Verification and Validation Report: Chest Scan

Team 16, Ace
Harrison Chiu
Hamza Issa
Ahmad Hamadi
Jared Paul
Gurnoor Bal

March 10, 2025

1 Revision History

Date	**Version**	**Notes**
10 March	1.0	Gurnoor and Harrison add section 3

2 Symbols, Abbreviations and Acronyms

Table 1, includes the definitions and descriptions of all relevant symbols, abbreviations and acronyms used in this VnV Plan document.

Symbol, Abbreviation or Acronym	**Definiton or Description**
ML	Machine Learning: A branch of artificial intelligence that involves the use of algorithms to allow computers to learn from and make predictions based on data. This is a core technology used in the project for analyzing chest X-rays.
DL	Deep Learning: A subset of machine learning involving neural networks with many layers, used to analyze various types of data, including images.
DICOM	Digital Imaging and Communications in Medicine: A standard for transmitting, storing, and sharing medical imaging information. It is used to manage medical images in the proposed solution.
CNN	Convolutional Neural Network: A type of deep learning model specifically designed for processing structured grid data like images, used in the project for chest X-ray analysis.
EHR	Electronic Health Record: A digital version of a patient's paper chart, used for storing patient information and history that may be integrated with the proposed solution.
API	Application Programming Interface: A set of rules and protocols for building and interacting with software applications, enabling the integration of the proposed solution with other systems.
MC	Mandated Constraints: Various constraints placed on the project's proposed solution that must be adhered to throughout the development process.
FR	Functional Requirement: A requirement that specifies what functionality the project's proposed solution must provide to meet user needs.

NFR	Nonfunctional Requirement: A requirement that specifies criteria that can be used to judge the operation of a system, rather than specific behaviors (e.g., performance, usability).
BUC	Business Use Case: A scenario that describes how the proposed solution can be used within a business context to achieve specific goals.
PUC	Product Use Case: A scenario that details how an individual user will interact with the proposed solution to achieve specific tasks.
MVP	Minimum Viable Product: A version of the proposed solution that includes only the essential features required to meet the core needs of the users and stakeholders.
MG	Module Guide
MIS	Module Interface Specification
PoC	Proof of Concept
SRS	Software Requirements Specification
FRTC	Functional Requirements Test Case
NFRTC	Nonfunctional Requirements Test Case
VnV	Verification and Validation

Contents

List of Tables

List of Figures

This document ...

3 Functional Requirements Evaluation

3.1 User Authentication (FR.8)

3.1.1 User Authentication (FRTC11, 12, 15, 16, 17 - Updated)

Initial State: The system is operational with user accounts created.
Input: Valid login credentials for an authorized user.
Expected Output: The system authenticates the user and grants access according to the user's role.
Actual Output: The system successfully authenticates the user and redirects them to the dashboard, allowing appropriate access based on their role.
Expected and Actual Output Match: True
Relevant Functional Requirement(s): FR.8 (Authentication and Authorization Mechanisms)

3.2 Image Upload (FR.1)

3.2.1 Chest X-ray Image Input Acceptance (FRTC1)

Initial State: The system is in a stable state with all components initialized and ready to receive input.
Input: A sample chest X-ray image in a valid format (JPG, PNG, DICOM).
Expected Output: The system accepts and reads the chest X-ray image successfully. No error messages or system anomalies occur.
Actual Output: The system successfully accepts, reads, and stores the uploaded image in the backend.
Expected and Actual Output Match: True
Relevant Functional Requirement(s): FR.1 (Image Upload)

Invalid Chest X-ray Image Format Rejection (FRTC2) **Initial State:** The system is ready to receive an image file.
Input: A sample image in an invalid format (e.g., .TXT, .DOCX).
Expected Output: The system rejects the invalid image input.

