CHESTLENS AI

Design Document

Group 18

Supervisor	
Name: **Email:** **Organization:**	Dr. Mehdi Moradi moradm4@mcmaster.ca McMaster University, CAS
Team Member 1	
Name: **Email:** **Student Number:**	Abhishek Sharma shara109@mcmaster.ca 400322503
Team Member 2	
Name: **Email:** **Student Number:**	Anthony Vu vua11@mcmaster.ca 400306059
Team Member 3	
Name: **Email:** **Student Number:**	Hussein Saad saadh@mcmaster.ca 400307995
Team Member 4	
Name: **Email:** **Student Number:**	Nathan Starr starrn@mcmaster.ca 400323095
Team Member 5	
Name: **Email:** **Student Number:**	Yuvraj Jain jainy3@mcmaster.ca 400259484

Table of Contents

1.0 Revision History

Version Number	Authors	Description	Date
0	• Abhishek Sharma • Anthony Vu • Hussein Saad • Nathan Starr • Yuvraj Jain	Started Initial Draft	January 15th, 2024
1	• Abhishek Sharma • Anthony Vu • Hussein Saad • Nathan Starr • Yuvraj Jain	Updated document for final submission.	March 27th, 2024

2.0 Introduction

2.1 Project Purpose

Currently, radiographers facilitate the imaging process for X-rays and radiologists manually analyze the produced medical images to find any abnormalities, injuries, or diseases. This can be very time-consuming and like all things has the possibility of errors. To mitigate this issue we are implementing an AI model to identify six common findings. These diseases include Atelectasis, Cardiomegaly, Consolidation, Edema, No Finding, and Pleural Effusion. The main objective of using an AI model on chest X-rays is to identify negative results or identify which of these diseases are present, the predictions of these diseases, the location of these diseases with the use of visual mapping. We believe that creating an AI for chest X-rays is beneficial since it can be used by radiologists to help confirm or identify any abnormalities, injuries, and diseases present in X-ray images. Also, having the assistance of an AI model will save radiologists time and save hospitals money and resources since the analysis process will be sped up and a diagnosis can be given to patients faster.

2.2 Document Purpose

The purpose of this document is to provide a detailed breakdown and description of all components of the system, and illustrate the relationship between the components and requirements defined in the Software Requirements Specification (SRS) document.

2.3 Project Scope

The application will have a frontend website that allows radiologists to input X-ray images to the AI model and see an output. The output will include patient information, and predictions for Atelectasis, Cardiomegaly, Consolidation, Edema, No Finding, and Pleural Effusion, and visual mappings on the inputted X-ray image to show the location of diseases that are present.

