback of the password strength. Disposable email domain filtering ensured valid email addresses.

5.4.3 Google OAuth Role Selection

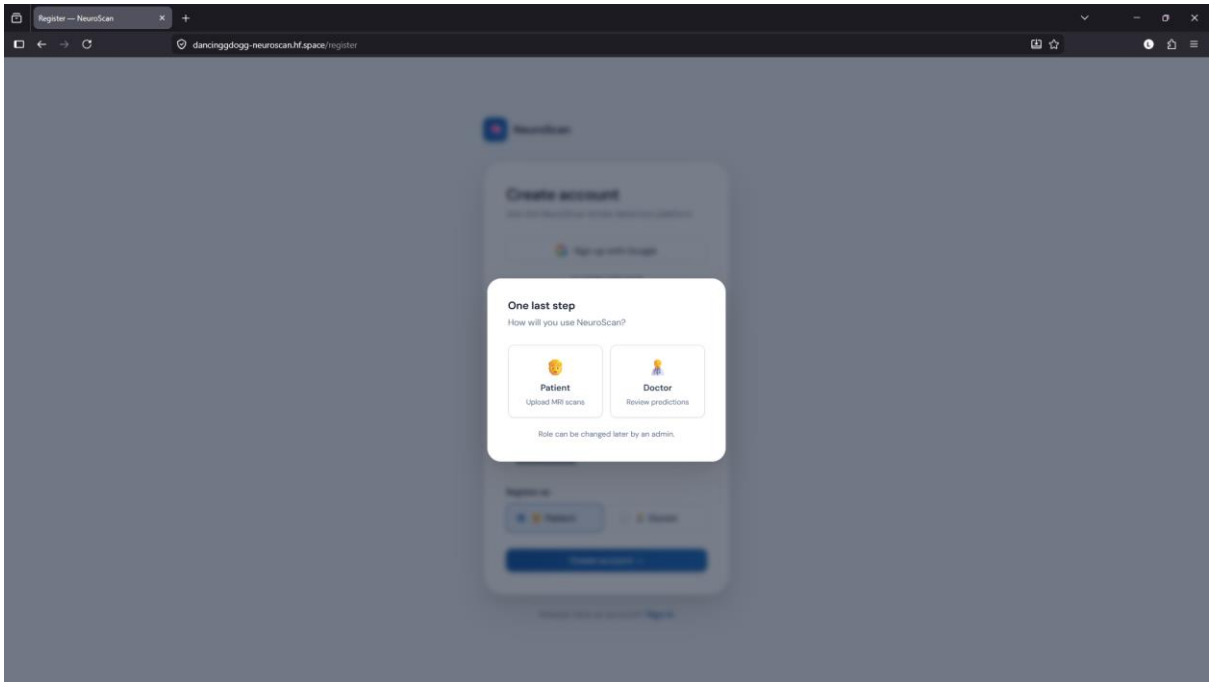

Figure 5.3 Google OAuth Role Selection Modal

The first time a user logged in with Google OAuth, a modal was displayed allowing the user to select a role (patient or doctor). This was due to the lack of role data in Google OAuth, so the system required this data to finish registration.

5.4.4 Home Page

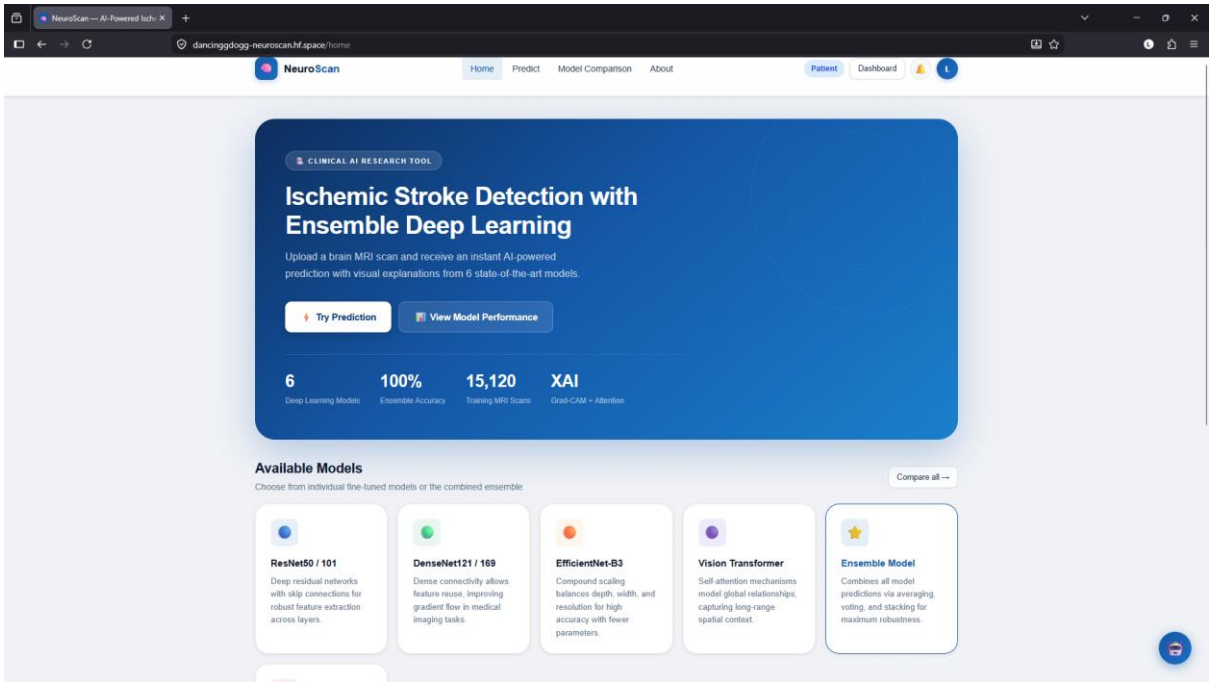

Figure 5.4 Home Page with Hero Section and Animated Statistics (1)

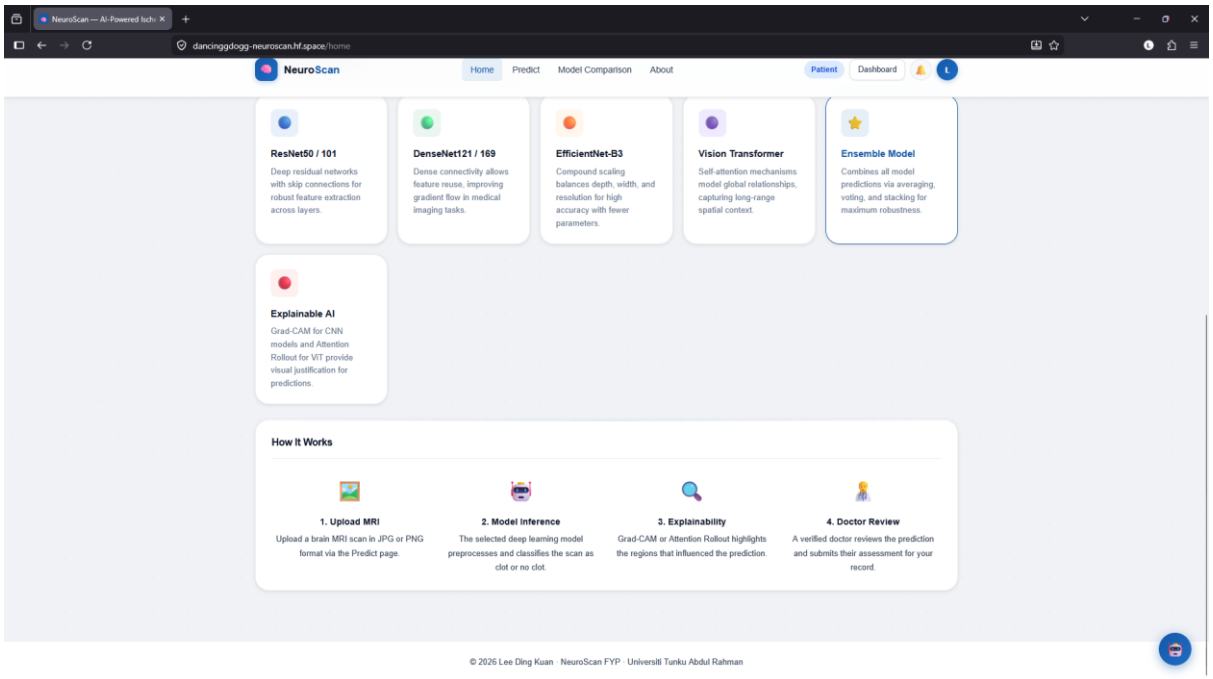

Figure 5.5 Home Page with Hero Section and Animated Statistics (2)

The home page included a hero section with a gradient background and rings that animated on scroll. Three count-up values (Models, Accuracy, Dataset size) animated on scroll using IntersectionObserver. This was followed by feature cards listing the main features of the system.

5.4.5 About Page

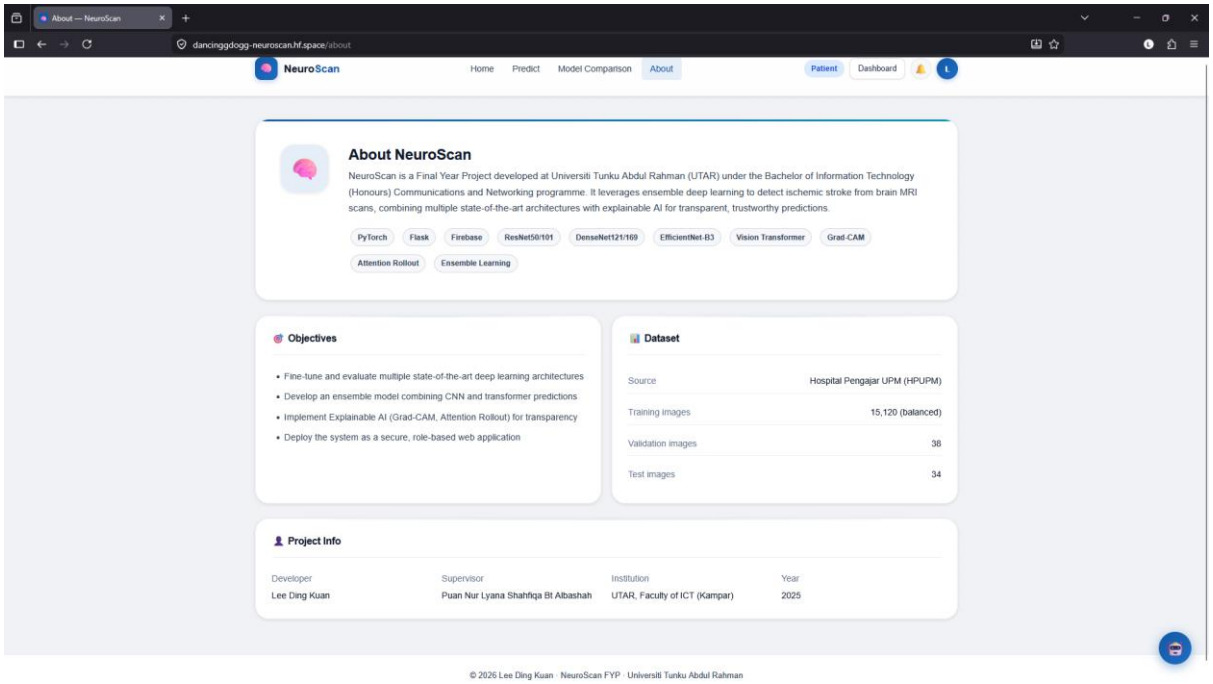

Figure 5.6 About Page

The about page displayed details about the project, including the technologies used (represented as clickable tags), the data source and the goals of the project.

5.4.6 Model Comparison Page

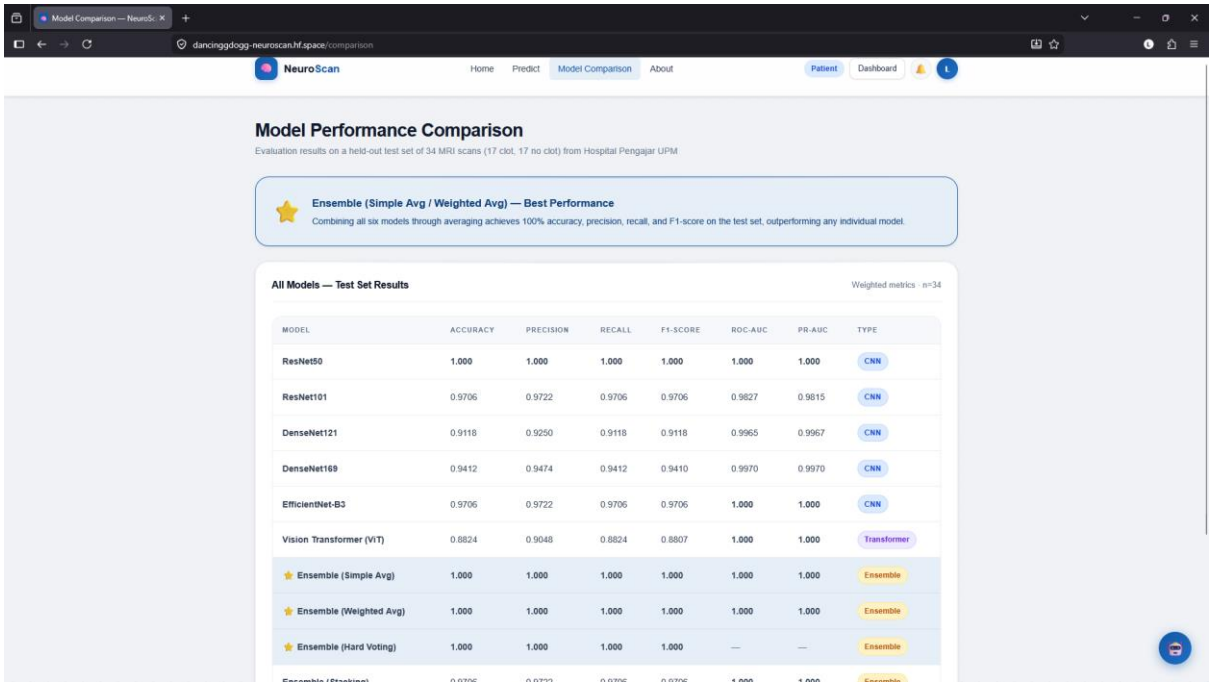

Figure 5.7 Model Comparison Page

The model comparison page presented a table with the test set performance metrics (accuracy, precision, recall, F1-score, ROC-AUC, PR-AUC) of all six individual models and four ensemble methods. This enabled users to choose the best model for their predictions.

5.4.7 Prediction Page

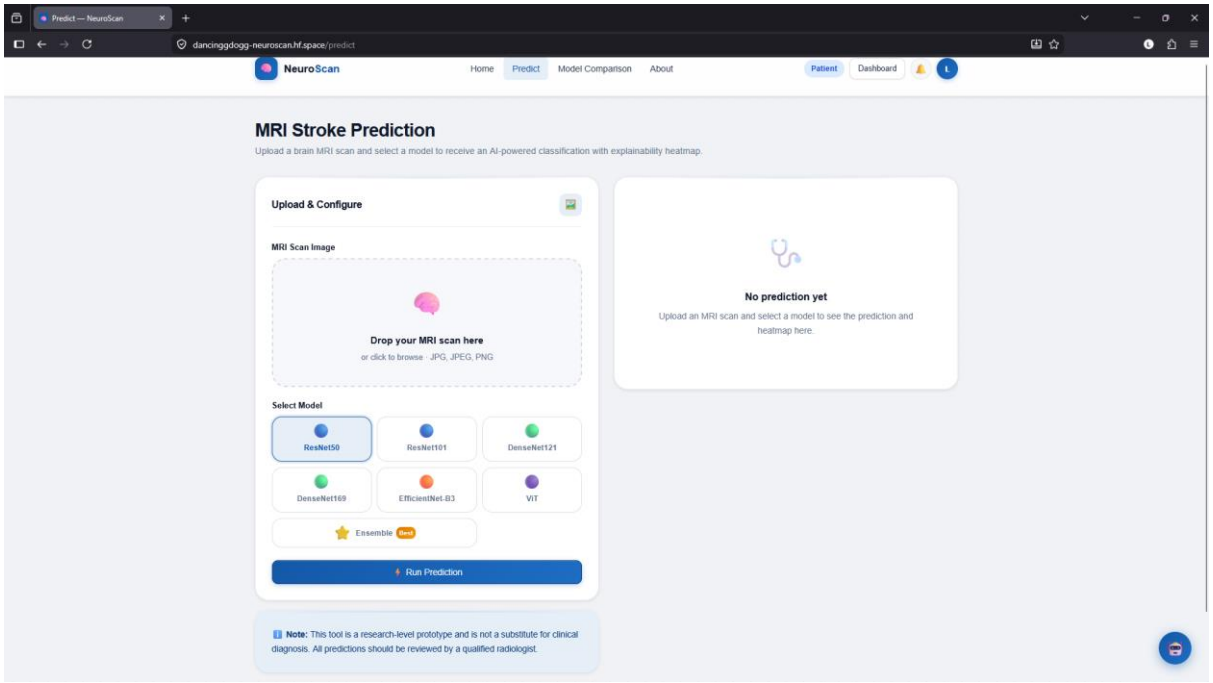

Figure 5.8 Prediction Page

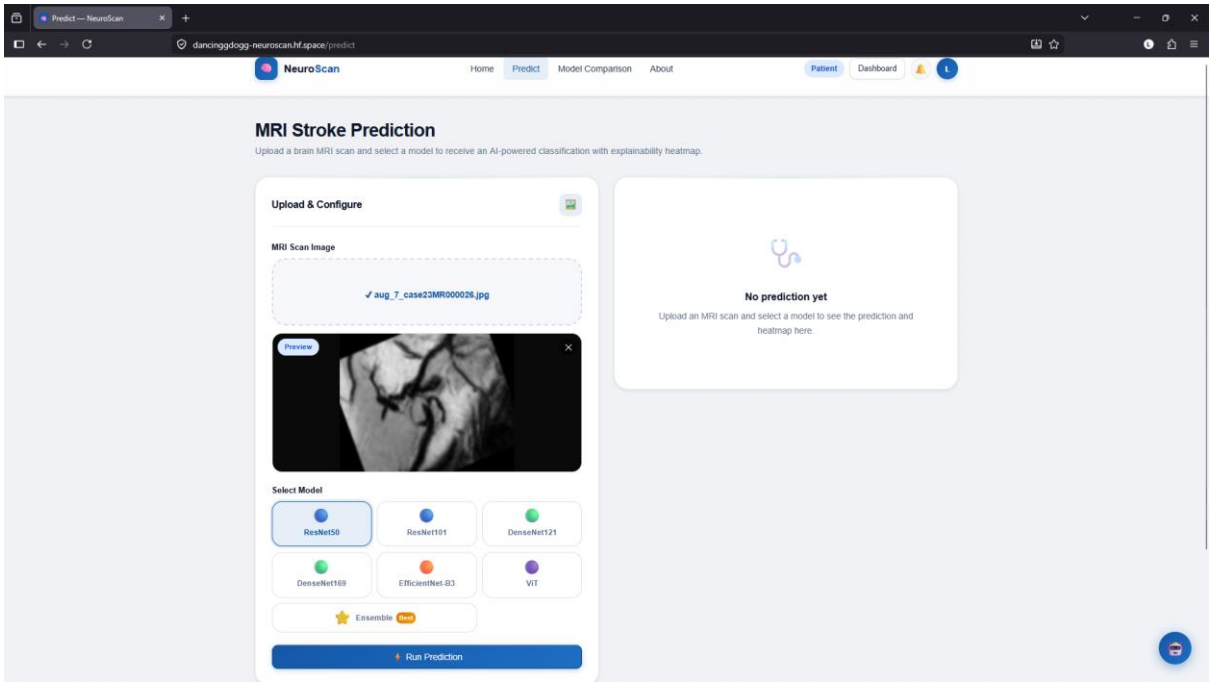

Figure 5.9 Prediction Page — Upload and Model Selection

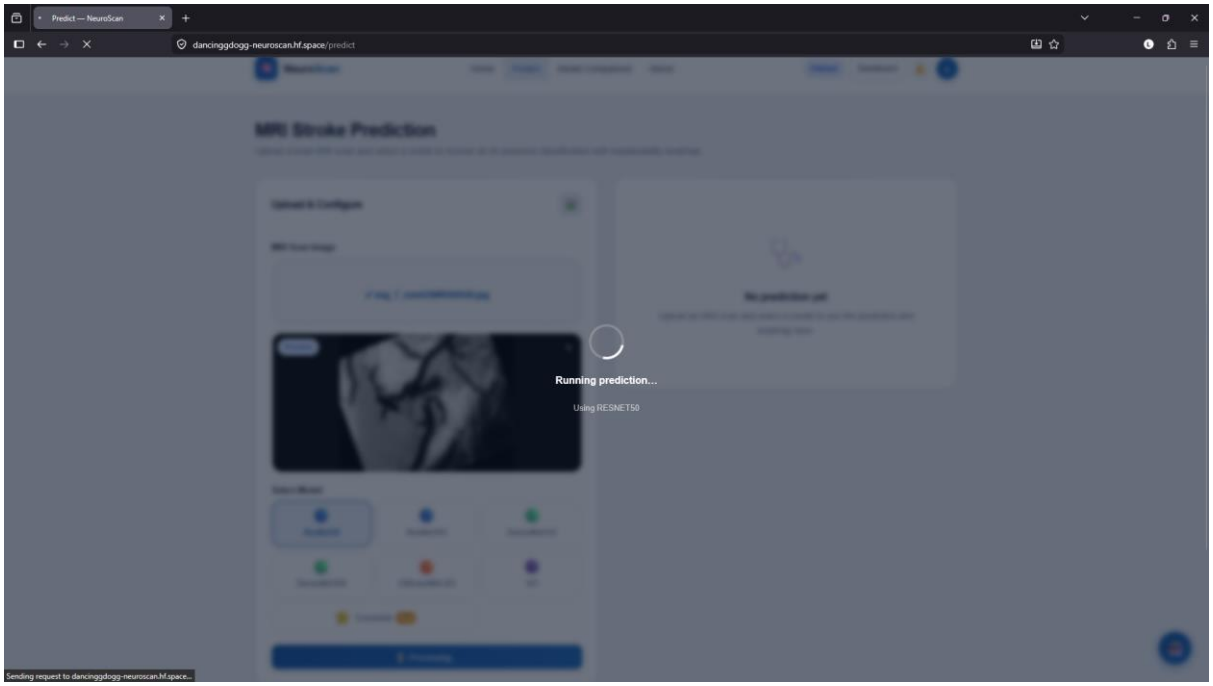

Figure 5.10 Prediction Page — Loading Spinner

The prediction page allowed users to upload their images via drag-and-drop or click and choose, and included an image preview (using JavaScript FileReader API). The models were presented in a radio grid as seven options: the six models (ResNet50, ResNet101, DenseNet121, DenseNet169, EfficientNet-B3, ViT) plus the ensemble model (which took up two columns and had a "Best" badge).