An appropriate error message is displayed, informing the user about supported formats. **Actual Output:** The system correctly rejects invalid file formats and provides an error message.
Expected and Actual Output Match: True

3.3 Image Preprocessing (FR.2 - Updated)

3.3.1 Image Preprocessing and Standardization (FRTC3 - Completely Revised)

Initial State: A chest X-ray image has been uploaded and is ready for preprocessing.
Input: A valid chest X-ray image in JPG, PNG, or DICOM format.
Expected Output: The system resizes the image to match the CNN model's required dimensions.
Pixel intensity values are normalized to enhance consistency. The preprocessed image is saved and ready for inference. **Actual Output:** The system successfully resizes and normalizes images, preparing them for model analysis.
Expected and Actual Output Match: True
Relevant Functional Requirement(s): FR.2 (Image Preprocessing)

3.3.2 Handling Invalid Image Formats (FRTC4 - Updated)

Initial State: The system is awaiting an image upload.
Input: An invalid image file (e.g., TXT, DOCX, unsupported formats, or corrupted image files).
Expected Output:

- The system detects that the image is in an unsupported format or corrupted.
- The system displays an error message to the user.
- The system prevents further processing and does not send the image to the model.

Actual Output: The system successfully identifies invalid formats and

blocks the upload, displaying an appropriate error message.
Expected and Actual Output Match: True
Relevant Functional Requirement(s): FR.2 (Image Preprocessing)

3.4 CNN Model Accurate Analysis (FR.3 - Updated)

3.4.1 Disease Classification with Confidence Scores (FRTC5 - Updated)

Initial State: The system has preprocessed an uploaded image and is ready for analysis.
Input: A preprocessed chest X-ray image with known disease patterns (e.g., Pneumonia, Cardiomegaly, Atelectasis).
Expected Output:

- The CNN model classifies the image and assigns a disease label.
- The model outputs confidence scores (e.g., Pneumonia: 85%, No Finding: 10%).
- The prediction results are displayed to the user.

Actual Output: The CNN model successfully classifies diseases and assigns confidence scores.
Expected and Actual Output Match: True
Relevant Functional Requirement(s): FR.3 (CNN Model Accurate Analysis)

3.4.2 Model Prediction for No Disease Cases (FRTC6 - Updated)

Initial State: The CNN model is ready for analysis, and a healthy chest X-ray is uploaded.
Input: A preprocessed chest X-ray image of a patient with no known disease.
Expected Output:

- The CNN model classifies the image as No Finding or returns very low probabilities for disease labels.
- The confidence scores reflect minimal probability of disease.

- The prediction results are displayed to the user.

Actual Output: The system correctly identifies images without disease and returns a No Finding classification.
Expected and Actual Output Match: True
Relevant Functional Requirement(s): FR.3 (CNN Model Accurate Analysis)

3.5 Display Results (FR.6 - Updated)

3.5.1 Diagnostic Report and Heatmap Access via Web Interface (FRTC10 - Updated)

Input: The user navigates to the results page after an image has been analyzed.
Expected Output: The system displays the predicted disease classification(s). The confidence scores are presented next to each disease label. (Optional) A heatmap overlay is shown, visually indicating affected areas in the X-ray. The interface is formatted clearly for easy interpretation.
Actual Output: The system correctly displays predicted disease labels, corresponding confidence scores, and a heatmap visualization (if enabled).
Expected and Actual Output Match: True
Relevant Functional Requirement(s): FR.6 (Display Results)

3.5.2 Display Prediction Results in Readable Format (FRTC18 - New)

Initial State: The system has completed model inference, and the user is accessing the results page.
Input: The user opens the result page after an image has been analyzed.
Expected Output: The predicted disease label(s) are prominently displayed. The corresponding confidence scores are formatted for clarity. No extraneous or misleading information is presented. Results are accessible within a reasonable timeframe (<5 seconds).
Actual Output: The system successfully displays predictions in a structured and readable format.
Expected and Actual Output Match: True

Relevant Functional Requirement(s): FR.6 (Display Results)