2.4 Assumptions and Invariants

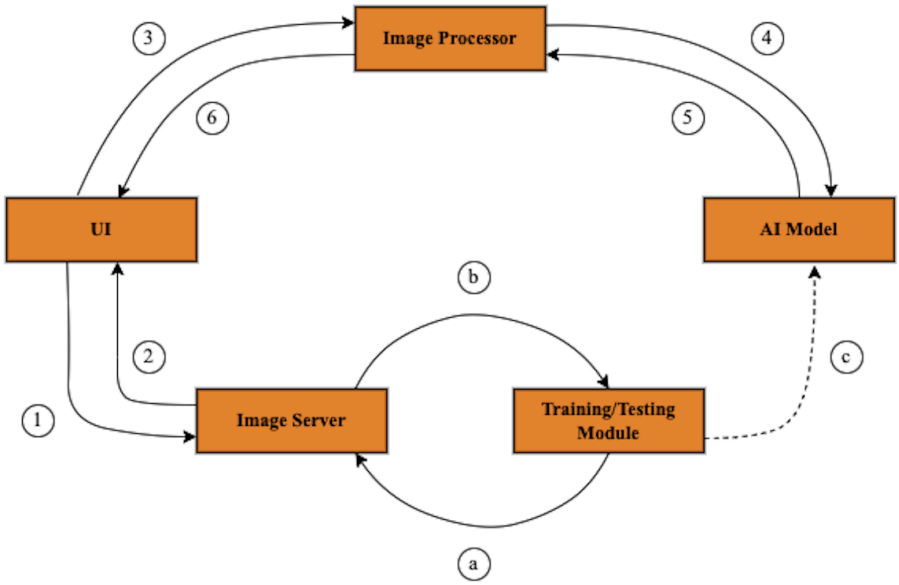
2.4.1 Assumptions

- Our model is being used by trained radiologists who are able to understand and verify model outputs.
- Radiologists make the final diagnosis after viewing the model output.
- X-rays are being inputted into the frontend website in DICOM format so that patient information and X-ray information can be extracted and displayed.
- The user has access to a strong and stable internet connection to ensure a seamless user experience on the application website.

2.4.2 Invariants

- Introduction of our software should not affect how X-rays are taken.
- How images are stored by the hospital should not be impacted by our product.
- The use of our software should not affect hospital X-ray image capturing procedures.