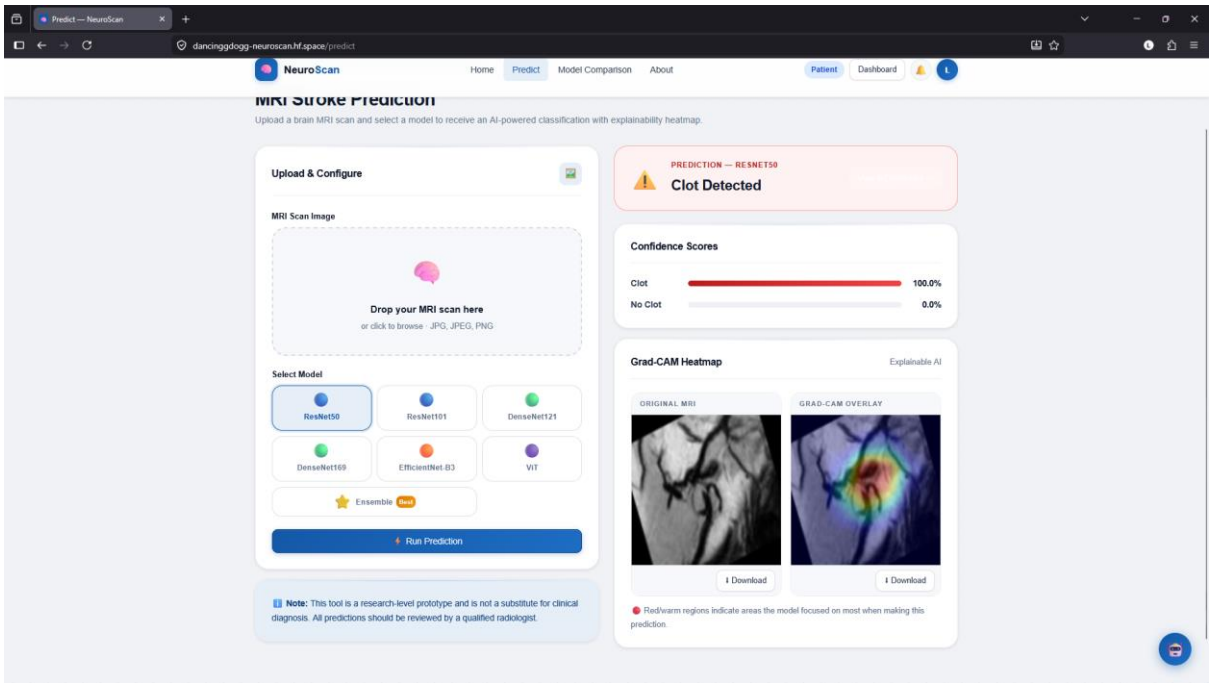

Figure 5.11 Prediction Result — Clot Detected (Red Banner with Pulsing Animation)

The result panel was shown in red, with a pulsating effect and a bouncing exclamation mark if a clot was present. Confidence bars were animated from 0% to the final width using cubic-bezier. The Grad-CAM or Attention Rollout heatmap was overlaid on the MRI image with the option to download.

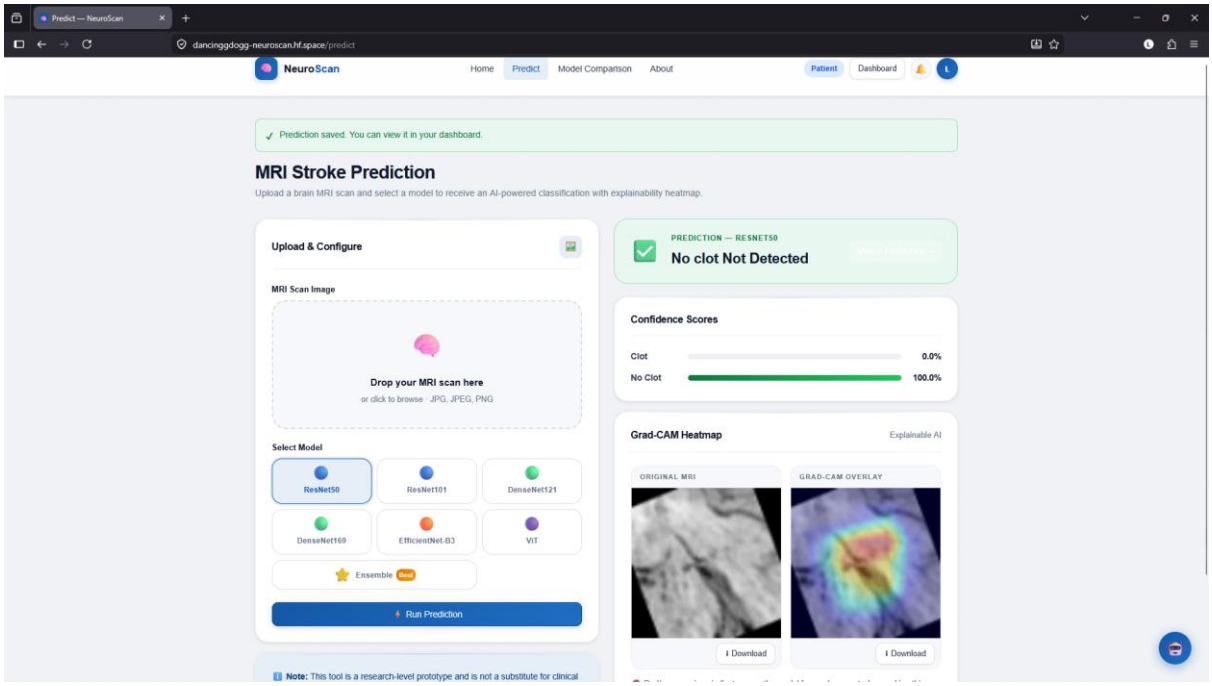

Figure 5.12 Prediction Result — No Clot Detected (Green Banner)

If no clot was found, the result banner was green. The bars and heatmap were shown in the same way as the clot detection result.

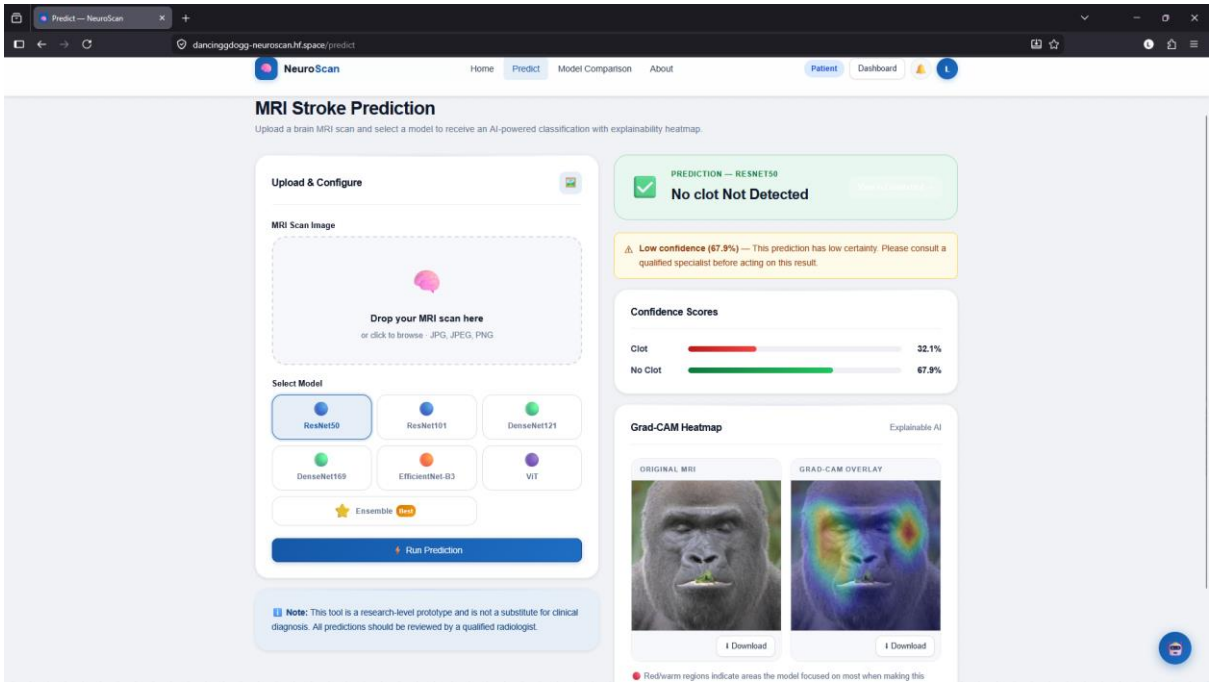

Figure 5.13 Confidence Warning Alert (Below 70%)

If the maximum prediction confidence was below 70%, a warning message was shown under the result banner to consult a doctor and use the ensemble model for better prediction.

5.4.8 Patient Dashboard

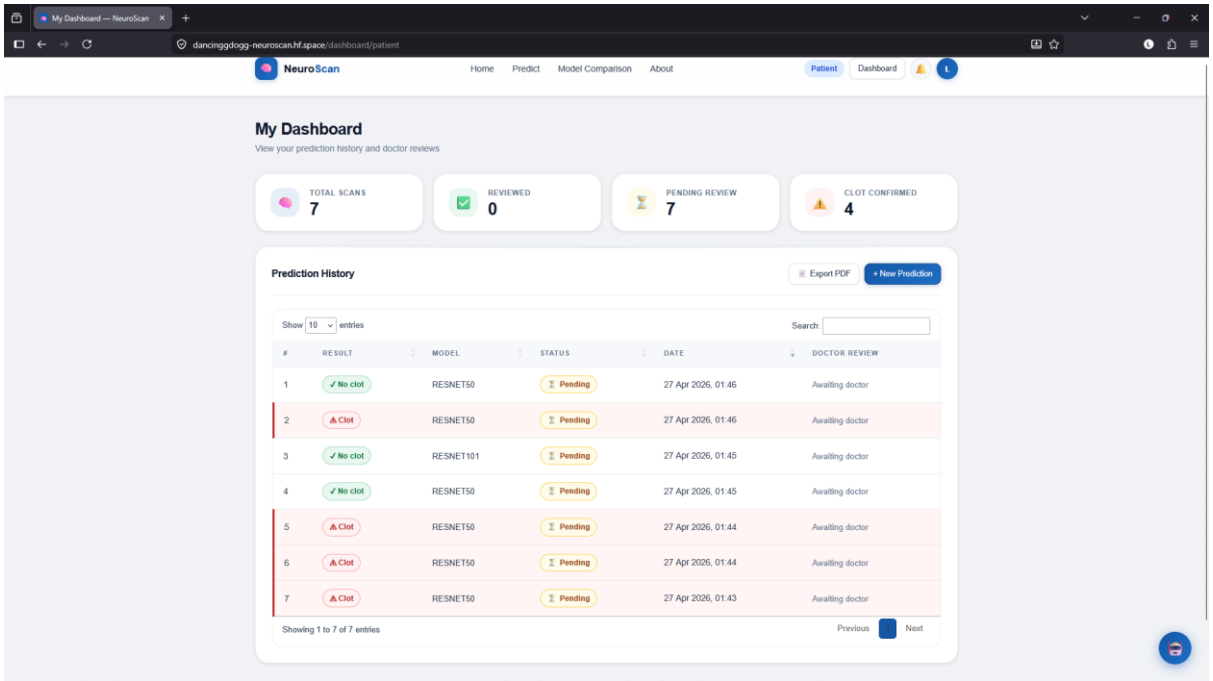

Figure 5.14 Patient Dashboard with Stat Cards and Prediction History

The patient dashboard showed four stat cards (Total Scans, Reviewed, Pending, Clot Detected) followed by a prediction history table using DataTables.js with search, sort and pagination features. Table rows with clot predictions were shaded with a red-tinted background and left border for easy recognition.

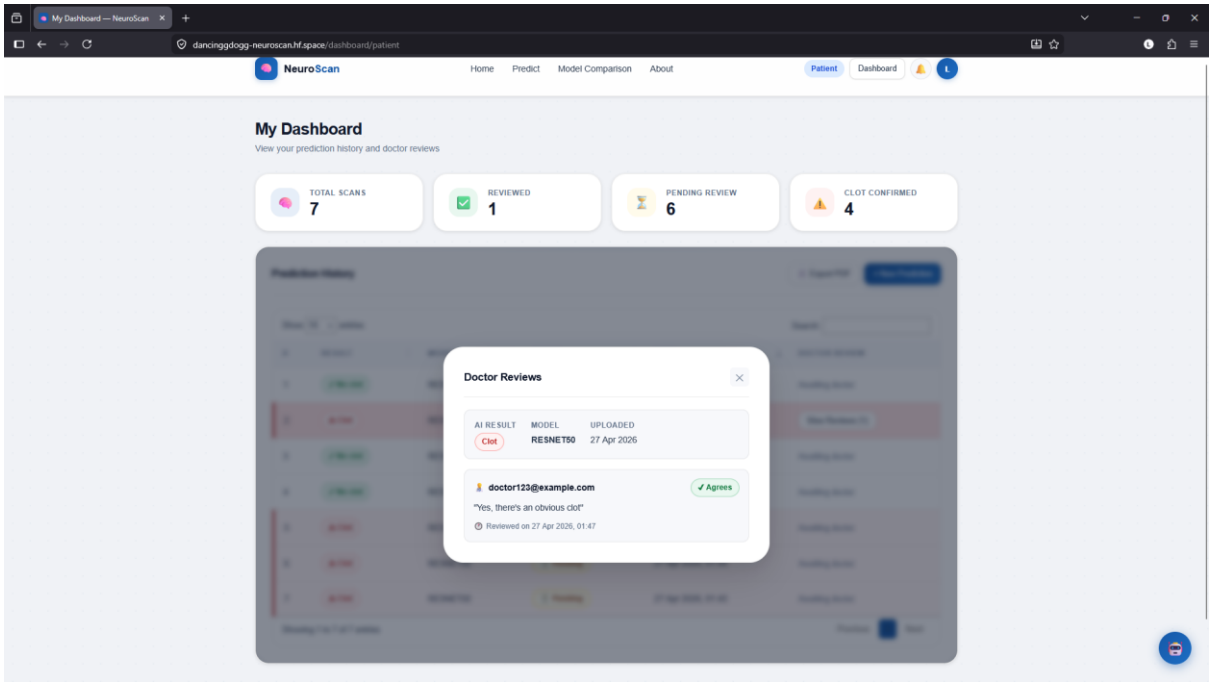

Figure 5.15 Patient Dashboard — View Reviews Modal

The "View Reviews" button for a reviewed prediction displayed a modal dialog with the doctor's review decision (agree or disagree), remarks and the time of the review.

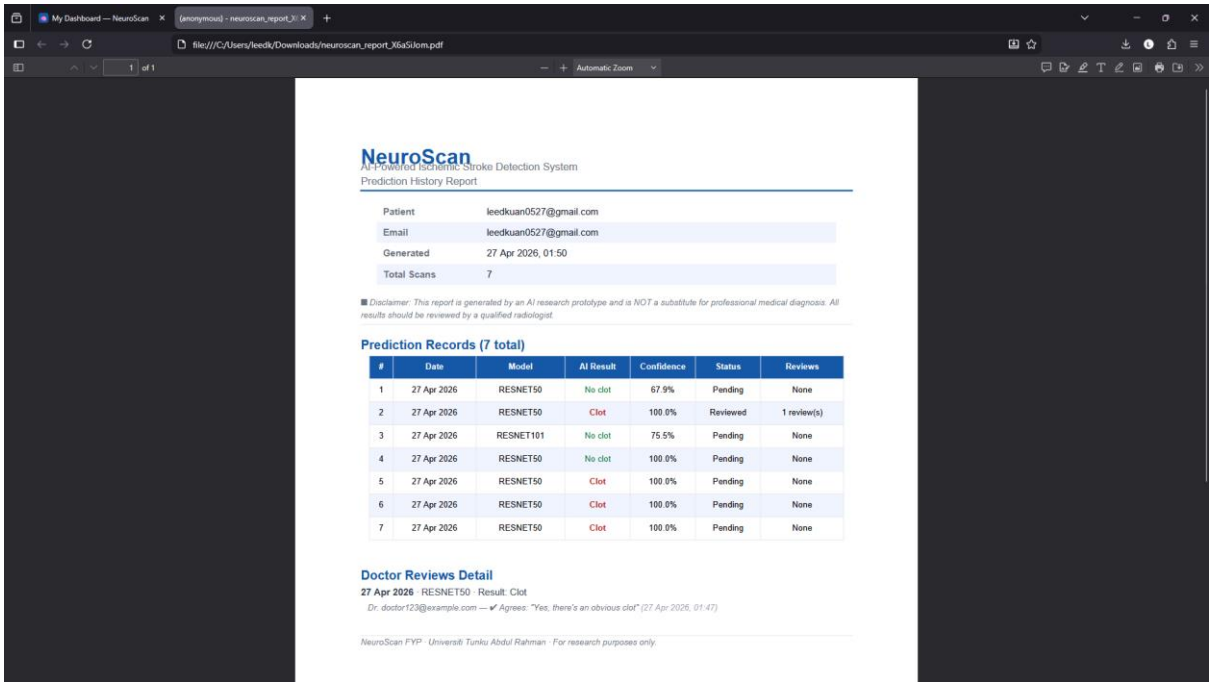

Figure 5.16 PDF Export Example

The "Export PDF" button allowed a downloadable PDF report of the patient information, predictions (with confidence scores), doctor review decisions, and a clinical disclaimer. The PDF was created with ReportLab.