3.6 Heatmap Report (FR.7)

3.6.1 Heatmap Display on Chest X-ray Images (FRTC9 - Updated)

Initial State: The CNN model has completed image analysis, and the system has generated a heatmap.
Input: The user navigates to the results page and requests a heatmap visualization.
Expected Output: The system overlays a heatmap on the chest X-ray image. The heatmap highlights areas where the model detected abnormalities. The overlay does not obscure critical image details.
Actual Output: The system successfully generates and overlays the heatmap on the X-ray image, maintaining image clarity.
Expected and Actual Output Match: True
Relevant Functional Requirement(s): FR.7 (Heatmap Report)

3.6.2 Heatmap Generation Error Handling (FRTC19 - New)

Initial State: The CNN model has completed analysis, but an error occurs in heatmap generation.
Input: The system attempts to generate a heatmap but encounters an issue (e.g., missing heatmap data, computation error).
Expected Output: The system displays a message indicating that the heatmap could not be generated. The rest of the diagnostic results (predictions and confidence scores) remain accessible.
Actual Output: The system successfully identifies heatmap generation errors and informs the user without affecting other results.
Expected and Actual Output Match: True
Relevant Functional Requirement(s): FR.7 (Heatmap Report)

3.7 User Dashboard (FR.9)

3.7.1 Display User Dashboard with Past Uploads (FRTC20 - New)

Initial State: The user is logged in and has previously uploaded X-ray images.
Input: The user navigates to the dashboard page.
Expected Output: The system retrieves and displays a list of past uploaded X-ray images. Each entry includes the upload timestamp and corresponding diagnosis results.
Actual Output: The system successfully loads and displays past uploads with timestamps and results.
Expected and Actual Output Match: True
Relevant Functional Requirement(s): FR.9 (User Dashboard)

3.8 Secure API for Model Inference (FR.9)

3.8.1 API Endpoint for Image Inference (FRTC21 - New)

Initial State: No request is made, and the API is available.
Input: A POST request containing a valid chest X-ray image.
Expected Output:

- The API processes the image and forwards it to the CNN model.
- The API returns a JSON response with the predicted disease label and confidence scores.
- The response follows the correct structure (e.g., ”disease”: ”Pneumonia”, ”confidence”: 85%).

Actual Output: The API successfully processes the request and returns the correct JSON response format.
Expected and Actual Output Match: True
Relevant Functional Requirement(s): FR.9 (Secure API for Model Inference)

3.8.2 API Handling of Invalid Image Format (FRTC22 - New)

Initial State: No request is made, and the API is available.
Input: A POST request containing an invalid file format (e.g., .TXT, .DOCX).
Expected Output:

- The API rejects the request.
- The API returns a structured error message (e.g., ”error”: ”Invalid file format. Please upload
- JPG, PNG, or DICOM.”).

Actual Output: The API correctly rejects invalid file formats and returns an appropriate error message.
Expected and Actual Output Match: True
Relevant Functional Requirement(s): FR.9 (Secure API for Model Inference)

3.9 Data Storage & Management (FR.7)

3.9.1 Secure Image and Result Storage (FRTC13 - Updated)

Initial State: A user has uploaded an X-ray image, and the system is ready to store the data.
Input: A valid chest X-ray image is uploaded, and the CNN model returns predictions.
Expected Output:

- The system stores the uploaded image in a secure storage location.
- The system stores prediction results in the database, linking them to the respective image.
- No unauthorized access to stored data occurs.

Actual Output: The system successfully stores images and results securely, ensuring accessibility for future retrieval.
Expected and Actual Output Match: True
Relevant Functional Requirement(s): FR.7 (Data Storage & Management)

3.9.2 Retrieval of Stored Results (FRTC14 - Updated)

Initial State: The database contains stored images and results for a logged-in user.
Input: A user navigates to the dashboard and requests to view past results.
Expected Output:

- The system retrieves and displays stored images with associated predictions.
- Each entry includes timestamps and confidence scores for past analyses.
- The data retrieval process is efficient, ensuring a smooth user experience.