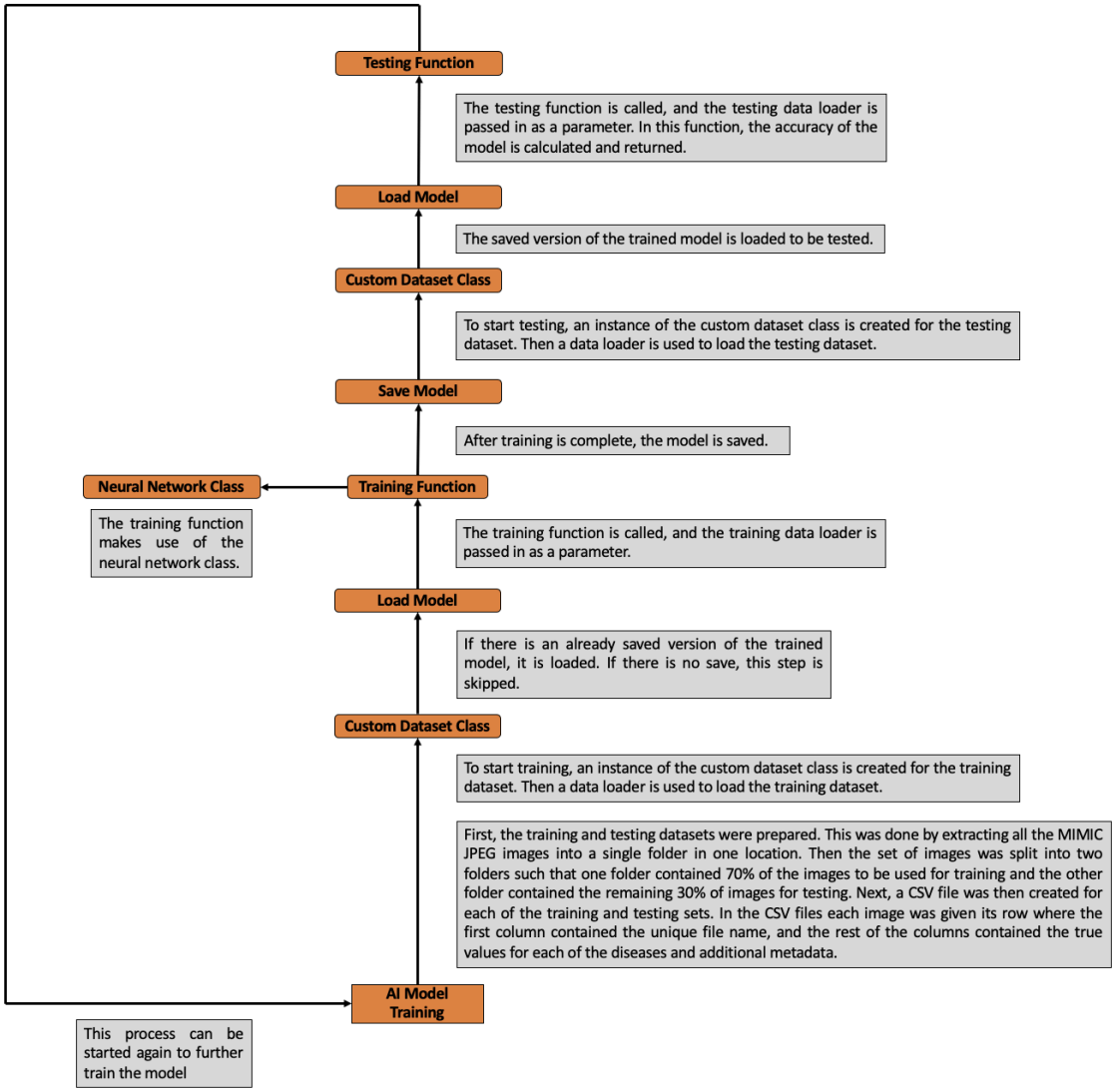
3.0 Project Component Diagram



1) An image is requested from the image server.
2) The requested image is sent to the UI.
3) The image is sent to the image processor to be prepared for the model. Image preparation includes converting the image from DICOM format to JPEG format, fixing the color channels, resizing the image, and converting the image to a tensor.
4) The processed image is sent to the model.
5) The results from the model are sent back to the image processor to be prepared for the UI.
6) The results are reformatted to be displayed on the UI.

a) The training/testing images and the corresponding CSV files are requested from the image server.
b) The requested images and files are sent to the training module.
c) When needed, the model is trained and then deployed.

AI Model Training Flowchart



4.0 Relationship Between Project Components and Requirements

System Component	Requirements Covered
UI	• P0: Provide patient information, implemented as a website, output AI model analysis, and include a comment section. • P3: Display previous X-rays. • Nonfunctional: Security and Privacy, Look and Feel, and Usability and Humanity.
Image Server	• P0: Compatible with DICOM images. • P2: Secure method of pushing/pulling

	images. ● Nonfunctional: Security and Privacy.
Image Processor	● P0: Compatible with DICOM images. ● Nonfunctional: Performance and Speed, and Operational and Environmental.
AI Model	● P0: Identify 6 findings (Atelectasis, Cardiomegaly, Consolidation, Edema, No Finding, and Pleural Effusion), ROC curve requirements, and compatibility with DICOM images. ● P1: Highlight affected areas, and have the model created/trained from scratch. ● P3: Color gradients. ● Nonfunctional: Performance and Speed, Operational and Environmental, Security and Privacy, and Legal.
Training/Testing Module	● P0: Identify 6 findings (Atelectasis, Cardiomegaly, Consolidation, Edema, No Finding, and Pleural Effusion), and ROC curve requirements. ● Nonfunctional: Security and Privacy, and Legal.

5.0 Project Components

5.1 UI

5.1.1 Normal Behaviour

The user (i.e., a radiologist) is able to login and see a list of their corresponding patients. Once a patient is selected, the user can then upload DICOM images, see the AI model's results in a table format, and add comments.

5.1.2 API and Structure

Inputted DICOM images should be sent to the Image Processor which involves converting the image from DICOM format to JPEG format, fixing the color channels, resizing the image, and converting the image to a tensor. The Appendix further details the database schemas.

5.1.3 Implementation

The technologies used were HTML, CSS, and JavaScript. When an image is uploaded, an event handler called loadFile displays the image. When the submit button is pressed, the upload() method from our backend is run which calls dcm_to_json(file_path, weights, mimix_csv), our main function that calls all our ML functions and sends the result in a JSON format for displaying.

5.1.4 Potential Undesired Behaviors

The user may input the wrong username, password, or an image of the wrong format (JPEG, PNG, etc.). Also, there are security risks around the lack of password security due to no use of encryption. In terms of reliability, the hosted website could have some downtime or slower responses due to high traffic. Another foreseeable issue is our website failing our unit tests.

5.2 Image Server

5.2.1 Normal Behaviour

The image server is responsible for managing the storage and retrieval of X-ray images. Through an API, it will respond to requests for storing, retrieving and managing images.