5.4.9 Doctor Dashboard

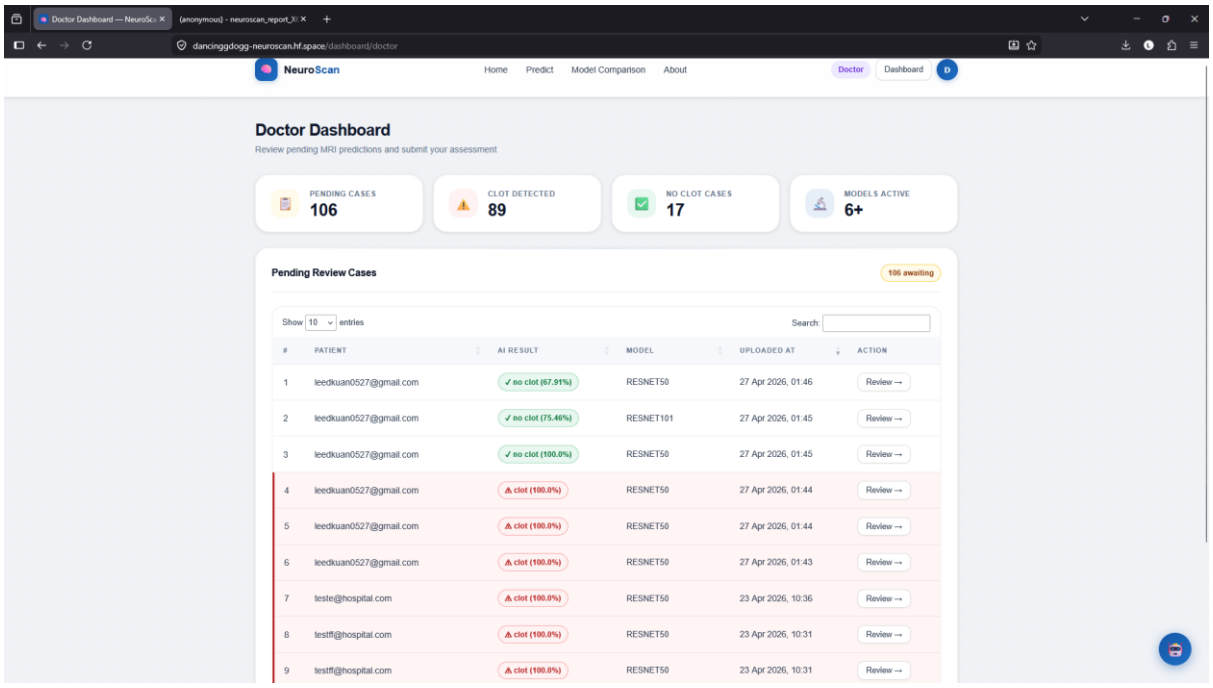

Figure 5.17 Doctor Dashboard with Pending Cases

The doctor dashboard table showed all outstanding prediction cases in a DataTables table. The table displayed the email, model, prediction, and date submitted.

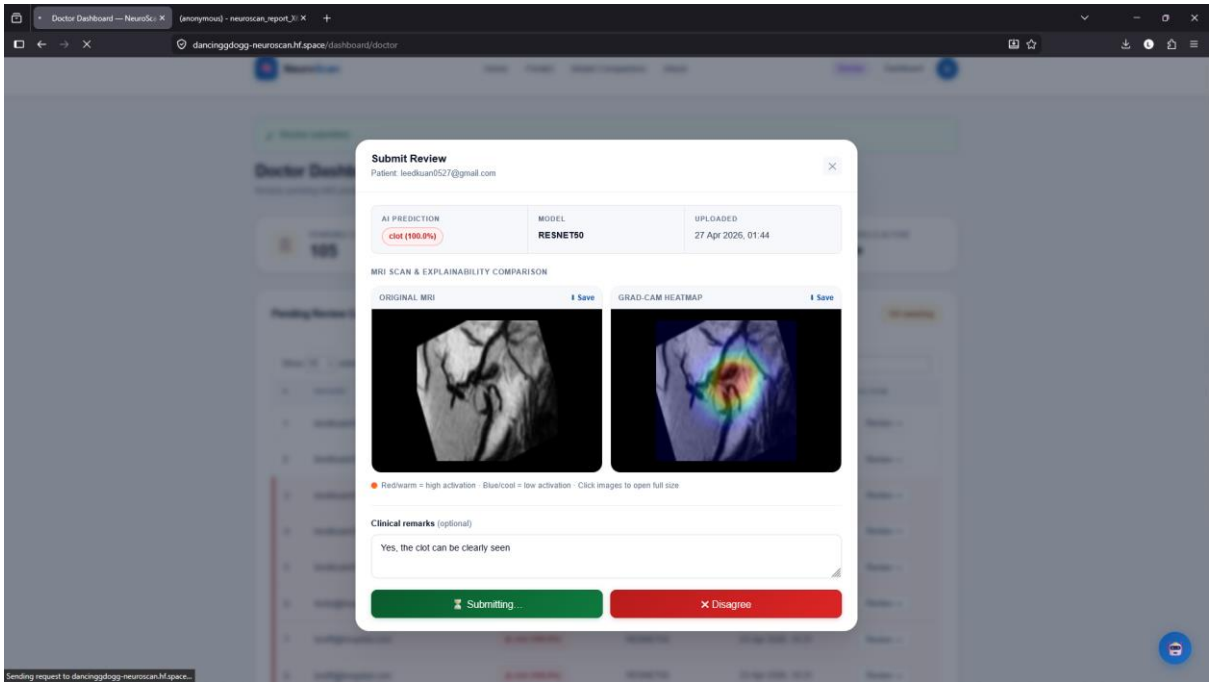

Figure 5.18 Doctor Review — MRI & XAI Image

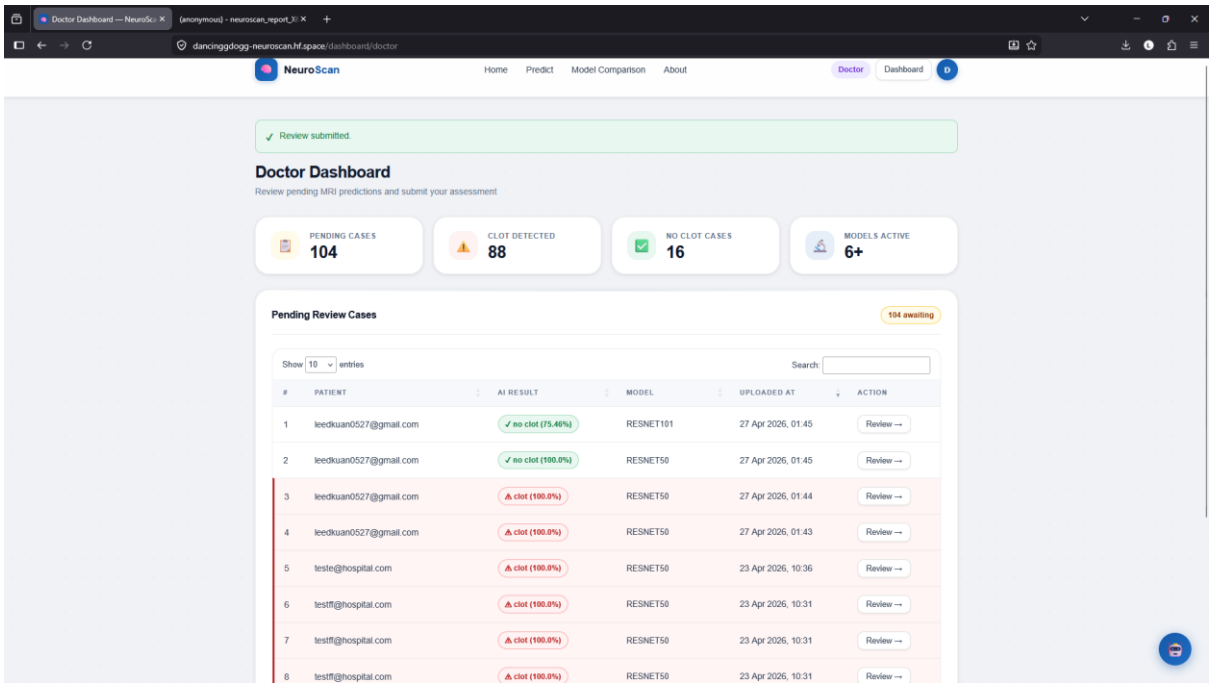

Figure 5.19 Doctor Review Modal

Clicking “Review” in the table opened a modal displaying the original MRI, the Grad-CAM or Attention Rollout heatmap, the prediction case and buttons for the doctor to submit their decision (Agree or Disagree) and optional notes.

5.4.10 Admin Dashboard

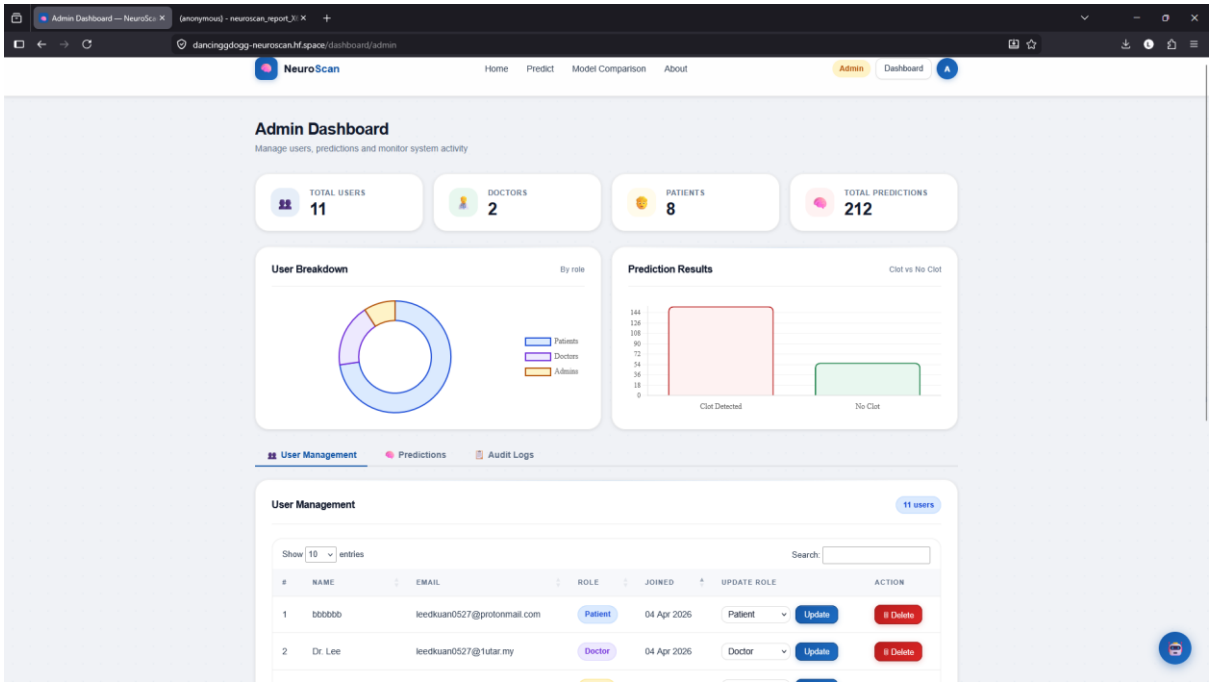

Figure 5.20 Admin Dashboard — Users Tab with Charts

The admin dashboard was divided into three tabs: Users, Predictions, and Logs. The Users tab showed a user distribution doughnut chart (Chart.js), a prediction results bar chart and a list of all users with options to change roles or delete users.

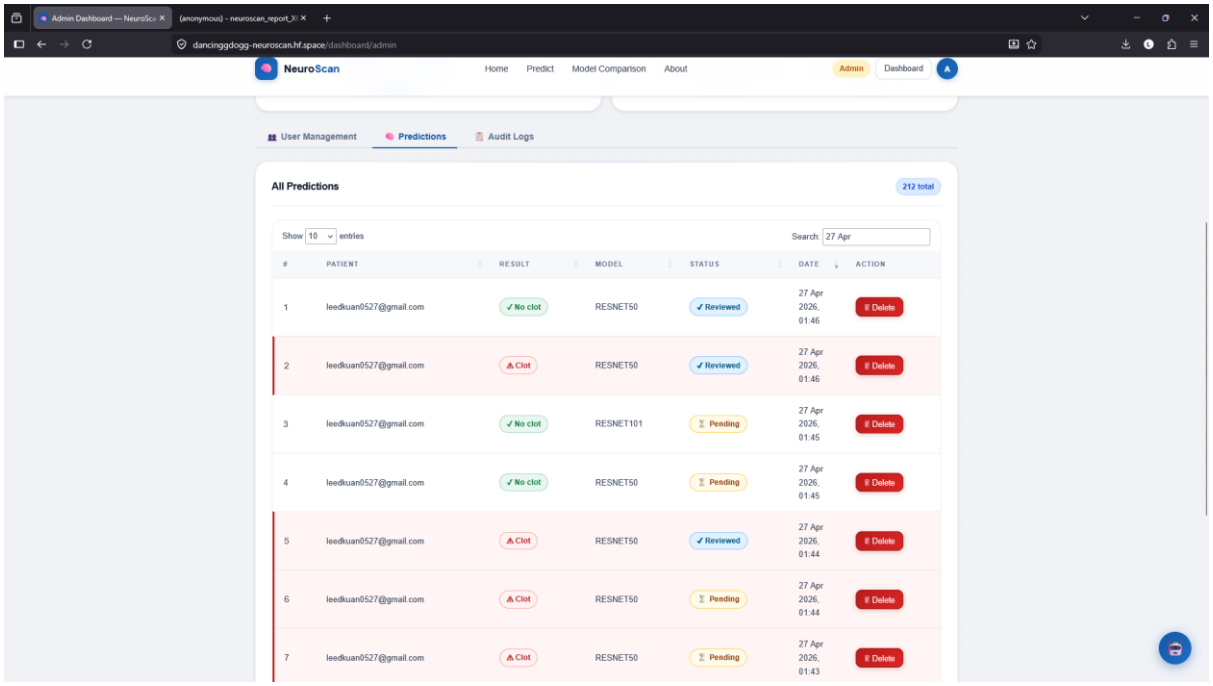

Figure 5.21 Admin Dashboard — Predictions Tab

The Predictions tab showed all prediction records in the system, and allowed for the deletion of individual predictions along with their corresponding reviews.

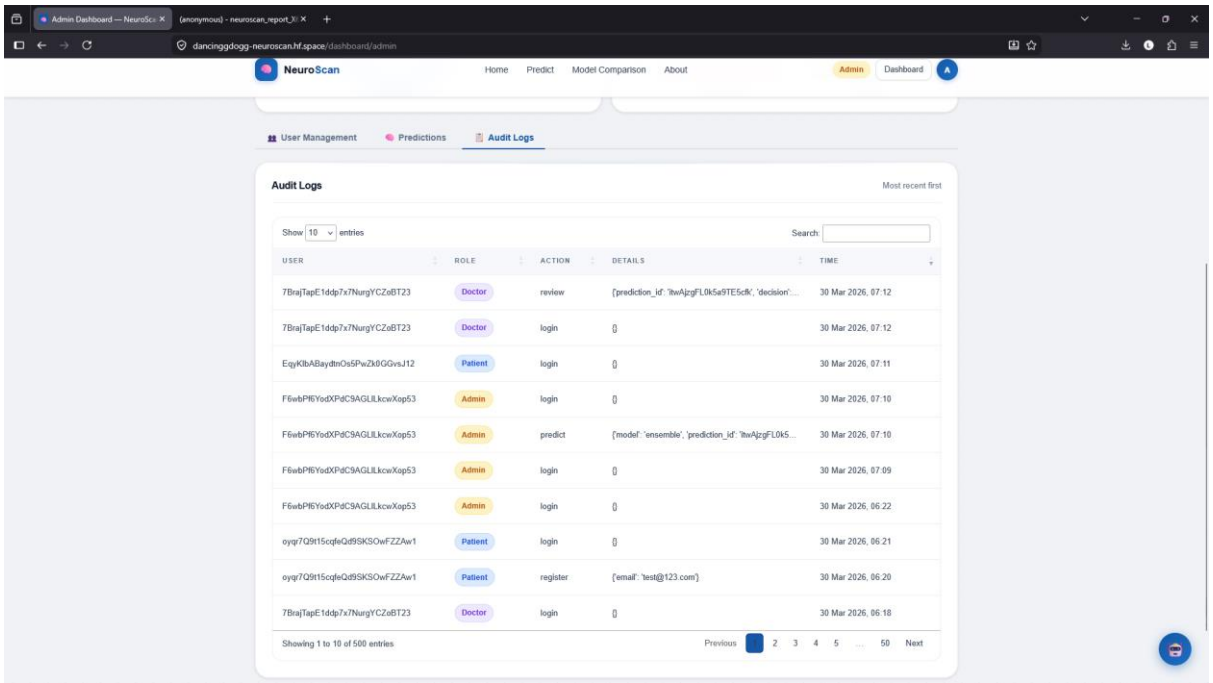

Figure 5.22 Admin Dashboard — Logs Tab

The Logs tab showed the audit log of all user actions, including user email, role, action type, details and time.

5.4.11 Notification System

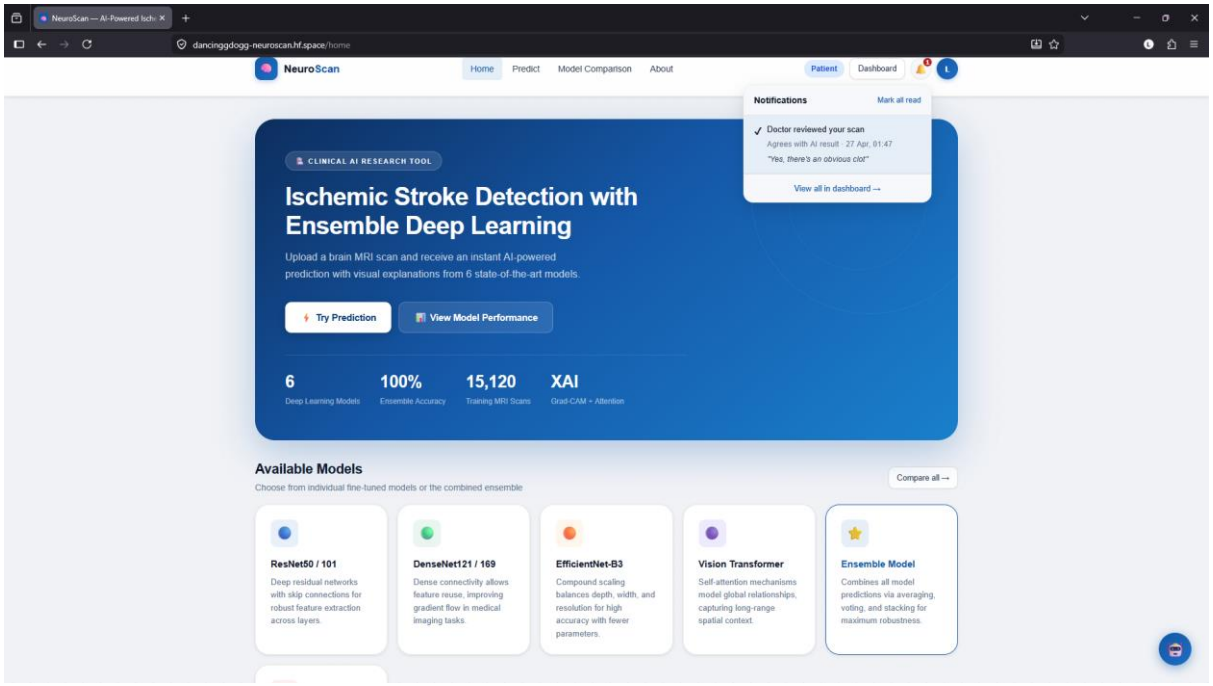

Figure 5.23 Notification Bell with Unread Badge

The bell notification icon was available only for patients in the navbar. The unread badge showed the number of unread notifications. A click on the bell displayed a drop-down of recent notifications, including the doctor's decision and remarks.

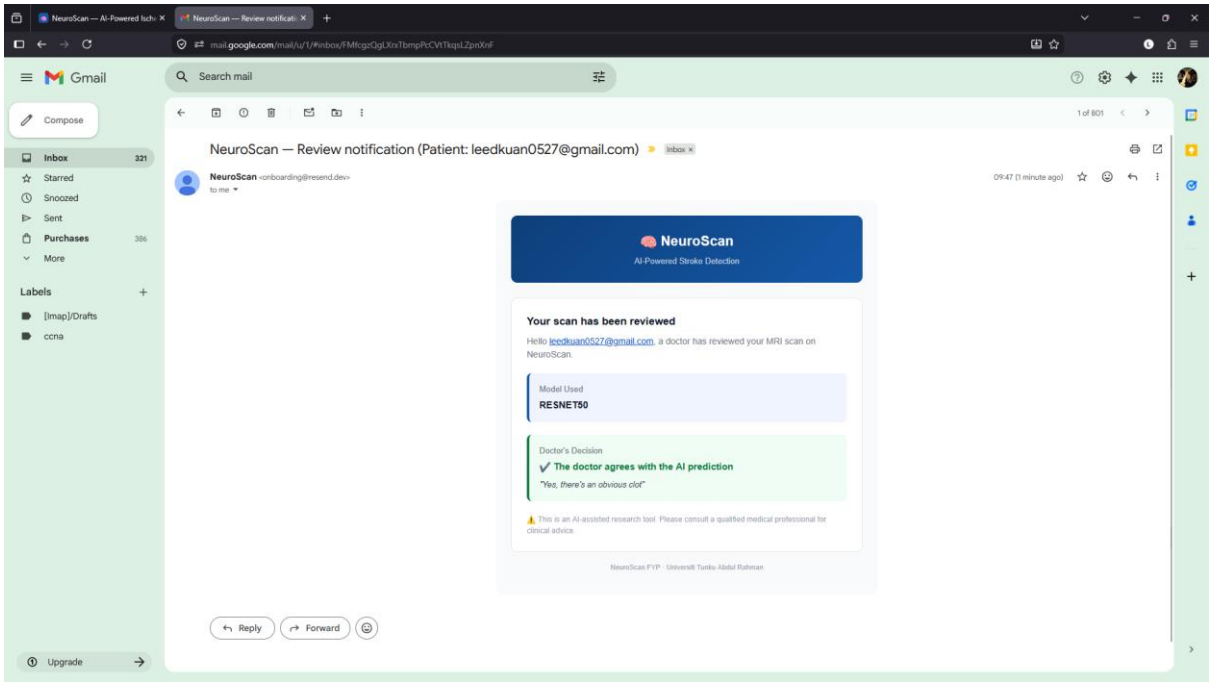

Figure 5.24 Email Notification Example

A doctor's review triggered an email to the patient using Resend API. The email included the doctor's decision, remarks, model, and a clinical disclaimer.