Actual Output: The system successfully retrieves and displays stored images and results on the dashboard.
Expected and Actual Output Match: True
Relevant Functional Requirement(s): FR.7 (Data Storage & Management)

3.10 Error Handling & Notifications (FR.9)

3.10.1 Invalid Image Format Handling (FRTC2 - Updated)

Initial State: The system is awaiting an image upload.
Input: A user attempts to upload an unsupported file format (e.g., TXT, DOCX, MP4).
Expected Output:

- The system rejects the upload.
- An error message appears stating, "Invalid file format. Please upload a JPG, PNG, or DICOM file."
- The user is prompted to try again with a valid file format.

Actual Output: The system correctly rejects invalid file formats and displays an appropriate error message.
Expected and Actual Output Match: True
Relevant Functional Requirement(s): FR.9 (Error Handling & Notifications)

3.10.2 Corrupted Image Upload Handling (FRTC4 - Updated)

Initial State: The system is awaiting an image upload.
Input: A user attempts to upload a corrupted chest X-ray image.
Expected Output:

- The system detects that the file is corrupted and cannot be processed.
- An error message appears stating, "File appears to be corrupted. Please upload a valid image."
- The system does not store or process the corrupted file.

Actual Output: The system successfully detects and rejects corrupted image files, displaying an appropriate error message.
Expected and Actual Output Match: True
Relevant Functional Requirement(s): FR.9 (Error Handling & Notifications)

3.11 Multi-Disease Classification (FR.11)

3.11.1 Multi-Disease Detection with Confidence Scores (FRTC5 and 6 - Updated)

Initial State: The system has preprocessed an uploaded image and is ready for analysis.
Input: A preprocessed chest X-ray image containing multiple known diseases (e.g., Pneumonia and Tuberculosis).
Expected Output:

- The CNN model classifies the image and returns a list of detected diseases.
- The confidence scores for each predicted disease are displayed. textbfitem Example output: Pneumonia: 85%, Tuberculosis: 70%.

Actual Output: The CNN model successfully detects multiple diseases in a single image and provides confidence scores.
Expected and Actual Output Match: True
Relevant Functional Requirement(s): FR.11 (Multi-Disease Classification)

3.11.2 No Multi-Disease Detection for Single Condition (FRTC5 - Updated)

Initial State: The system has preprocessed an uploaded image and is ready for analysis.
Input: A preprocessed chest X-ray image with only one known disease (e.g., Cardiomegaly).
Expected Output:

- The CNN model classifies the image and returns a single disease label.
- Confidence scores for unrelated diseases remain low. **Example output:** Cardiomegaly: 92%, No Finding: 8%.

Actual Output: The system correctly identifies only one disease when multiple diseases are not present.
Expected and Actual Output Match: True
Relevant Functional Requirement(s): FR.11 (Multi-Disease Classification)

3.12 Model Confidence (FR.12)

3.12.1 Low Confidence Warning for Uncertain Predictions (FRTC8 - Updated)

Initial State: The system has preprocessed an uploaded image and is ready for analysis.
Input: A preprocessed chest X-ray image that leads to low confidence predictions (e.g., all detected diseases have confidence scores below 50%). **Expected Output:**

- The system returns disease classifications with confidence scores.
- If the highest confidence score is below 50%, the system displays a warning: "Low confidence in prediction – consider consulting a radiologist."