5.2.2 API and Structure

The API for the Image Server will include functions for uploading, downloading, and managing images requests to the image server.

5.2.3 Implementation

In our current configuration, all uploaded DICOM images will come from the user's local machine. Once uploaded, a JPG version of the image and the patient information from the DICOM header will be stored on a hosted MySQL database. On the front end, users can then select among these past scans and run it on either Prebuilt or Our Model.

5.2.4 Potential Undesired Behaviors

Despite login authentication, potential security risks surrounding DICOM images on local machines and the MySQL database are possible. Also, a very slow internet could cause performance issues given that uploading is an integral part of the application. Internet or database downtime is another potential issue.

5.3 Image Processor

5.3.1 Normal Behaviour

Image preparation includes converting the image from DICOM format to JPEG format, fixing the color channels, resizing the image, and converting the image to a tensor. The image processor is also responsible for passing the model output of disease predictions and color gradients to the front end.

5.3.2 API and Structure

Images received from the UI undergo processing within the Image Processor component. A POST API request is initiated to the AI Model. The result is a JSON output with data including disease predictions and color gradient images.

5.3.3 Implementation

We make use of Flask to implement a Python microservice, first the DICOM image is received and converted to a tensor. Then, the tensor is put into a JSON file. Next, a POST API call is used to send the JSON to the model to be processed. Lastly, a JSON file containing all the results from the AI model is returned.

5.3.4 Potential Undesired Behaviors

The image is incompatible to run through the AI model, is not a chest X-ray, or is corrupted.

5.4 AI Model

5.4.1 Normal Behaviour

The model accepts inputted tensors of chest x-rays and runs them through a convolutional neural network to identify the locations and probabilities of 6 findings (Atelectasis, Cardiomegaly, Consolidation, Edema, No Findings, and Pleural Effusion).

5.4.2 API and Structure

Contains the following functions:

dcm_patient_data(path_file): Input is a string of the file path to a DICOM image and returns a dictionary of patient information contained in the DICOM image.

run_model(file_tensor, weight_string): Input is a tensor of an X-ray and returns a dictionary containing a tag for (Atelectasis, Cardiomegaly, Consolidation, Edema, No Finding, and Pleural Effusion).

create_json(original_dict): Accepts a list of dictionaries and returns a JSON.

load_checkpoint(checkpoint, model, optimizer): Input is a string to a .pth.tar file, a model, and an optimizer. It then loads the stored model and optimizer from the .pth.tar into the inputted model and optimizer.

create_densenet121(num_classes): Is used for defining our neural network; it is a convolutional neural network. The model can be run after loading weights and accepts a tensor.

5.4.3 Implementation

The model starts with the file path to the DICOM image as well as that image as a processed tensor. First, dcm_patient_data is run to get the patient information out of the DICOM image. Next, run_model is called and the trained model is loaded using load_checkpoint with the weights obtained during training. Once loaded, the inputted tensor is run through our model. Lastly, the outputs of the model are combined with patient data using the create_json function to create a JSON file. This file is then sent to the image processor.

5.4.4 Potential Undesired Behaviors

A potential undesired behavior are false positives where the model returns that a patient has a disease that they do not have. The model may also return false negatives where the model returns that a patient does not have a certain disease when they actually do. False negatives are of more concern when compared to false positives since not identifying diseases in time can have fatal consequences. Also, the model could have a slow response time of over 10 seconds, and the server hosting the model could be down.

5.5 Training/Testing Module

5.5.1 Normal Behaviour

The training and testing module for the AI model should allow the model trainers/testers to train the model on a specific set of JPEG images and allow them to test the accuracy of the model on a specific set of testing JPEG images.

5.5.2 API and Structure

The training/testing module of the project consists of four main API/structures. These include a custom dataset class called "MIMIC_Dataset", the actual model itself defined in a function called "create_densenet121", the training module defined in a function called "train_model" and lastly the testing module defined in a class called "analyze_model". All of these APIs and structures are described in more detail in section 5.5.3 below.