5.4.12 AI Chatbot

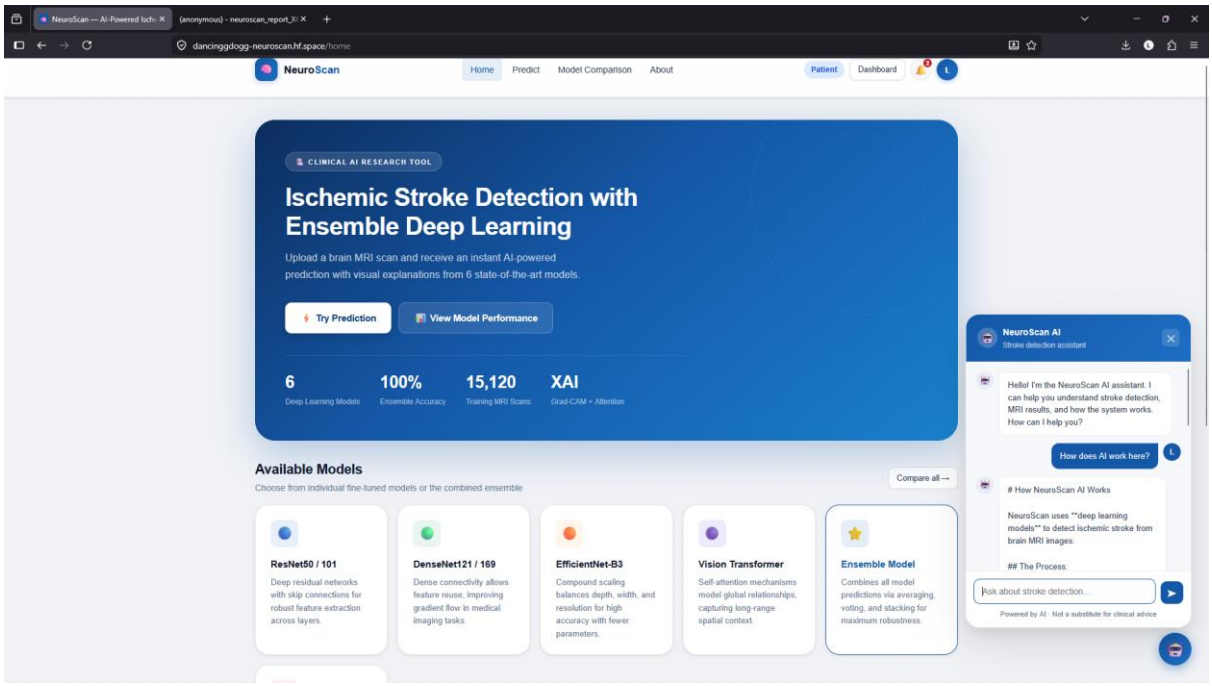

Figure 5.25 AI Chatbot (NeuroScan AI, Powered by Claude Haiku)

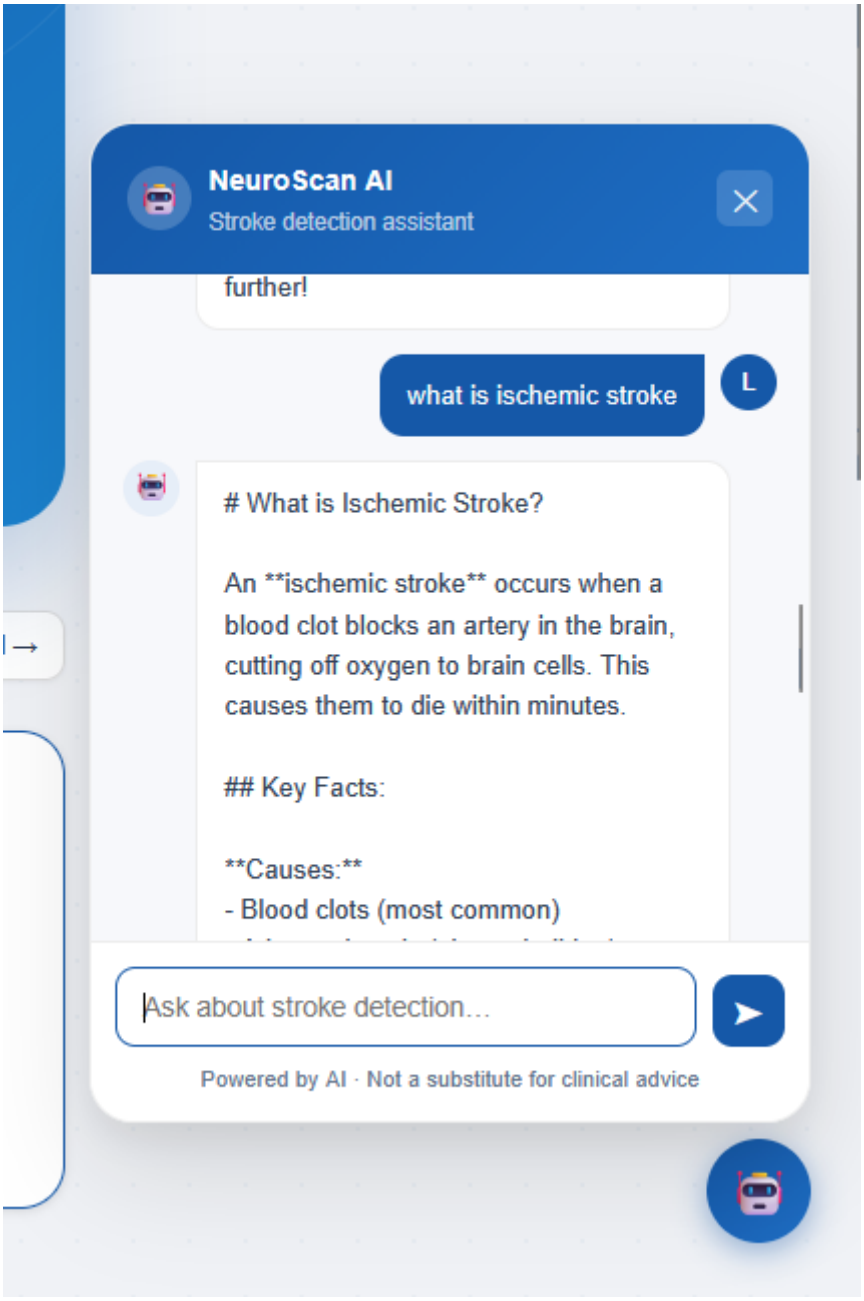

Figure 5.26 AI Chatbot Close-up

All registered users had access to the AI chatbot through the robot button in the lower-right of the screen. The chat panel facilitated conversational interactions with a typing indicator (three dots), user messages in blue bubbles (aligned to the right) and AI replies in white bubbles (aligned to the left). The chatbot was set up with the system prompt to focus on stroke and only provide stroke-related medical information.

5.4.13 Settings Page

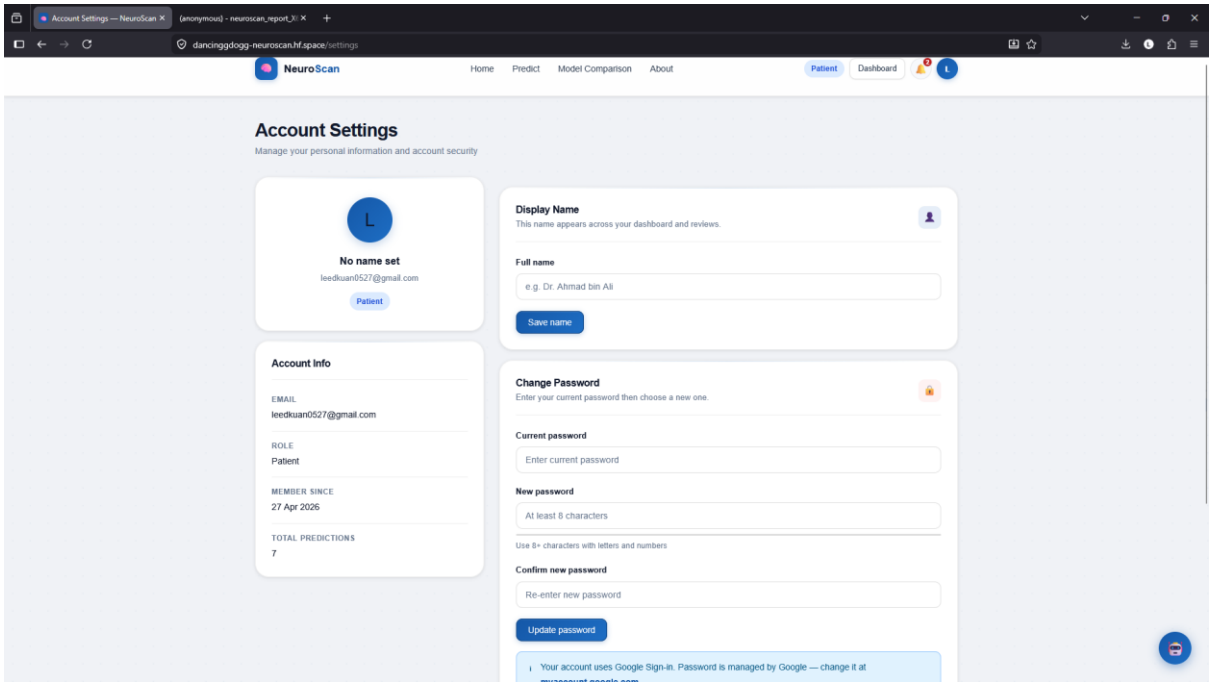

Figure 5.27 Settings Page

The user settings page showed the profile picture and account details, and enabled the user to change their name. Users with email/password could modify their password with a real-time bar indicating password strength. Finally, an informative message was displayed to Google OAuth users that their password was managed by Google.

5.4.14 Session Timeout

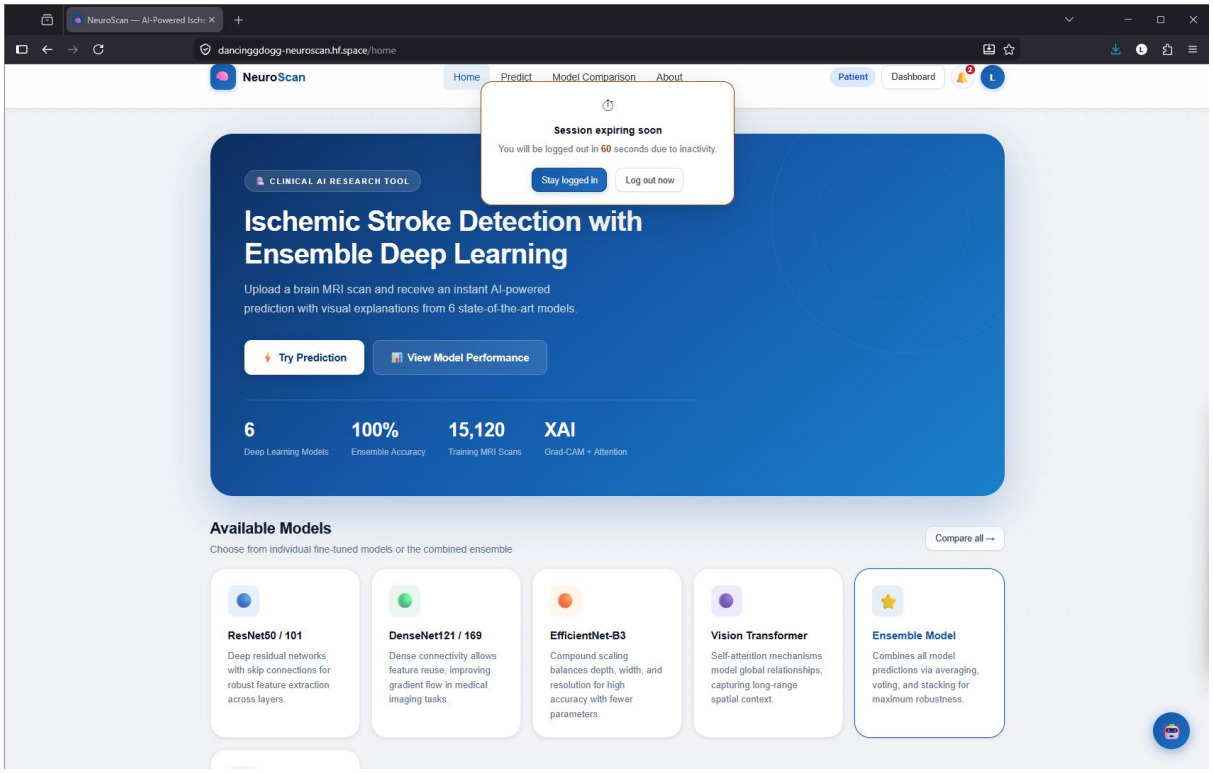

Figure 5.28 Session Timeout Warning Modal

A session timeout warning, with a 60-second countdown, was shown after 29 minutes of inactivity. Clicking on “Stay logged in” would re-initialize the session by performing an AJAX ping to the /ping endpoint. After 60 seconds, the user was logged out and redirected to the login page.

5.4.15 Error Pages

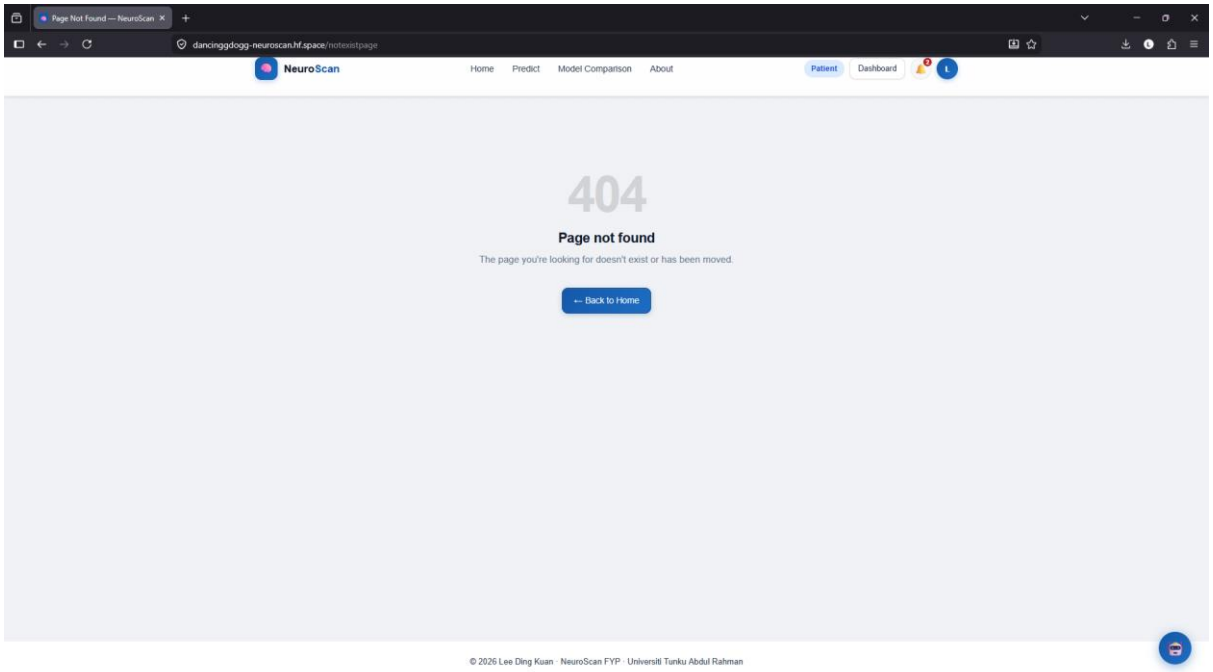

Figure 5.29 404 Error Page

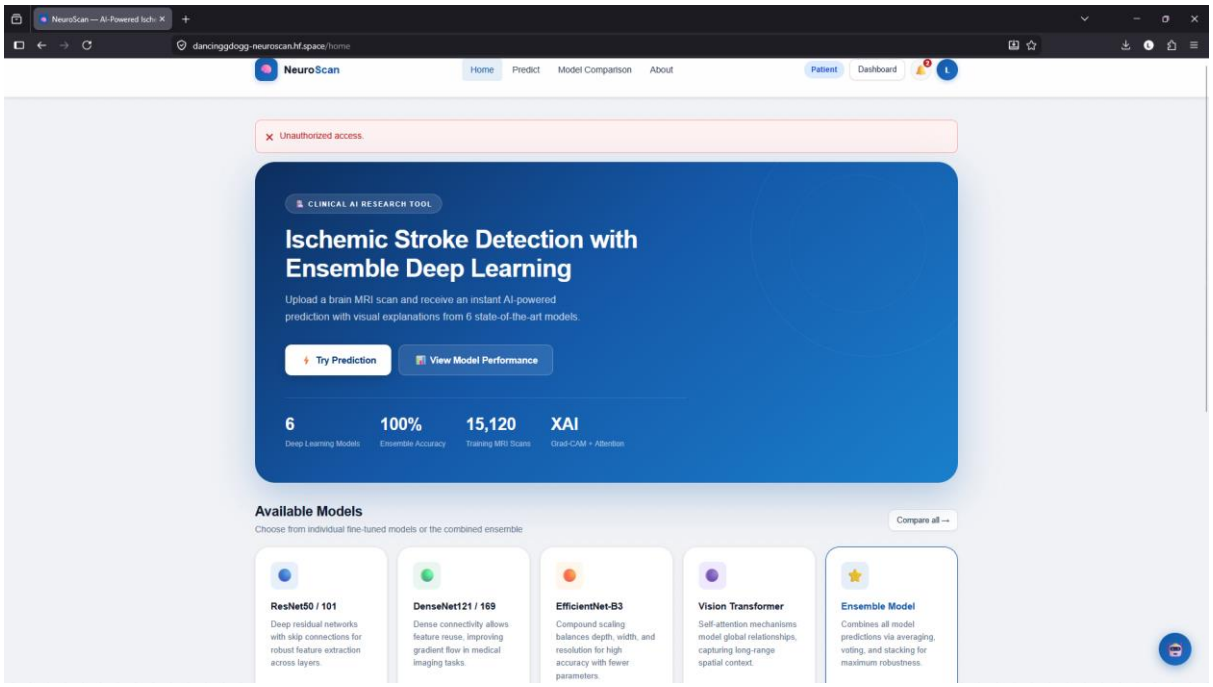

Figure 5.30 Access Restriction

Custom error pages were created for 403 (Forbidden) and 404 (Not Found) HTTP error codes to enhance the user experience if users encountered an access error or a bad URL.

5.5 Implementation Issues and Challenges

This project faced a number of major implementation challenges that were overcome:

1. Hugging Face Spaces push errors: When deploying, Git pushes to the Hugging Face Spaces remote often resulted in "fatal: the remote end hung up unexpectedly" errors with Windows PowerShell. This issue was fixed by using HTTP/1.1 with Git instead of the default, HTTP/2. Further, Hugging Face service rate-limited Git pushes with HTTP 429 error responses after consecutive pushes, making it necessary to wait over 5 minutes before attempting another Git push.
2. Deletion of Firebase service account key: During development, the Firebase service account key was deleted, preventing the application from accessing Firebase services. This was fixed by generating a new key via the Google Cloud Console using a new service account with two IAM roles: Firebase Admin SDK Administrator Service Agent and Cloud Datastore User. A new key was then generated, saved as firebase-key.json and the Hugging Face secret FIREBASE_KEY_JSON was updated.
3. Model weight storage strategy: At first, model weights were stored in the Hugging Face Spaces Git repository. But the total size of 697 MB for seven model weight files led to push failures and was beyond practical repository limits. To overcome this, a separate Hugging Face model repository (DancinggDogg/NeuroScan-models) was set up and the models were downloaded at runtime using the huggingface_hub library. This added a 5-10 minute cold start delay, but allowed the application repository to remain small.
4. SMTP blocking on Hugging Face: Outbound email via SMTP was disallowed on the free tier of Hugging Face Spaces, making it impossible to use "conventional" email services like Gmail SMTP. Following several attempts (including ProtonMail and institutional email SMTP, which also did not work), the Resend API was used as it relies on HTTPS-based REST API calls, rather than SMTP.
5. CSRF and Firebase JS SDK incompatibility: CSRF protection from Flask-WTF was not compatible with Firebase's JavaScript SDK authentication. Firebase Authentication ID tokens are sent as asynchronous JSON POST requests from the client, and do not carry CSRF tokens found in standard HTML form requests. The only option was to turn off CSRF protection, with SameSite=Lax cookies and verification of Firebase ID tokens on the server.
6. Firestore N+1 query pattern: Both the doctor and admin dashboards originally had poor performance with N+1 queries, whereby each row of predictions had to make an additional Firestore query to retrieve the user's email. This was fixed by caching all user

documents in a dictionary at the beginning of the dashboard route, and looking up these users from the cache rather than querying Firestore for each user.