5.5.3 Implementation

To prepare the dataset for the training of the AI model, all of the JPEG images from the MIMIC dataset are uploaded to Google Drive and the department server. Then, the set of images is split into three folders such that one folder contains 70% of the images to be used for training, the second folder is used for validation and contains 10% of the images and the final folder contains the remaining 20% of images for testing. A CSV file exists for each of the datasets containing the true disease values for each image and additional metadata.

There is a custom dataset class called "MIMIC_Dataset(Dataset)". This class takes in a file path to the CSV file for the dataset, the root directory for the images in the dataset, and a transformation option. In the class, the color channels for the images are corrected, the images are resized, and the images are converted to a tensor. Next, this class extracts the true values for the image from the CSV file and then turns the true values into a tensor. Lastly, this class makes a (image, label) tuple for each item in the dataset.

Next, there is the actual model itself, it is defined in a function called "create_densenet121". The model is a convolutional model and it is constantly being modified/tweaked based on the accuracy results produced from the model validation to ensure that higher accuracy can be achieved.

To start training, an instance of the custom dataset class is created for the training dataset by passing in the path to the CSV file for the training dataset, the root directory of the training images and "transforms.ToTensor()" for the transformation parameter. Then, this custom dataset

for training is passed into a data loader, the batch size is declared and the shuffle parameter is set to true to ensure that the images in the dataset are in a different order every time the model is trained.

If there is an already saved version of the trained model, it is loaded. If there is no previous save, this step is skipped.

After this the training module is called. The training module is implemented in a function called "train_model". The training module uses the "Adam" optimizer to optimize the training and the "BCEWithLogitsLoss" loss function from the torch.nn import. The training module loops over the (image, label) tuples in the data loader in batches and trains the model using the model and the optimizer and loss function previously mentioned. The training module loops over the complete training dataset a number of times as specified by the number of epochs. After training is complete, the model is saved in a pytorch file.

To start validation/testing, an instance of the custom dataset class is created for the validation/testing dataset in the same way it was done for the training dataset. The custom dataset is also passed into the data loader in the way as done for training.

Next, the saved version of the trained model is loaded to be validated/tested.

After this the testing function is called. The testing module is implemented in a function called "analyze_model". The module loops over the images and labels in the data loader and passes these images into the loaded model and stores the outputs. Next, the outputs from the model are reformatted and then compared to the true values stored in the data loader and the ROC, AUC, F1 score and thresholds are calculated and outputted.

This training and validation process can be started again as in when required to further train the model.

5.5.4 Potential Undesired Behaviors

One potential undesired behavior of the training/validation modules is having the training/validation fail if there is incorrect correspondence between the images in the CSV file and the dataset. Another potential undesired behavior is that the accuracy results produced by the model training are not up to expectation so the model has to be modified, retrained, and retested which is very time consuming. Lastly, another potential issue of the training and analyze module is that it can be overfitted to the training set leading to poor performance on new inputs.

6.0 User Interface

Our frontend allows the user to upload an x-ray image from their device, submit it through a click of a button, see the model results displayed on a table chart, and record notes for patients. There will be a login page and once logged in, the doctor will be able to see their list of patients and select one. Figures 1 to 3 in the Appendix showcases the current implementation. Also, the Databases section in the Appendix details our schema design for the patient, scan, and doctor tables.

Figma: Link

7.0 Appendix

7.1 Databases

The application uses MySQL DBMS for storing information related to patients, patient scans, and doctors. We make use of three tables, below is the DDL of each:

7.1.1 Patient Table

Column Name	Date Type	Description	Constraint
p_id	VARCHAR(50)	A unique patient identifier	PRIMARY KEY
p_name	VARCHAR(50)	The name of the patient	NOT NULL
d_id	VARCHAR(50)	The ID of the doctor in charge of the patient	FOREIGN KEY
p_sex	VARCHAR(50)	The sex of the patient	NOT NULL
p_birthdate	DATE	The date of birth of the patient	NOT NULL

7.1.2 Scan Table

Column Name	Data Type	Description	Constraint
s_id	VARCHAR(20)	A unique scan identifier	PRIMARY KEY
p_id	VARCHAR(20)	The scan associated with the patient	FOREIGN KEY
s_acqdate	DATE	The date of acquisition, ie, when the scan is taken	NOT NULL
s_pos	VARCHAR(20)	Position of the scan	NOT NULL
s_dicom	BLOB	The original scan of the patient	NOT NULL
s_orientation	VARCHAR(30)	The patient's orientation when the scan is taken	NOT NULL
s_comment	VARCHAR(300)	The comments made by the assisting doctor on the scan	

7.1.3 Doctor Table

Column Name	Data Type	Description	Constraint

d_id	VARCHAR(50)	A unique doctor identifier	PRIMARY KEY
d_username	VARCHAR(50)	Username of the doctor for login	UNIQUE
d_password	VARCHAR(50)	Password of the doctor for login	NOT NULL

7.2 UI/UX Design

Figure 1

Figure 2

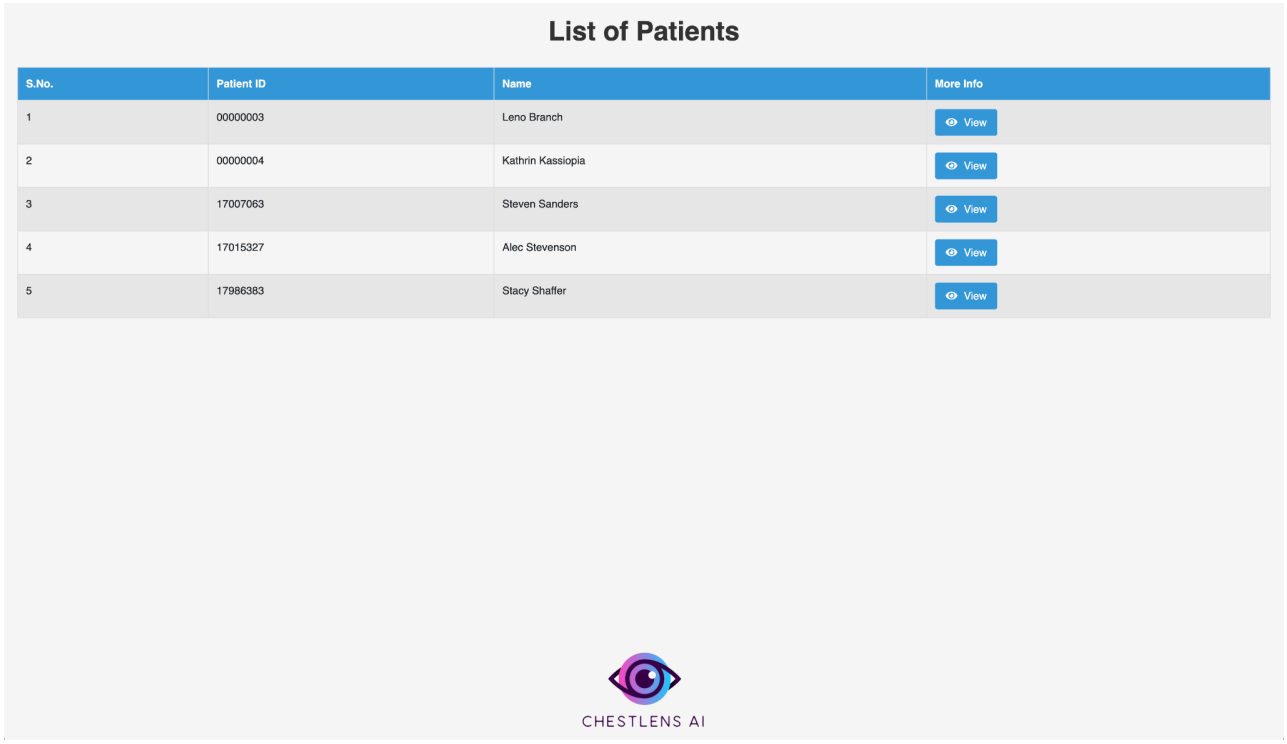
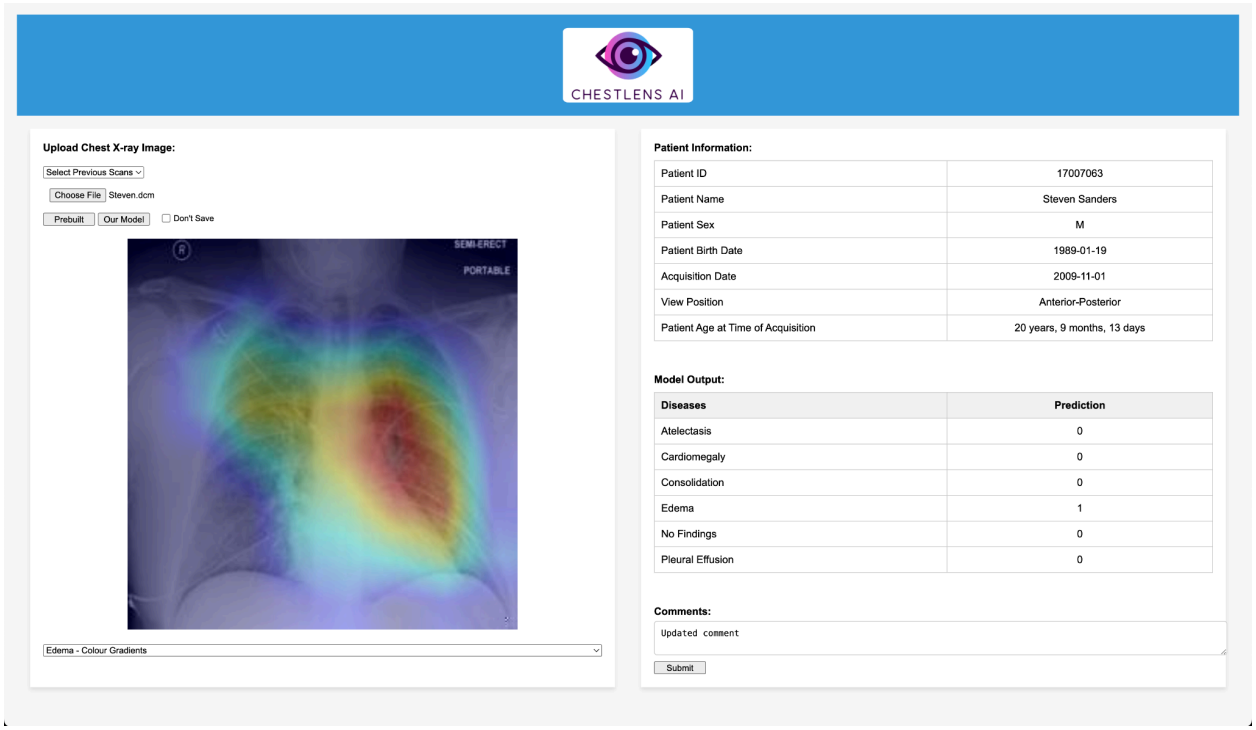


Figure 3


CHESTLENS AI
Upload Chest X-ray Image:
Select Previous Scans
Choose File
Steven.dcm
Prebuilt
Our Model
Don't Save
SEMI-ERECT
PORTABLE
Edema - Colour Gradients
Patient Information:
Patient ID 17007063
Patient Name Steven Sanders
Patient Sex M
Patient Birth Date 1989-01-19
Acquisition Date 2009-11-01
View Position Anterior-Posterior
Patient Age at Time of Acquisition 20 years, 9 months, 13 days
Model Output:
Diseases Prediction
Atelectasis 0
Cardiomegaly 0
Consolidation 0
Edema 1
No Findings 0
Pleural Effusion 0
Comments:
Updated comment
Submit