7. Grad-CAM hook re-registration: Initially, the GradCAM objects were created on each request to make a prediction, which caused new forward and backward hooks to be registered on the model's convolutional layers. This led to memory leaks and performance degradation over time. This problem was solved by caching GradCAM objects during the app startup in a GRAD_CAM_CACHE dictionary, so the hooks were registered only once for each model.
8. Ephemeral storage on Hugging Face: Hugging Face Spaces has ephemeral storage that is reset on container restart. This meant that MRI images uploaded and Grad-CAM heatmaps generated were deleted on restart of the container. This was considered a limitation. One possible improvement would be to store images in Firebase Cloud Storage or an object storage service.

5.6 Concluding Remark

This chapter described the implementation of the NeuroScan system including the configuration of the hardware and software, as well as deployment. The system has been deployed as a fully functional web app at https://dancinggdogg-neuroscan.hf.space. The system was illustrated with 30 screenshots depicting the main features for the three types of users. A number of key implementation challenges were addressed, such as deployment platform limitations, integration with authentication and performance tuning. The approaches taken to address these issues were sensible and well-explored. The following chapter tests the system and assesses its performance.

CHAPTER 6

System Evaluation and Discussion

This chapter assesses the NeuroScan system through functional testing, cross-browser compatibility testing, security testing, model performance testing and an analysis of how the project goals have been achieved.

6.1 System Testing and Performance Metrics

The system was tested through three types of testing: functional testing to check that all features operated correctly, cross-browser testing to check that the system behaved the same across multiple web browsers and security testing to check that the security features were effective. Performance of the machine learning models was assessed using classification metrics on the test data.

6.2 Testing Setup and Result

6.2.1 Functional Testing

Functional tests were conducted on the deployed system (https://dancinggdogg-neuroscan.hf.space) to confirm that system features worked across the three user roles. The results of the functional testing are shown in Table 6.1.

Table 6.1 Functional Test Results (38 Test Cases)

#	Module	Test Case	Expected	Result
1	Auth	Email/password login with valid credentials	Login success	Pass
2	Auth	Email/password login with invalid credentials	Error message	Pass
3	Auth	Google OAuth login (existing user)	Login success	Pass
4	Auth	Google OAuth login (new user, role picker appears)	Role modal	Pass
5	Auth	Register with valid email and password	Account created	Pass

6	Auth	Register with blocked email domain (mailinator.com)	Rejected	Pass
7	Auth	Logout redirects to login page	Redirect	Pass
8	Predict	Upload valid MRI image (JPEG)	Preview shown	Pass
9	Predict	Upload non-image file (e.g. .txt)	Rejected	Pass
10	Predict	Run prediction with individual model (ResNet50)	Result shown	Pass
11	Predict	Run prediction with ensemble model	Result shown	Pass
12	Predict	Grad-CAM heatmap displayed for CNN model	Heatmap visible	Pass
13	Predict	Attention Rollout heatmap displayed for ViT	Heatmap visible	Pass
14	Predict	Confidence warning appears when prediction < 70%	Warning shown	Pass
15	Patient	Patient dashboard shows prediction history	Table loaded	Pass
16	Patient	Clot predictions highlighted with red tint	Red rows	Pass
17	Patient	View Reviews modal opens with doctor info	Modal opens	Pass
18	Patient	Export PDF downloads valid report	PDF download	Pass
19	Patient	Notification bell shows unread count	Badge visible	Pass
20	Patient	Mark notification as read	Badge updates	Pass
21	Doctor	Doctor dashboard shows pending cases	Table loaded	Pass
22	Doctor	Review modal shows MRI + heatmap	Images shown	Pass
23	Doctor	Submit agree decision with notes	Review saved	Pass
24	Doctor	Submit disagree decision	Review saved	Pass

25	Doctor	Email notification sent to patient after review	Email received	Pass
26	Admin	Admin dashboard loads all three tabs	Tabs work	Pass
27	Admin	Update user role from patient to doctor	Role updated	Pass
28	Admin	Delete user (cascades to predictions + reviews)	User deleted	Pass
29	Admin	Delete prediction and linked reviews	Deleted	Pass
30	Admin	Charts render on admin dashboard	Charts visible	Pass
31	Chatbot	Chatbot button visible for authenticated users	Button visible	Pass
32	Chatbot	Send message and receive AI response	Response shown	Pass
33	Settings	Update display name	Name updated	Pass
34	Settings	Change password (email/password user)	Password changed	Pass
35	Session	Session timeout warning after 29 min inactivity	Modal appears	Pass
36	Session	Auto-logout after countdown expires	Logged out	Pass
37	Access	Patient cannot access /dashboard/admin	403 error	Pass
38	Access	Doctor cannot access /dashboard/patient	403 error	Pass

All 38 functional test cases were completed successfully, demonstrating that the system's primary features, role-based access control and integration points were working properly.

6.2.2 Cross-Browser Compatibility Testing

This test was executed across three main desktop web browsers: Google Chrome (134), Mozilla Firefox (137) and Microsoft Edge (134). A total of 90 test cases were performed across 10 categories. Table 6.2 summarises the results.

Table 6.2 Cross-Browser Compatibility Test Summary

Browser	**Pass**	**Partial**	**Fail**	**Total**

Google Chrome 134	86	4	0	90
Mozilla Firefox 137	86	4	0	90
Microsoft Edge 134	86	4	0	90

There were no critical issues. The partial results were consistent across browsers and reflected design choices, rather than browser-specific bugs: the link redirection of the "View in Dashboard" button not working as expected, the responsive layout (≤540px) not replacing the visible navigation links with a hamburger menu (the latter being a design trade-off for this clinical application targeted at the desktop), and a minor CSP warning in the Chrome console regarding the Firebase SDK source map files. The 90 test case results are detailed in the appendix.

6.2.3 Security Testing

The security tests confirmed the effectiveness of the security measures. Table 6.3 shows the results of security testing.

Table 6.3 Security Test Results

#	Security Measure	Test Performed	Expected	Result
1	Rate limiting (login)	Sent 6 POST requests to /session_login within 1 minute	HTTP 429	Pass
2	Rate limiting (predict)	Submitted 11 predictions within 1 minute	HTTP 429	Pass
3	Rate limiting (chat)	Sent 21 chat messages within 1 minute	HTTP 429	Pass
4	Magic byte validation	Uploaded a .txt file renamed to .jpg	Rejected	Pass
5	CSP headers	Inspected response headers in DevTools → Network tab	CSP present	Pass
6	Role-based access	Patient user accessed /dashboard/admin URL directly	403 error	Pass
7	Prediction ownership	Verified patient dashboard only shows own predictions	Own only	Pass
8	Session timeout	Left session idle for 30+ minutes	Auto-logout	Pass

9	Blocked email domains	Attempted registration with mailinator.com email	Rejected	Pass
10	HTML sanitisation	Doctor submitted review notes containing <script> tag	Escaped	Pass

All 10 security tests were successful and the security measures implemented were shown to be effective against the attack vectors.

6.2.4 Prediction Latency Testing

The prediction latency tests were performed to assess whether the system achieved the goal of providing predictions in less than 10 seconds per image (specified in Section 3.3). The latency was measured as the total time from when the POST /predict request was received to the response from the server, as indicated in the browser's DevTools Network tab. Testing was done on the deployed system, which had a shared CPU instance, no GPU, on the free tier of Hugging Face Spaces.

The same MRI image was used for all tests to avoid the effects of varying inputs. Three runs were performed for each model after a warm-up run, which was ignored. This warm-up run accounted for model caching. The first run of each model had higher latency due to loading model weights and Grad-CAM hooks into memory. Later runs were performed on cached models and thus measured steady-state inference latency. Table 6.4 shows the latencies.

Table 6.4 Prediction Latency Test Results

Model	Run #1 (ms)	Run #2 (ms)	Run #3 (ms)	Average (ms)	Target met? (Average<10,000ms)
ResNet50	3,064	2,432	1,879	2,458	YES
ResNet101	2,031	2,409	2,158	2,199	YES
DenseNet121	1,787	1,797	1,917	1,834	YES
DenseNet169	1,941	1,835	1,983	1,920	YES
EfficientNetB3	1,759	1,708	1,912	1,793	YES
ViT	1,989	1,913	1,808	1,903	YES
Ensemble	3,688	3,655	3,589	3,644	YES

Each of the seven configurations was within the 10-second latency budget. The average return times ranged from 1,793ms to 2,458ms, a tight range for such a variety of architectures. EfficientNet-B3 had the lowest average latency (1,793ms), in line with its design principle of

compound scaling for efficiency. The smallest ResNet variant, ResNet50, exhibited the largest individual model latency (2,458ms), which could be due to variance in the allocation of shared CPU resources on the Hugging Face Spaces free tier instead of being inherently inefficient.

Interestingly, the Vision Transformer (ViT) model attained a comparable average latency (1,903ms) compared to CNNs. This can be explained by the Grad-CAM caching and ViT processor pre-loading optimisations performed during the start-up of the application, as outlined in Section 4.4.2. Without these optimisations, the ViT would have taken much longer due to the re-initialisation of the processor and re-registration of the attention hook for each request.

The ensemble, which triggered the six base models in parallel using Python's ThreadPoolExecutor, had an average latency of 3,644ms - or 1.5 to 2 times longer than the individual models. This delay was anticipated, given the ensemble must wait for all six models to complete inference and then aggregate their probabilities. Despite this performance penalty, the ensemble was still significantly faster than the 10-second target, and the parallelisation approach was better than running the six models in sequence.

One interesting aspect was the overhead for the first run, where the first execution was significantly longer than subsequent runs. For instance, ResNet50's first run was 7,407ms, while the average of subsequent runs was 2,458ms, and DenseNet121's first run was 8,991ms, with an average steady-state run time of 1,834ms. This is expected, with PyTorch models being JIT compiled and allocated memory on first use. This would not be an issue for users of a production system with long-lived processes and pre-loaded models at system startup. But the Hugging Face Spaces free tier could cause containers to be recycled due to inactivity, so users who connect to the system after a cold start may encounter a one-off performance dip before the system returns to steady state.

In summary, the results of the prediction latency experiment indicated that all model configurations achieved the target performance under CPU-only deployment. They also confirmed the improvements made to the ML pipeline (caching, parallelisation).

6.3 Model Performance Analysis

The individual models and ensemble models were tested on the held-out test set of 34 MRI scans (17 clot, 17 no clot). The full set of performance metrics are given in Table 6.5.

Table 6.5 Model Performance Metrics on Test Set (n=34)

Model	Acc	Prec	Rec	F1	ROC	PR	Type
ResNet50	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	CNN
ResNet101	0.9706	0.9722	0.9706	0.9706	0.9827	0.9815	CNN
DenseNet121	0.9118	0.9250	0.9118	0.9118	0.9965	0.9967	CNN
DenseNet169	0.9412	0.9474	0.9412	0.9410	0.9970	0.9970	CNN
EfficientNet-B3	0.9706	0.9722	0.9706	0.9706	1.000	1.000	CNN
ViT	0.8824	0.9048	0.8824	0.8807	1.000	1.000	Trans.
Ens. (Simple Avg)	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	Ens.
Ens. (Weighted Avg)	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	Ens.
Ens. (Hard Voting)	**1.000**	**1.000**	**1.000**	**1.000**	—	—	Ens.
Ens. (Stacking)	0.9706	0.9722	0.9706	0.9706	1.000	1.000	Ens.

From the model performance analysis, ResNet50 outperformed the other models, with performance (accuracy, precision, recall, F1-score, ROC-AUC, PR-AUC) of 1.000. ResNet101 and EfficientNet-B3 had the second-highest accuracy (0.9706). DenseNet169 (0.9412) achieved higher accuracy than DenseNet121 (0.9118), proving that increased depth in DenseNet can lead to better performance. The Vision Transformer (ViT) had the lowest accuracy (0.8824) among the individual models, which was in line with reports that ViT needs to pre-train on larger datasets and tends to fall short of CNNs on small datasets [8].

Three of the four ensemble techniques (simple averaging, weighted averaging, and hard voting) attained perfect scores on all relevant performance measures, which showed the benefits of ensemble learning. The stacking ensemble's accuracy of 0.9706 was comparable to ResNet101 and EfficientNet-B3, but did not outperform the other ensemble methods. This indicated that, at least for this dataset, the additional complexity of a meta-learner did not lead to improvements over simple probability averaging.

All models achieved ROC-AUC values and PR-AUC values above 0.98, with many models scoring 1.000. This demonstrated good discrimination at all thresholds. The ROC-AUC and PR-AUC values were not calculated for hard voting because it returns a class label, not a probability.

The results were promising but it was necessary to point out that the test set consisted of 34 images from a single hospital (HPUPM). The 100% results from some of the models may be due to the lack of diversity in the test set rather than necessarily generalising well to external data sets. More diverse and larger test sets would be required to prove clinical generalisation.

The confusion matrices for all 10 models on the 34 MRI scans (17 clot, 17 no clot) test set are provided in Figures 6.1 to 6.10. ResNet50, Ensemble (Simple Average), Ensemble (Weighted Average) and Ensemble (Hard Voting) had zero misclassifications. DenseNet121 had the most misclassifications (3 out of 34) and ViT had 4 misclassifications, which is in line with its lower accuracy of 0.8824.

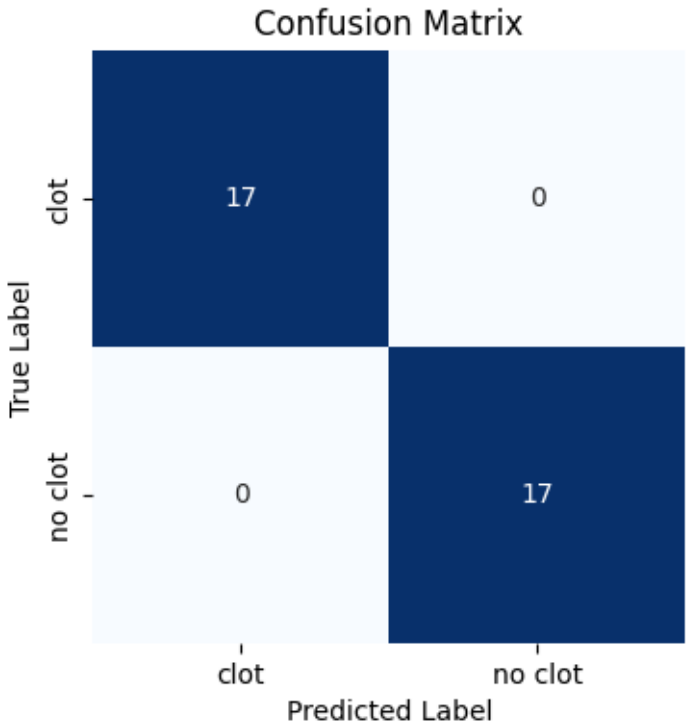

Figure 6.1 Confusion Matrix for ResNet50

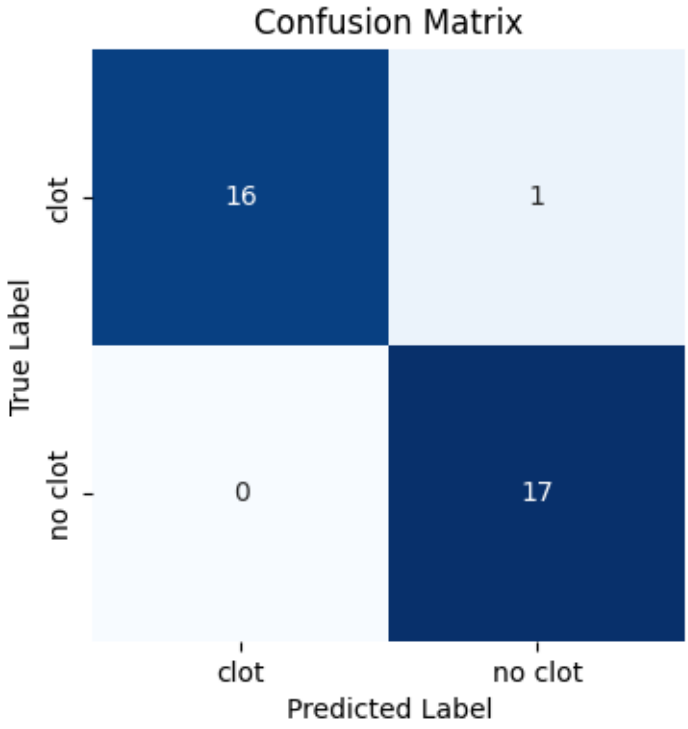

Figure 6.2 Confusion Matrix for ResNet101

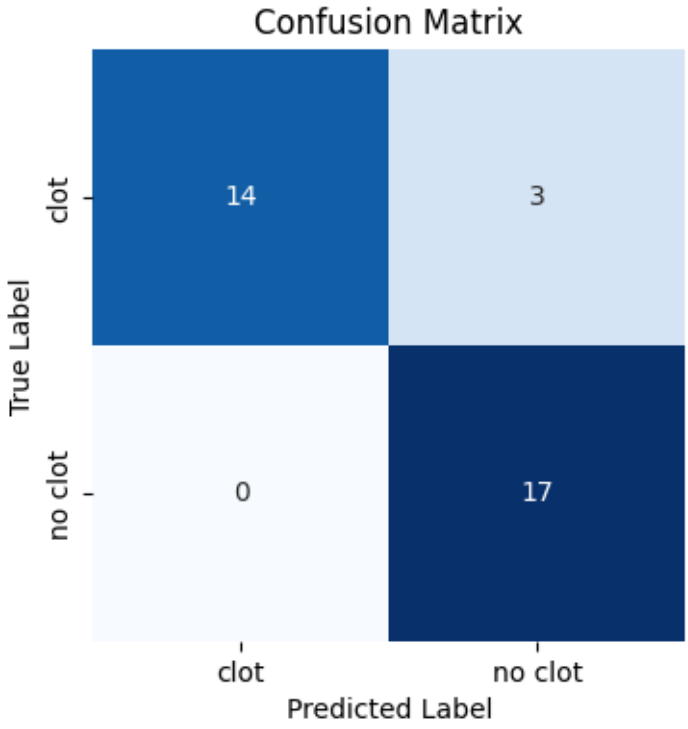

Figure 6.3 Confusion Matrix for DenseNet121

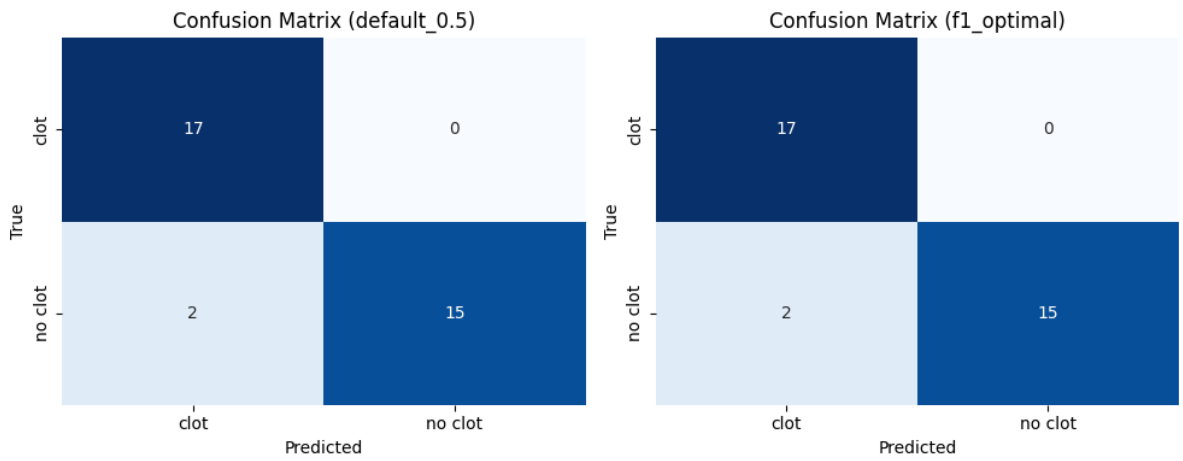

Figure 6.4 Confusion Matrix for DenseNet169

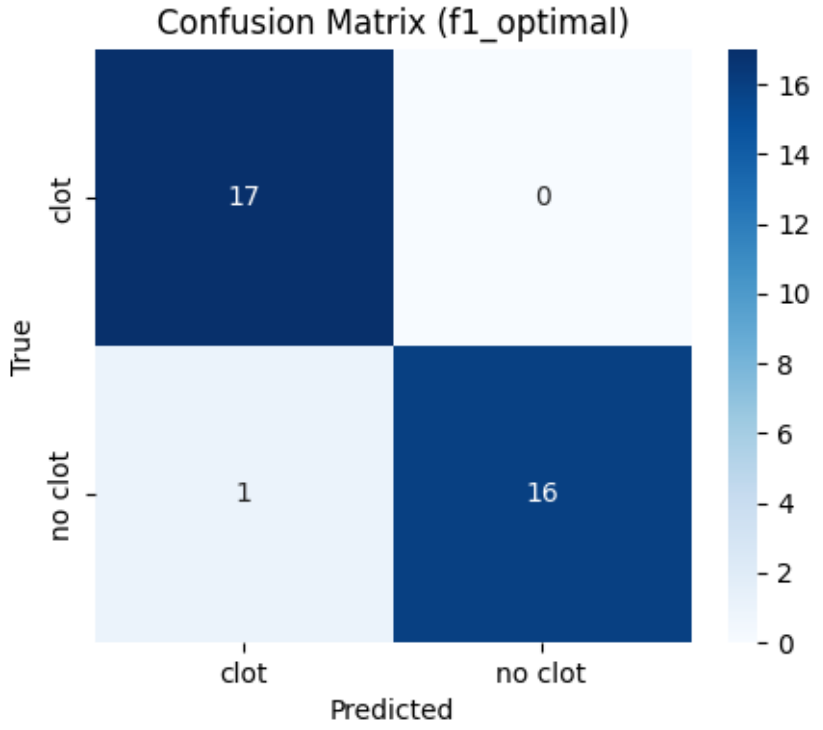

Figure 6.5 Confusion Matrix for EfficientNetB3

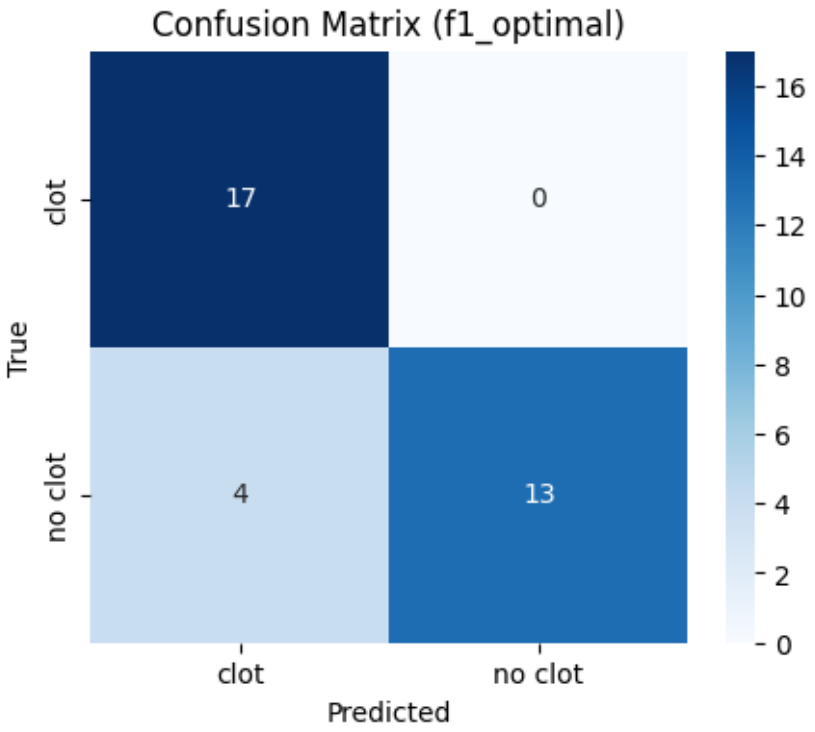

Figure 6.6 Confusion Matrix for ViT

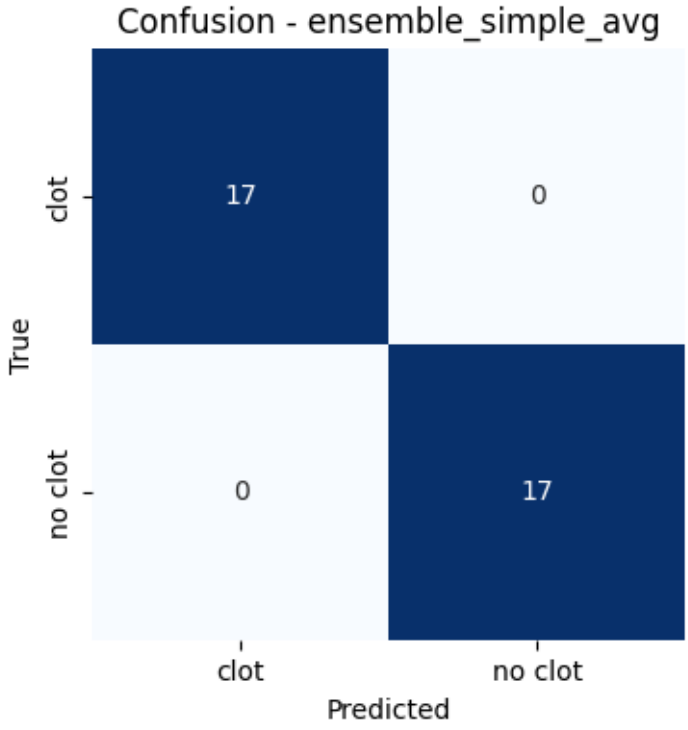

Figure 6.7 Confusion Matrix for Ensemble (Simple Average)

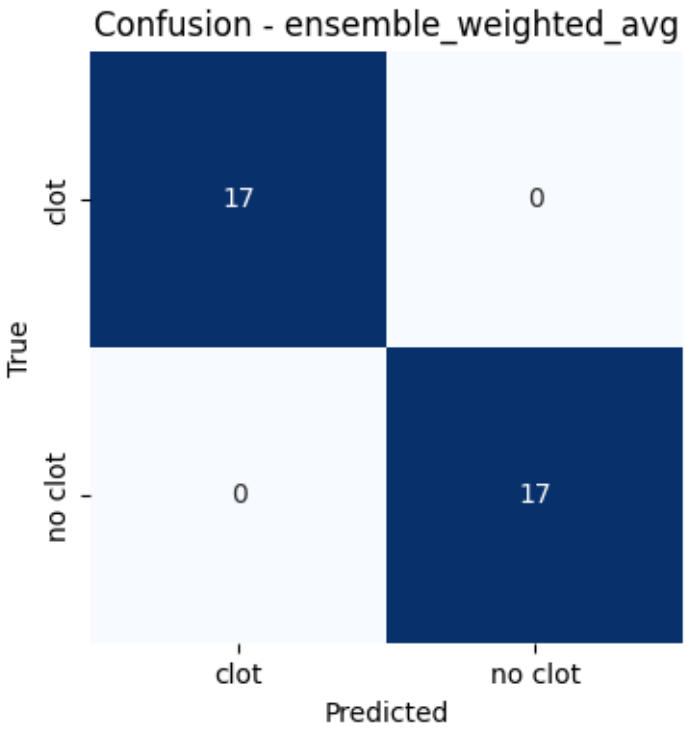

Figure 6.8 Confusion Matrix for Ensemble (Weighted Average)

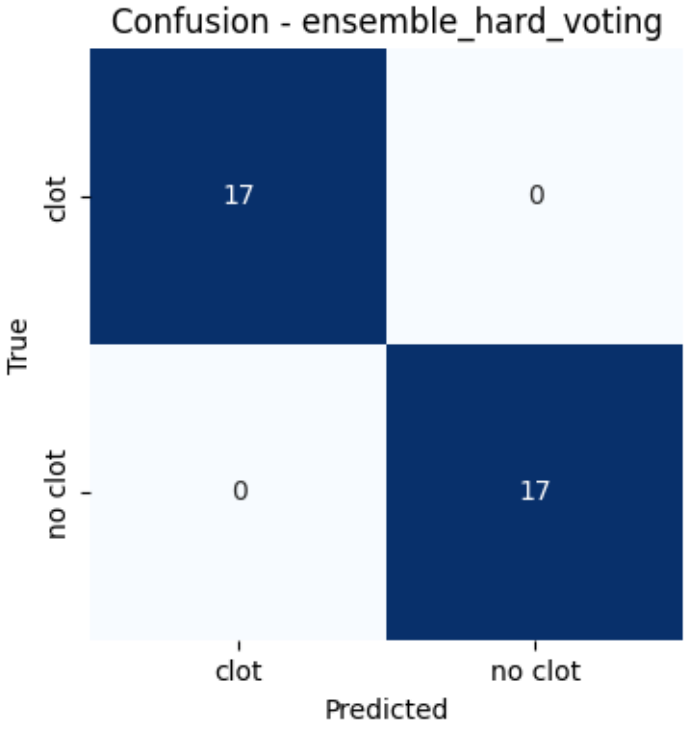

Figure 6.9 Confusion Matrix for Ensemble (Hard Voting)

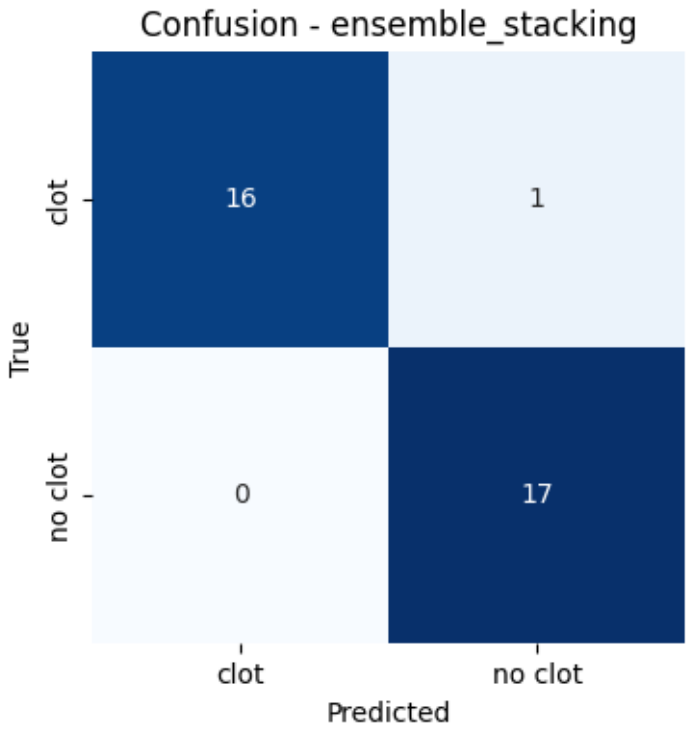

Figure 6.10 Confusion Matrix for Ensemble (Stacking)

The ROC curves for all models except Ensemble (Hard Voting), which doesn't output probabilities, are presented in Figures 6.11 to 6.19. The ROC-AUC for all models was greater than 0.98, and ResNet50, EfficientNet-B3, ViT, and all ensemble methods that output probabilities had a perfect ROC-AUC of 1.000. Ensemble (Hard Voting) was not given an ROC curve because it does not produce probabilities.

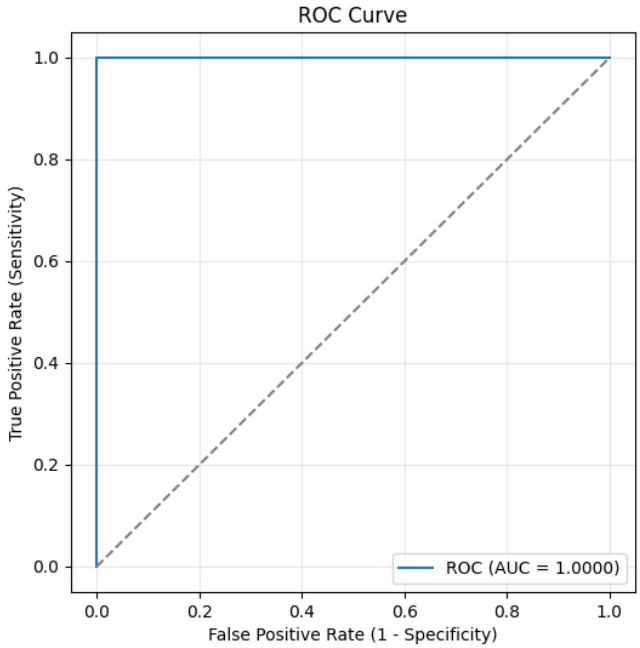

Figure 6.11 ROC Curve for ResNet50

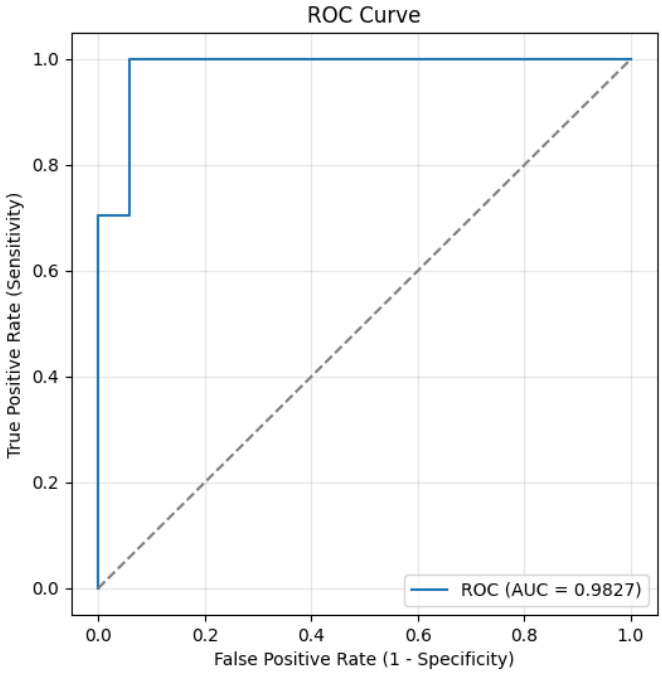

Figure 6.12 ROC Curve for ResNet101

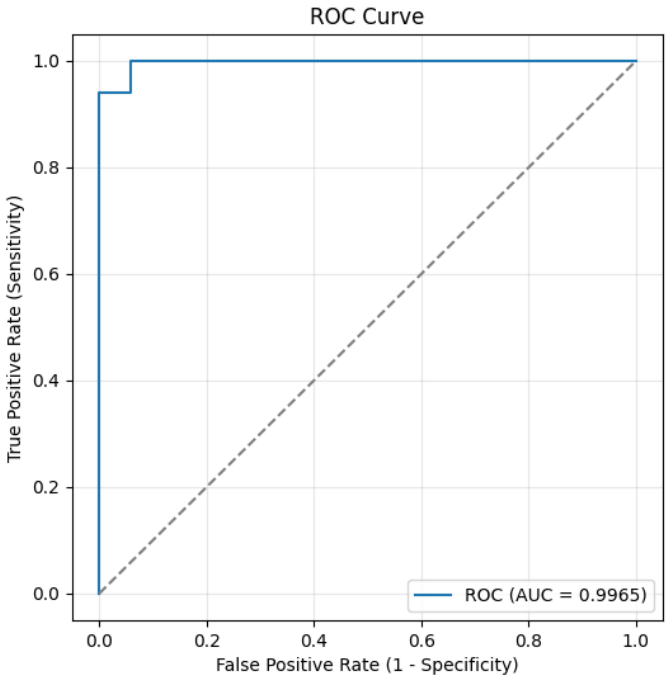

Figure 6.13 ROC Curve for DenseNet121

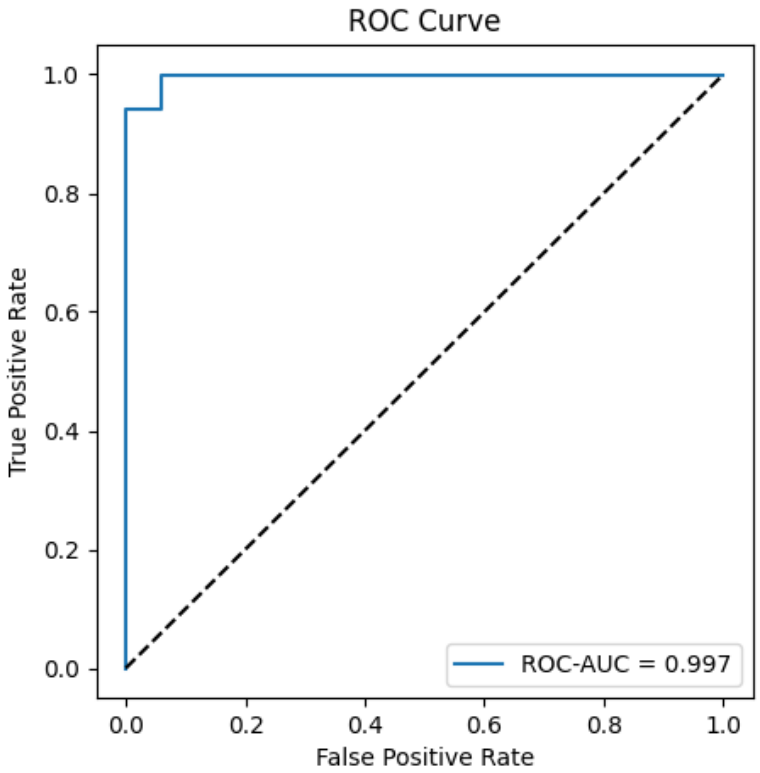

Figure 6.14 ROC Curve for DenseNet169

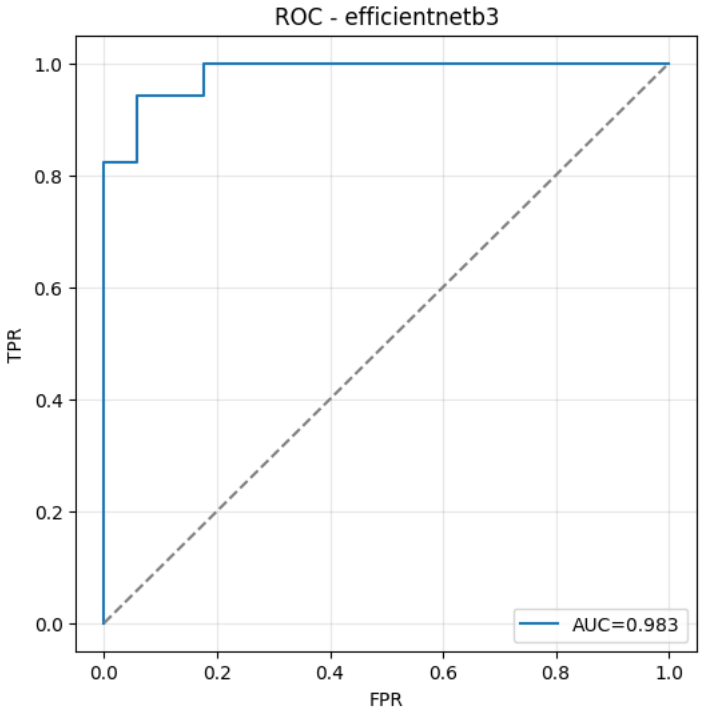

Figure 6.15 ROC Curve for EfficientNetB3

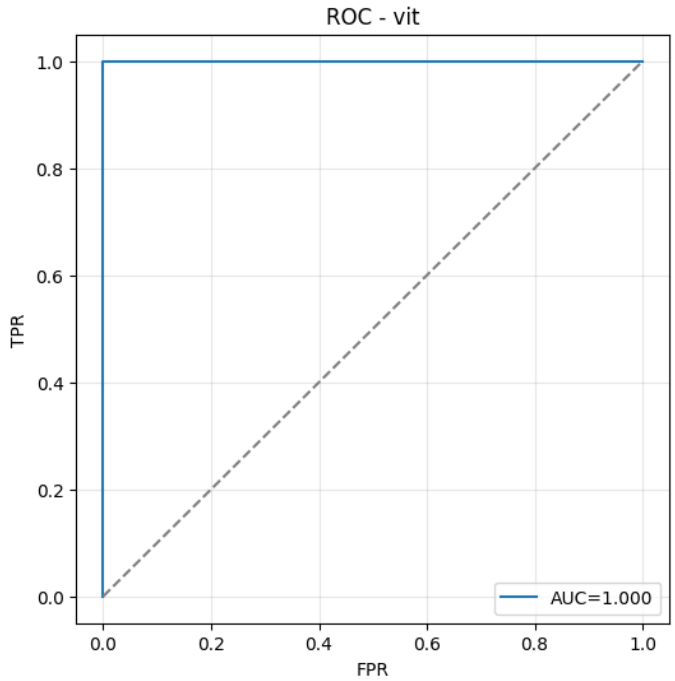

Figure 6.16 ROC Curve for ViT

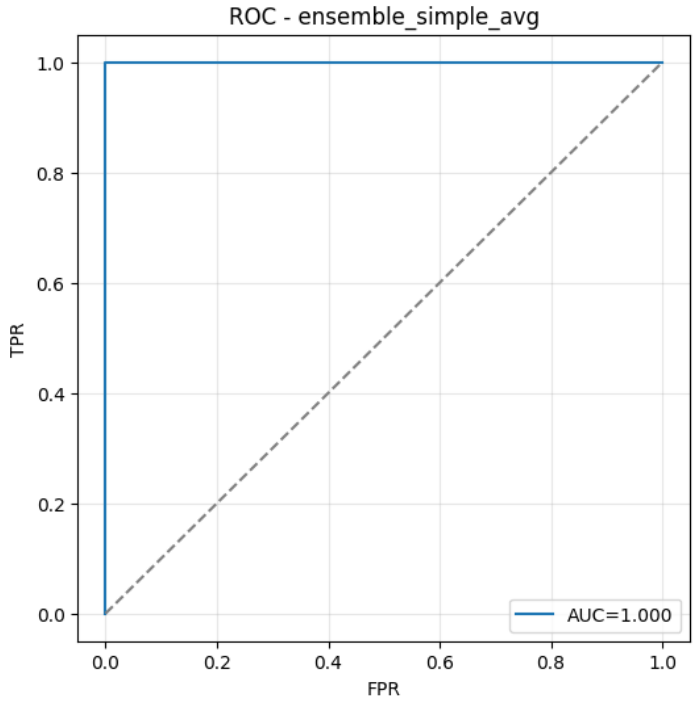

Figure 6.17 ROC Curve for Ensemble (Simple Average)

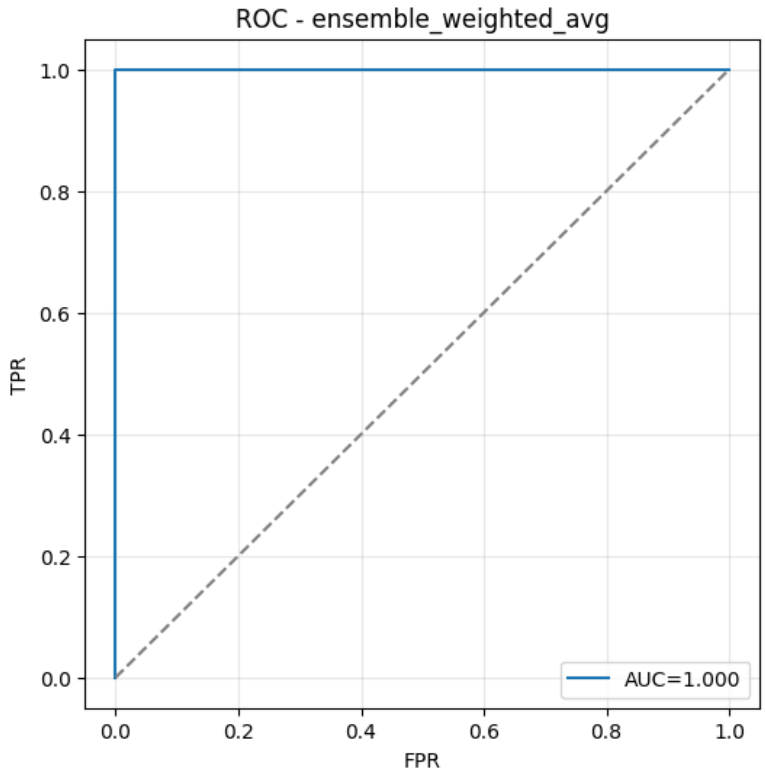

Figure 6.18 ROC Curve for Ensemble (Weighted Average)

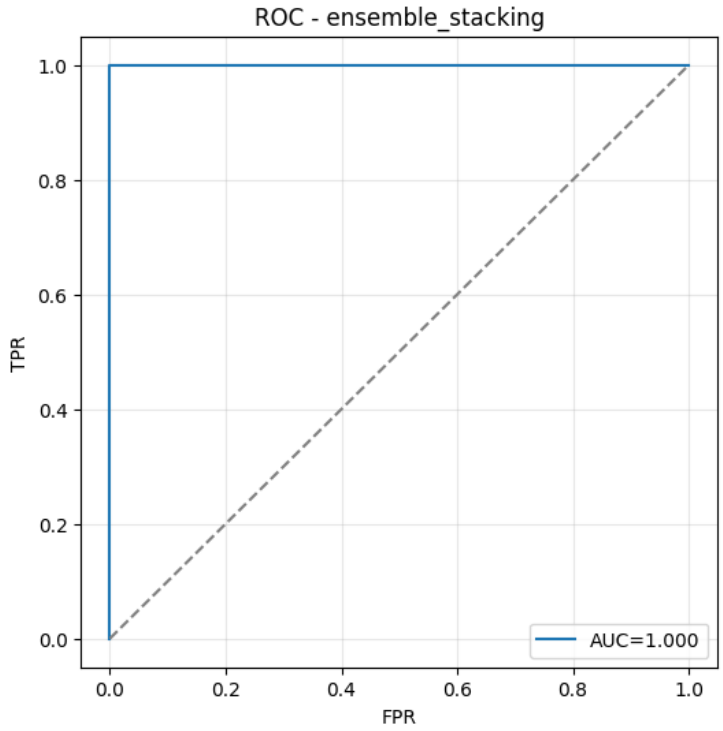

Figure 6.19 ROC Curve for Ensemble (Stacking)

Figures 6.20 to 6.23 show example heatmaps produced by Grad-CAM (for EfficientNet-B3) and Attention Rollout (for ViT) on clot and no clot MRI scans. The Grad-CAM heatmaps identified specific brain regions where the CNN focused on features relevant to a clot, while the Attention Rollout maps indicated more generalised attention on the MRI scan. The visualisations showed that the models' attention was drawn to relevant parts of the brain.

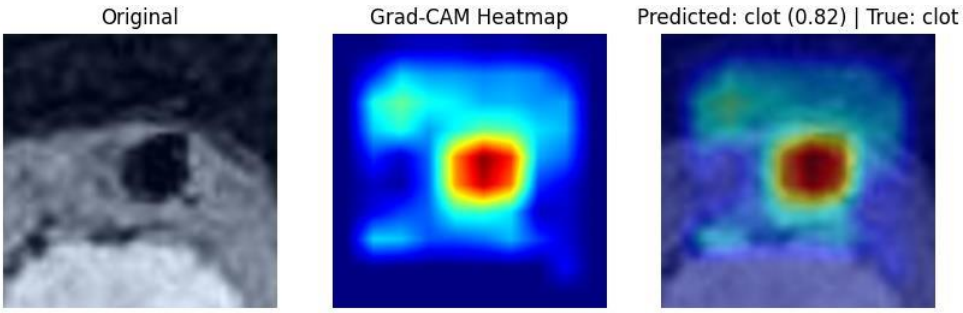

Figure 6.20 Grad-CAM Heatmap Examples (EfficientNetB3-Clot)

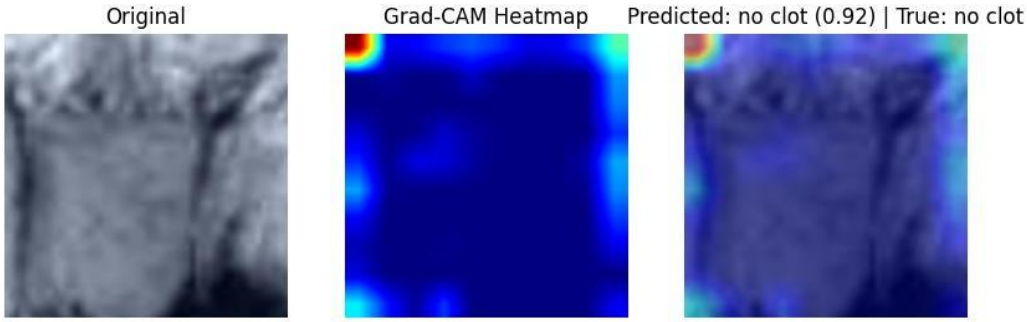

Figure 6.21 Grad-CAM Heatmap Examples (EfficientNetB3-No Clot)

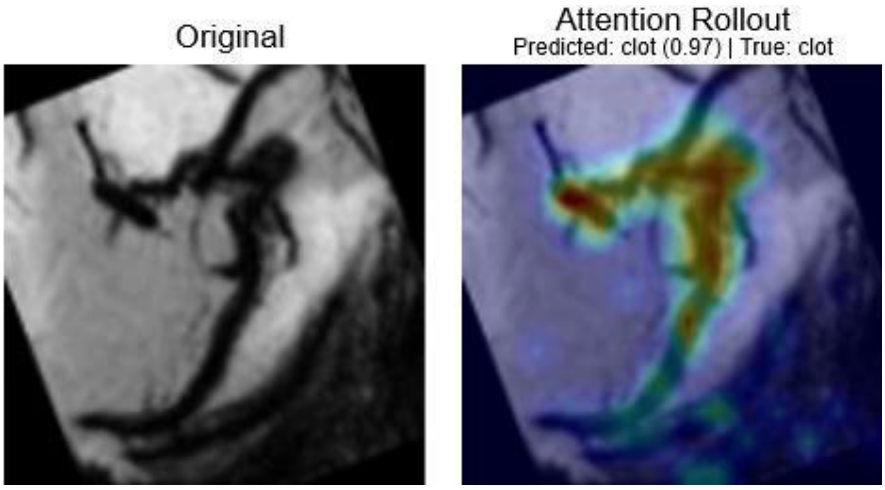

Figure 6.22 Attention Rollout Examples (ViT-Clot)

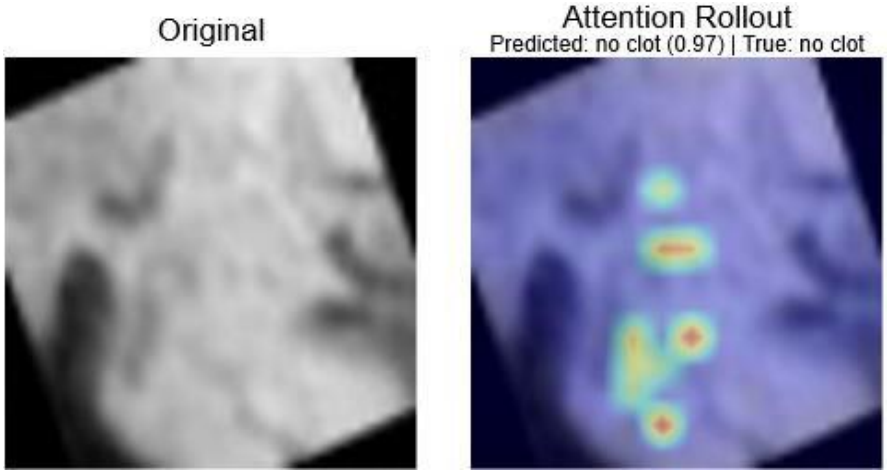

Figure 6.23 Attention Rollout Examples (ViT-No Clot)

6.4 Objectives Evaluation

This chapter assesses whether the four project goals outlined in Chapter 1 have been achieved.

Objective 1: To fine-tune and evaluate multiple state-of-the-art deep learning models.

This objective was met. This project fine-tuned six pre-trained models (ResNet50, ResNet101, DenseNet121, DenseNet169, EfficientNet-B3, and Vision Transformer) on the HPUPM MRI data. The models were thoroughly assessed using accuracy, precision, recall, F1-score, ROC-AUC, PR-AUC, confusion matrices, and calibration curves. ResNet50 achieved perfect accuracy (1.000) and the lowest-performing model (ViT) had an accuracy of 0.8824, surpassing the 0.85 accuracy target set in Chapter 3. But ViT's F1-score (0.8807) was slightly lower than the 0.90 target, due to its poor performance on the small HPUPM dataset (see Section 6.3). However, ViT demonstrated excellent discrimination ability across all decision thresholds with a perfect ROC-AUC of 1.000. Latency testing of predictions (see Section 6.2.4) confirmed that all models achieved the 10-second target for prediction latency, assuming they are deployed entirely on a CPU.

Objective 2: To develop an ensemble deep learning model.

This objective was successfully met. This project used four methods: simple averaging, weighted averaging, hard voting and stacking. The four ensemble methods all achieved accuracy greater than 0.90. Simple averaging, weighted averaging, and hard voting methods all reached 100% accuracy, only the stacking ensemble reached 0.9706 accuracy. The ensemble method capitalised on the complementary properties of CNNs and the Vision Transformer, as expected.

Objective 3: To implement explainable artificial intelligence (XAI).

This objective was fully achieved. Grad-CAM was implemented for the five CNNs, and Attention Rollout was implemented for the Vision Transformer. These methods produced heatmaps of the MRI image to show the parts of the image the model used to make a decision. Importantly, the XAI functionality was integrated into the web app rather than being an offline visualisation tool, enabling users to see heatmaps in conjunction with the model's prediction.

Objective 4: To deploy the ensemble model in a secure, user-friendly web application.

This objective was completely met. The project has built a Flask web application using Firebase Authentication and Firestore Database with three user roles (patient, doctor, admin) and

customised dashboards. The system featured a full doctor review and approval workflow with double notifications (bell icon and email), export to PDF, an AI chat bot built using Claude Haiku, and a robust security mechanism (Flask-Talisman CSP, Flask-Limiter rate limiting, session timeout, magic byte validation, prediction ownership, HTML sanitisation and blocked email domains). The app was successfully deployed on Hugging Face Spaces using Docker and can be accessed at https://dancinggdogg-neuroscan.hf.space.

Table 6.6 summarises the objectives evaluation.

Table 6.6 Objectives Evaluation Summary

#	Objective	Target	Status
1	Fine-tune and evaluate multiple deep learning models	Accuracy > 0.85	Achieved
2	Develop ensemble deep learning model	Accuracy > 0.90	Achieved
3	Implement explainable AI (Grad-CAM + Attention Rollout)	Heatmaps	Achieved
4	Deploy in secure, user-friendly web application	Live deploy	Achieved

6.5 Concluding Remark

This chapter provided a detailed assessment of the NeuroScan system. Functional testing showed all 38 test cases passed for authentication, prediction, dashboards, notifications, chatbot, settings and access control. Cross-browser testing on 90 test cases revealed no critical bugs on Chrome, Firefox, and Edge. Security testing confirmed 10 security measures were effective. Performance testing showed that ResNet50 and three ensemble models had 100% accuracy on the test set, and most models surpassed all the performance goals, with ViT just below the F1-score goal but with 100% ROC-AUC. The four project objectives were met. The study noted and discussed the limitations of the small test set and single dataset source. The following chapter concludes the report with findings and comments on how to move forward.

Chapter 7

Conclusion and Recommendation

7.1 Conclusion

The project successfully created NeuroScan, a web tool that uses an ensemble of deep learning models to detect ischemic stroke from brain MRI images. The project met the medical need for more reliable and efficient stroke diagnosis by integrating six state-of-the-art deep learning architectures with explainable AI, and embedding them in a secure, role-based web application.

During the model development phase, six pre-trained models were fine-tuned using a set of 15,120 MRI images from Hospital Pengajar Universiti Putra Malaysia (HPUPM). ResNet50 scored 1.000 accuracy on the test set of 34 images. ResNet101 and EfficientNet-B3 had 0.9706 accuracy, and DenseNet169 (0.9412) had higher accuracy than DenseNet121 (0.9118). The Vision Transformer model had an accuracy of 0.8824, which was in line with the literature that indicates transformers perform better with larger datasets than CNNs. The individual models all met the minimum accuracy threshold of 0.85.

Four different averaging techniques were considered: simple averaging, weighted averaging, hard voting, and stacking. Three of the four methods reached 1.000 accuracy, suggesting that the combination of different architectural styles led to more reliable predictions. Simple averaging was chosen for the final system because it was simple and performed as well as more sophisticated methods.

To enhance transparency and user understanding, Explainable AI (XAI) was implemented using Grad-CAM for the five CNN models and Attention Rollout for the Vision Transformer. Both produced heatmaps showing the parts of the MRI that were important to the prediction. While many studies only utilise XAI for offline visualisation, this project implemented both techniques in the web application to display heatmaps together with predictions.

The web application was built with Flask, Firebase Authentication, and Cloud Firestore, incorporating a three-level role-based access control (RBAC) for patients, doctors, and administrators. This system included a full clinical workflow: patients uploaded their MRI scans to receive AI predictions, doctors confirmed predictions with clinical comments, and administrators provided user management and monitored events via audit logs. Other features included a chatbot using Anthropic's Claude Haiku, a two-pronged notification system (in-app

bell with unread badge and email via Resend API), a PDF report generator (ReportLab), and a security stack with Flask-Talisman Content Security Policy (CSP) headers, Flask-Limiter rate limiting, session timeout with countdown timer, magic byte file verification, prediction ownership verification, and HTML sanitisation.

The system was hosted on Hugging Face Spaces using Docker and can be accessed at https://dancinggdogg-neuroscan.hf.space. The testing process verified that the 38 functional tests passed, 90 cross-browser tests had no critical issues on Chrome, Firefox or Edge, and the 10 security tests confirmed the success of the prevention measures. The four goals of the project were fully met.

But there were limitations to this project. The data were obtained from a single hospital (HPUPM) and the test set was small (n=34 images), so there were concerns about how well the model would generalise to other hospitals. The Hugging Face Spaces free plan limited the use of CPUs for inference, ephemeral storage (Grad-CAM images lost on restart), no SMTP (used a Resend API instead), and 5-10 minute "cold start" time for downloading model weights. CSRF protection was turned off due to incompatibilities with Firebase JS SDK. The system was not trialled in a clinical environment or with hospital workflows.

7.2 Recommendation

Following the lessons learnt from developing NeuroScan, there are several directions that could be taken. The key next steps to enhance the system are to test the models on MRI data from multiple centres or hospitals. This would give some indication of the models' generalisability and overcome the problem of using data from one site. Domain adaptation and federated learning could be investigated to enable models to be able to work across different imaging protocols and scanners.

This study used binary classification (clot vs no clot). It may be possible to explore this for stroke subtypes such as large vessel occlusion, lacunar infarction, cardioembolic stroke or for ischemic vs haemorrhagic. This would require that the dataset be larger with these labels.

At the time, manual functional testing was done. Automating these tests (using pytest for unit testing and Selenium for end-to-end tests) would make the tests more consistent and reliable, especially for regression testing after the code has changed. The current deployment is using the free tier of Hugging Face Spaces which does not support inference on GPU, and hence slower inference times. The inference times would be improved if this project can use GPU for

deployment, such as Hugging Face Spaces Pro, Amazon Web Services (AWS) or Google Cloud, especially for the ensemble-based predictions (which run six models on CPU).

Another way to improve the system is by improving how medical images are stored. As of now, user-uploaded MRI scans and Grad-CAM heatmaps are stored in temporary storage of the container, and will not persist after the container restarts. Storing medical images in Firebase Cloud Storage or external object storage systems (such as Amazon S3) would allow the system to keep images across re-deployments and container restarts.

For clinical use, the system would require clinical validation and medical regulation compliance via collaborations with medical institutions, regulatory approvals and medical privacy policies. It would also need to be integrated into hospital Electronic Health Record (EHR) and Picture Archiving and Communication System (PACS) systems.

The explanations from Grad-CAM and Attention Rollout were helpful but the system could also include other XAI techniques, such as SHAP (SHapley Additive exPlanations) or LIME (Local Interpretable Model-agnostic Explanations) or concept explanations to provide doctors with different types of interpretability to review.

The current system has an asynchronous doctor review process (patient submits, doctor reviews). New versions may provide real-time features to support clinical practices such as co-reviewing cases with multiple doctors, video conferencing and second-opinion requests.

REFERENCES

[1] World Health Organization, "The top 10 causes of death," World Health Organization, Aug. 07, 2024. [Online]. Available: https://www.who.int/news-room/fact-sheets/detail/the-top-10-causes-of-death

[2] V. L. Feigin et al., "World stroke organization (WSO): Global stroke fact sheet 2022," *International Journal of Stroke*, vol. 17, no. 1, pp. 18–29, Jan. 2022, doi: 10.1177/17474930211065917.

[3] J. L. Saver, "Time Is Brain—Quantified," *Stroke*, vol. 37, no. 1, pp. 263–266, Jan. 2006, doi: 10.1161/01.str.0000196957.55928.ab.

[4] J. N. D. Fernandes, V. E. M. Cardoso, A. Comesaña-Campos, and A. Pinheira, "Comprehensive Review: Machine and Deep Learning in Brain Stroke Diagnosis," *Sensors*, vol. 24, no. 13, p. 4355, 2024, doi: 10.3390/s24134355.

[5] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Las Vegas, NV, USA, 2016, pp. 770–778, doi: 10.1109/CVPR.2016.90.

[6] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely Connected Convolutional Networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2017, pp. 2261–2269, doi: 10.1109/cvpr.2017.243.

[7] M. Tan and Q. V. Le, "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks," in *Proc. 36th Int. Conf. Mach. Learn. (ICML)*, Long Beach, CA, USA, 2019, pp. 6105–6114.

[8] A. Dosovitskiy et al., "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2021. [Online]. Available: https://arxiv.org/abs/2010.11929.

[9] L. Breiman, "Bagging Predictors," *Machine Learning*, vol. 24, no. 2, pp. 123–140, 1996, doi: 10.1023/a:1018054314350.

[10] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, "Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Venice, Italy, 2017, pp. 618–626, doi: 10.1109/ICCV.2017.74.

[11] L. L. Quinn et al., "Interobserver variability studies in diagnostic imaging: a methodological systematic review," *British Journal of Radiology*, vol. 96, no. 1148, Jun. 2023, doi: 10.1259/bjr.20220972.

[12] H. Guan and M. Liu, "Domain Adaptation for Medical Image Analysis: A Survey," *IEEE Transactions on Biomedical Engineering*, vol. 69, no. 3, pp. 1–1, 2021, doi: 10.1109/tbme.2021.3117407.

[13] R. Mohammadi, I. Shokatian, M. Salehi, H. Arabi, I. Shiri, and H. Zaidi, "Deep learning-based auto-segmentation of organs at risk in high-dose rate brachytherapy of cervical cancer," *Radiotherapy and Oncology*, vol. 159, pp. 231–240, Jun. 2021, doi: 10.1016/j.radonc.2021.03.030.

[14] S. M. Thomas, J. G. Lefevre, G. Baxter, and N. A. Hamilton, "Interpretable Deep Learning Systems for Multi-Class Segmentation and Classification of Non-Melanoma Skin Cancer," *Medical Image Analysis*, vol. 68, no. 101915, p. 101915, Nov. 2020, doi: 10.1016/j.media.2020.101915.

[15] C. Wang et al., "Deep learning for predicting subtype classification and survival of lung adenocarcinoma on computed tomography," *Translational Oncology*, vol. 14, no. 8, p. 101141, Aug. 2021, doi: 10.1016/j.tranon.2021.101141.

[16] A. Anaya-Isaza, L. Mera-Jiménez, and M. Zequera-Diaz, "An overview of deep learning in medical imaging," *Informatics in Medicine Unlocked*, vol. 26, p. 100723, 2021, doi: 10.1016/j.imu.2021.100723.

[17] M. S. Choi et al., "Clinical evaluation of atlas- and deep learning-based automatic segmentation of multiple organs and clinical target volumes for breast cancer," *Radiotherapy and Oncology*, vol. 153, Sep. 2020, doi: 10.1016/j.radonc.2020.09.045.

[18] M. Ayoub, Z. Liao, S. Hussain, L. Li, C. W. J. Zhang, and K. K. L. Wong, "End to end vision transformer architecture for brain stroke assessment based on multi-slice classification and localization using computed tomography," *Computerized Medical Imaging and Graphics*, vol. 109, p. 102294, Oct. 2023, doi: 10.1016/j.compmedimag.2023.102294.

[19] B. R. Gaidhani, R. R. Rajamenakshi, and S. Sonavane, "Brain Stroke Detection Using Convolutional Neural Network and Deep Learning Models," in *Proc. 2nd Int. Conf. Intell. Comput. Control Syst. (ICICCS)*, 2019, doi: 10.1109/ICCONS.2019.8969052.

[20] D. M. Ibrahim, N. M. Elshennawy, and A. M. Sarhan, "Deep-chest: Multi-classification deep learning model for diagnosing COVID-19, pneumonia, and lung cancer chest diseases," *Computers in Biology and Medicine*, vol. 132, p. 104348, May 2021, doi: 10.1016/j.compbiomed.2021.104348.

[21] Z. Zhuang, Z. Yang, A. Noel, C. Wei, P. Jin, and S. Zhuang, "Breast ultrasound tumor image classification using image decomposition and fusion based on adaptive multi-model spatial feature fusion," *Computer Methods and Programs in Biomedicine*, vol. 208, pp. 106221–106221, Sep. 2021, doi: 10.1016/j.cmpb.2021.106221.

[22] A. Vaswani et al., "Attention Is All You Need," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 30, 2017.

[23] C. Chen, O. Li, C. Tao, A. J. Barnett, J. Su, and C. Rudin, "This looks like that: deep learning for interpretable image recognition," in *Proc. 33rd Annu. Conf. Neural Inf. Process. Syst. (NeurIPS)*, 2019.

[24] J. R. Zech, M. A. Badgeley, M. Liu, A. B. Costa, J. J. Titano, and E. K. Oermann, "Variable generalization performance of a deep learning model to detect pneumonia in chest radiographs: A cross-sectional study," *PLOS Medicine,* vol. 15, no. 11, p. e1002683, Nov. 2018, doi: 10.1371/journal.pmed.1002683.

[25] J. Luo, P. Dai, Z. He, Z. Huang, S. Liao, and K. Liu, "Deep learning models for ischemic stroke lesion segmentation in medical images: A survey," *Computers in Biology and Medicine*, vol. 175, pp. 108509–108509, Jun. 2024, doi: 10.1016/j.compbiomed.2024.108509.

APPENDIX

A.1 Cross-Browser Compatibility Test

NeuroScan — Cross-Browser Compatibility Test Report

FYP2 · Lee Ding Kuan · UTAR · 2026

Test URL: https://danciнggdogg-neuroscan.hf.space

Legend: ☑ PASS = Works as expected | ⚠ PARTIAL = Minor issue | ✗ FAIL = Broken/unusable | N/A = Not applicable

#	Category	Test Case	Chrome	Firefox	Edge	Notes / Issues
1. PAGE LOADING & LAYOUT						
1	**Page Load**	Home page loads without console errors	☑	☑	☑	
2	**Page Load**	Hero section renders correctly (gradient, stats, rings)	☑	☑	☑	
3	**Page Load**	Hero stats count-up animation plays on scroll	☑	☑	☑	
4	**Page Load**	Feature cards grid aligns properly	☑	☑	☑	
5	**Page Load**	Footer displays correctly with © 2026	☑	☑	☑	
6	**Page Load**	About page renders (tech tags, objectives, dataset info)	☑	☑	☑	
7	**Page Load**	Comparison page table renders with all 10 model rows	☑	☑	☑	
8	**Page Load**	404 error page renders when visiting invalid URL	☑	☑	☑	Subtle radial-gradient dot pattern on body::before — renders correctly but barely visible by design
9	**Page Load**	Dot-grid background texture visible on body	☑	☑	☑	Subtle radial-gradient dot pattern on body::before — renders correctly but barely visible by design
2. NAVIGATION & RESPONSIVE						
10	**Navigation**	Navbar sticky positioning works on scroll	☑	☑	☑	
11	**Navigation**	Navbar backdrop-filter blur effect visible	☑	☑	☑	
12	**Navigation**	Active nav link highlights correctly (blue pill)	☑	☑	☑	
13	**Navigation**	Brand icon gradient renders	☑	☑	☑	
14	**Navigation**	Favicon (brain SVG) loads in browser tab	☑	☑	☑	Nav links hide intentionally via CSS media query at ≤540px (display:none). No hamburger menu — users on small screens rely on dashboard links and direct URLs. Design trade-off, not a browser bug.
15	**Responsive**	Layout adapts at 768px breakpoint (mobile)	☑	☑	☑	Nav links correctly hidden at ≤540px as per CSS media query
16	**Responsive**	Layout adapts at 540px breakpoint (small mobile)	⚠	⚠	⚠	Nav links hide intentionally via CSS media query at ≤540px (display:none). No hamburger menu — users on small screens rely on dashboard links and direct URLs. Design trade-off, not a browser bug. App is designed for desktop/laptop use (clinical context).
17	**Responsive**	Navbar nav links hide on small screen (≤540px)	☑	☑	☑	Nav links correctly hidden at ≤540px as per CSS media query — intentional behaviour
3. AUTHENTICATION						
18	**Auth**	Login page split-layout renders (left branding + right form)	☑	☑	☑	
19	**Auth**	Email/password login works	☑	☑	☑	
20	**Auth**	Google OAuth popup opens and completes login	☑	☑	☑	Users can register with weak passwords (e.g. 8 chars, no uppercase). Firebase enforces minimum 6 chars; app enforces 8 chars. Strength bar is advisory only — no mandatory complexity rule beyond length. Known design decision.
21	**Auth**	Google OAuth role picker modal appears for new users	☑	☑	☑	
22	**Auth**	Register page renders with role selector (Patient/Doctor)	☑	☑	☑	
23	**Auth**	Email/password registration works	☑	☑	☑	Users can register with weak passwords (e.g. 8 chars, no uppercase). Firebase enforces min 6 chars; app enforces 8 chars. Strength bar is advisory only. Known design decision.
24	**Auth**	Password strength bar animates on input	☑	☑	☑	
25	**Auth**	Blocked email domain validation triggers (e.g. mailinator.com)	☑	☑	☑	
26	**Auth**	Logout redirects to login page	☑	☑	☑	
4. PREDICT PAGE						
27	**Predict**	File dropzone renders with drag-over highlight	☑	☑	☑	
28	**Predict**	MRI image preview shows after file selection (FileReader)	☑	☑	☑	
29	**Predict**	Preview clear (✕) button works	☑	☑	☑	
30	**Predict**	Model selector radio grid displays all 7 options	☑	☑	☑	
31	**Predict**	Ensemble option spans 2 columns with 'Best' badge	☑	☑	☑	
32	**Predict**	Loading overlay (spinner + blur) appears on submit	☑	☑	☑	
33	**Predict**	Result banner renders (green=no clot, red=clot)	☑	☑	☑	
34	**Predict**	Clot banner pulsing animation plays	☑	☑	☑	
35	**Predict**	Confidence bars animate from 0% to target width	☑	☑	☑	
36	**Predict**	Grad-CAM / Attention Rollout images display	☑	☑	☑	
37	**Predict**	XAI image download buttons work	☑	☑	☑	
38	**Predict**	Toast notification appears (bottom-right corner)	☑	☑	☑	
39	**Predict**	Confidence warning shows when prediction < 70%	☑	☑	☑	
40	**Predict**	'View in Dashboard' button navigates correctly	⚠	⚠	⚠	'View in Dashboard' button present in result banner but link redirection not working. Minor UX issue — user can navigate to dashboard via navbar.

51	**5. DASHBOARDS**						
52	41	**Patient**	Stat cards render (Total Scans, Reviewed, Pending, Clot)	☑	☑	☑	
53	42	**Patient**	Prediction history table loads with DataTables pagination	☑	☑	☑	
54	43	**Patient**	DataTables search/filter works	☑	☑	☑	
55	44	**Patient**	Colour-coded red-tint rows for clot predictions	☑	☑	☑	
56	45	**Patient**	Left red border accent on clot rows	☑	☑	☑	
57	46	**Patient**	'View Reviews' modal opens and shows doctor info	☑	☑	☑	
58	47	**Patient**	Export PDF button downloads a valid PDF file	☑	☑	☑	
59	48	**Doctor**	Pending cases table loads with DataTables	☑	☑	☑	
60	49	**Doctor**	Review modal opens with MRI images + metadata	☑	☑	☑	
61	50	**Doctor**	Agree/Disagree buttons submit review successfully	☑	☑	☑	
62	51	**Doctor**	Toast notification appears after review submission	☑	☑	☑	
63	52	**Admin**	All 3 tabs switch correctly (Users, Predictions, Logs)	☑	☑	☑	
64	53	**Admin**	User role doughnut chart renders (Chart.js)	☑	☑	☑	
65	54	**Admin**	Prediction results bar chart renders (Chart.js)	☑	☑	☑	
66	55	**Admin**	Update role dropdown + button works	☑	☑	☑	
67	56	**Admin**	Delete user modal confirms and deletes	☑	☑	☑	
68	57	**Admin**	Delete prediction modal confirms and deletes	☑	☑	☑	
69	**7. NEUROSCAN AI CHATBOT (CLAUDE HAIKU)**						
70	58	**Notif**	Bell icon shows in navbar (patient role only)	☑	☑	☑	
71	59	**Notif**	Unread badge count displays correctly	☑	☑	☑	
72	60	**Notif**	Dropdown opens on bell click	☑	☑	☑	
73	61	**Notif**	Mark individual notification as read works	☑	☑	☑	
74	62	**Notif**	'Mark all read' button works	☑	☑	☑	
75	63	**Notif**	Email notification received (via Resend / admin inbox)	☑	☑	☑	
76	**7. NEUROSCAN AI CHATBOT (CLAUDE HAIKU)**						
77	64	**Chatbot**	Floating 🤖 button visible (bottom-right, authenticated only)	☑	☑	☑	
78	65	**Chatbot**	Chat panel opens/closes on button click	☑	☑	☑	
79	66	**Chatbot**	Welcome message displays on first open	☑	☑	☑	
80	67	**Chatbot**	User message appears in blue bubble (right-aligned)	☑	☑	☑	
81	68	**Chatbot**	Typing indicator (3 dots) animates while waiting	☑	☑	☑	
82	69	**Chatbot**	AI response appears in white bubble (left-aligned)	☑	☑	☑	
83	70	**Chatbot**	Shift+Enter creates newline, Enter sends message	☑	☑	☑	
84	71	**Chatbot**	Conversation history maintained within session	⚠	⚠	⚠	Chat history resets on page navigation — expected behaviour since conversation state is held in JS memory, not persisted server-side. Consistent across all browsers. Known limitation.
85	**8. SESSION & SECURITY**						
86	72	**Session**	Session timeout warning appears after ~29 min inactivity	☑	☑	☑	
87	73	**Session**	Countdown timer counts down from 60	☑	☑	☑	
88	74	**Session**	'Stay logged in' button resets session	☑	☑	☑	
89	75	**Session**	Auto-logout occurs when countdown reaches 0	☑	☑	☑	
90	76	**Security**	Flask-Talisman CSP headers present (check DevTools → Network)	☑	☑	☑	
91	77	**Security**	Rate limiting triggers after 10 predictions/min (429 error)	☑	☑	☑	Rate limiting configured at 10 predictions/min via Flask-Limiter. Endpoint returns HTTP 429 when exceeded.
92	78	**Security**	Non-image file upload rejected (magic byte check)	☑	☑	☑	
93	79	**Security**	Unauthorised role access redirects (e.g. patient → admin dashboard)	☑	☑	☑	
94	**9. SETTINGS PAGE**						
95	80	**Settings**	Avatar + account info card renders	☑	☑	☑	
96	81	**Settings**	Display name update works	☑	☑	☑	
97	82	**Settings**	Password change works (email/password users)	☑	☑	☑	
98	83	**Settings**	Google user sees info note about Google-managed password	☑	☑	☑	
99	84	**Settings**	Password strength bar works on new password input	☑	☑	☑	Strength bar is advisory — no mandatory complexity enforcement beyond 8-char minimum.
100	**10. CSS & ANIMATIONS**						
101	85	**CSS**	DM Sans font loads correctly	☑	☑	☑	
102	86	**CSS**	Card fadeUp animation plays on load	☑	☑	☑	
103	87	**CSS**	Stat card staggered animation (delays) visible	☑	☑	☑	
104	88	**CSS**	Button hover states work (translateY, shadow changes)	☑	☑	☑	
105	89	**CSS**	Modal backdrop blur + scale transition works	☑	☑	☑	
106	90	**CSS**	Scrollbar styling (if custom) works	☑	☑	☑	

107		
108	**SUMMARY**	
109	**Total Test Cases**	90
110	**Chrome — Pass / Partial / Fail**	☑ 86 / ⚠ 4 / ✕ 0 (N/A: 0)
111	**Firefox — Pass / Partial / Fail**	☑ 86 / ⚠ 4 / ✕ 0 (N/A: 0)
112	**Edge — Pass / Partial / Fail**	☑ 86 / ⚠ 4 / ✕ 0 (N/A: 0)
113	**Overall Compatibility Rating**	Excellent — 94.4% full pass rate. No browser-specific failures. 0 critical issues. Minor responsive trade-offs at small screens (app designed for desktop/laptop clinical use).
114	**Tested By**	Lee Ding Kuan
115	**Test Date**	4/23/2026
116	**Browser Versions Used**	Edge: Version 147.0.3912.72 Chrome: Version 147.0.7727.117 Firefox: 149.0.2

A.2 Poster

UNIVERSITI TUNKU ABDUL RAHMAN · FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY
Development of An Ensemble Deep Learning Model for Brain Stroke Detection and Classification in MRI Scans
Lee Ding Kuan
BIT (Hons) Communications and Networking
Supervisor: Puan Nur Liyana Shamfaja
Bt Abdashah
May 2026
INTRODUCTION
Ischemic stroke accounts for over 62% of all strokes and requires rapid diagnosis from MRI scans. This project developed NeuroScan — an ensemble deep learning web application that detects ischemic stroke from brain MRI with explainable AI and a secure three-tier clinical workflow.
OBJECTIVES
• Fine-tune and evaluate 6 deep learning models for MRI-based stroke detection
• Develop ensemble models using 4 combination methods
• Implement explainable AI (Grad-CAM + Attention Rollout)
• Deploy in a secure, role-based web application
DATASET
MRI scans from Hospital Pengajar UPM (HPUPM), preprocessed to 224×224 with ImageNet normalisation and data augmentation.
15,120 TRAINING
38 VALIDATION
34 TEST
SYSTEM ARCHITECTURE
FRONTEND LAYER
Jinja2 Templates · Custom CSS · Vanilla JavaScript
DataTables.js · Chart.js
Patient
Doctor
Admin
HTTP Req/Res
BACKEND LAYER (Flask)
ML Inference
6 Models + Ensemble
XAI Engine
Grad-CAM · Attn Rollout
Services
Chatbot · PDF · Notif
Security: CSP · Rate Limiting · Session Timeout · Magic Bytes · RBAC
Read / Write
DATA & EXTERNAL SERVICES
Firebase Auth
User identity
Firestore DB
5 collections
HF Model Repo
697 MB · 7 models
APIs
Anthropic · Resend
Deployed via Docker on Hugging Face Spaces
MODEL PERFORMANCE
Model | Acc | Prec | Rec | F1 | ROC
ResNet50 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000
ResNet101 | 0.9706 | 0.9722 | 0.9706 | 0.9706 | 0.9827
DenseNet121 | 0.9118 | 0.9250 | 0.9118 | 0.9118 | 0.9965
DenseNet169 | 0.9412 | 0.9474 | 0.9412 | 0.9410 | 0.9970
EfficientNet-B3 | 0.9706 | 0.9722 | 0.9706 | 0.9706 | 1.000
ViT | 0.8824 | 0.9048 | 0.8824 | 0.8807 | 1.000
Ens. (Simple Avg) | 1.000 | 1.000 | 1.000 | 1.000 | 1.000
Ens. (Weighted) | 1.000 | 1.000 | 1.000 | 1.000 | 1.000
Ens. (Hard Vote) | 1.000 | 1.000 | 1.000 | 1.000 | —
Ens. (Stacking) | 0.9706 | 0.9722 | 0.9706 | 0.9706 | 1.000
ResNet50 and 3 ensemble methods achieved perfect accuracy (1.000). All models exceeded the 0.85 target. ViT's lower accuracy (0.8824) is consistent with literature on small-scale datasets.
EXPLAINABLE AI
Grad-CAM highlights clot-relevant regions for CNN models. Attention Rollout generates patch-level attention maps for ViT. Both are embedded in the web interface for real-time clinical interpretation alongside predictions.
Original
Grad-CAM Heatmap
Grad-CAM · Clot (0.82 confidence)
Original
Attention Rollout
Attention Rollout · Clot (0.97 confidence)
KEY FEATURES
6 DL Models + Ensemble
Doctor Review Workflow
AI Chatbot (Claude Haiku)
CSP · Rate Limit · Timeout
Grad-CAM & Attn Rollout
In-app + Email Notifications
PDF Report Export
Docker on HF Spaces
SYSTEM WORKFLOW
Start
Patient uploads MRI scan
Select model · Preview · Submit
Ensemble inference + XAI
6 models · Grad-CAM · Attention Rollout
Patient views result + heatmap
Confidence bars · Status: pending review
Doctor reviews & submits decision
Agree / Disagree · Clinical remarks
Patient receives notification
In-app bell + email (Resend API)
End
Patient
Upload · View · Export
Doctor
Review · Decide · Note
Admin
Manage · Logs · Charts
TESTING & EVALUATION
Functional 38/38 ✓
Browser 0 fail ✓
Security 10/10 ✓
CONCLUSION
NeuroScan combined six deep learning architectures into an ensemble framework, achieving perfect accuracy (1.000) with three methods. Deployed publicly with real-time XAI, clinical workflow, and comprehensive security. All four objectives fully achieved.
OBJECTIVES STATUS
Obj 1: 6 models evaluated ✓
Obj 2: Ensemble developed ✓
Obj 3: XAI implemented ✓
Obj 4: System deployed ✓