Actual Output: The system correctly identifies low-confidence cases and displays the warning message.
Expected and Actual Output Match: True

Projects	**Links**
This project	https://github.com/harrisonchiu/xray
CXR-Capstone	https://github.com/RezaJodeiri/CXR-Capstone
CXR-LLaVA	https://github.com/ECOFRI/CXR_LLaVA https://arxiv.org/abs/2310.18341

Relevant Functional Requirement(s): FR.12 (Model Confidence)

3.12.2 High Confidence Predictions Displayed Normally (FRTC7 - Updated)

Initial State: The system has preprocessed an uploaded image and is ready for analysis.
Input: A preprocessed chest X-ray image that leads to high-confidence predictions (e.g., Pneumonia: 85%). **Expected Output:**

- The system returns disease classifications with confidence scores.
- If confidence scores are above 50%, no warning is displayed.

Actual Output: The system correctly displays high-confidence predictions without unnecessary warnings.
Expected and Actual Output Match: True
Relevant Functional Requirement(s): FR.12 (Model Confidence)

4 Nonfunctional Requirements Evaluation

4.1 Usability

4.2 Performance

4.3 etc.

5 Comparison to Existing Implementation

5.1 CXR-LLaVA Comparison

CXR-LLaVA is an open-source multimodal large language model designed to interpret chest X-ray images and generate radiologic reports for those images. By integrating advanced vision transformers with large language models, it aims to emulate the diagnostic capabilities of human radiologists.

Their model was trained on many open source CXR datasets. These include:

- BrixIA: A dataset focusing on COVID-19 pneumonia severity scoring.
- CheXpert: A large dataset with labeled CXR images covering multiple pathologies.
- MIMIC-CXR: Comprises de-identified CXR images and corresponding radiology reports.
- NIH Chest X-ray Dataset: Contains over 100,000 CXR images with annotations for 14 disease labels.
- PadChest: Offers a wide variety of thoracic disease labels and radiographic findings.
- RSNA COVID-19 AI Detection Challenge: Provides CXR images related to COVID-19 cases.
- VinDR-CXR: A dataset with annotations for various lung diseases.

Collectively, there are more than 500 000 different CXR images which is a lot of training data.

CXR-LLaVA's architecture is made of:

- Image Encoder: A Vision Transformer (ViT-L/16) processes CXR images at a resolution of 512x512 pixels in grayscale.
- Language Model: LLAMA2-7B-CHAT serves as the foundational language model, facilitating the generation of coherent and contextually relevant radiologic reports.

Generally, CXR-LLaVA and our project share the common goal of analysing chest x-ray images to identify diseases. However, they still differ in several aspects.

Our project focuses on detecting diseases in CXR images with a CNN in the backend. CXR-LLaVA on top of that, generates a comprehensive radiologic report from the CXR image, summarizing its findings in order to emulate a proper radiologist diagnosis. As a result, their model architecture is more complex, combining a Vision Transformer for the image encoder and a Llama language model to output a textual report based on the visual input from the vision transformer. Vision transformers and language models are very data hungry. So, they also use a lot more datasets, composed of over 500 000 different images.

However, there is no front end. In order to use it, you must follow their instructions to load their trained model and then run it in the terminal.

It is an advanced research project with its own research paper referencing it.

5.2 CXR-Capstone

The CXR-Capstone project is an open-source project that analyzes chest X-ray (CXR) images to predict potential diseases using a CNN model with multi-label prediction. It also uses demographic information such as race, age, and gender. In this way, it classifies images based on demographics as well for the intent of mitigating biases.

The project is very similar to ours as it has a frontend running a backend model that takes in the user's uploaded image and analyses if there are any diseases. On top of that, it also tracks the progression of conditions over time and generates a simple diagnostic report based on the image.

Their project uses a newer model architecture of DETR (detection transformer). It is said to have higher accuracy and run time performance than traditional CNNs. Its architecture is based on a transformer which is typically more data hungry, so it also uses a much larger dataset, MIMIC-CXR, which has over 300 000 images.

The end product is still similar to our project but has slightly more features of tracking progression (assuming the users continuously upload new images of themselves) and report generation.

6 Unit Testing

7 Changes Due to Testing

[This section should highlight how feedback from the users and from the supervisor (when one exists) shaped the final product. In particular the feedback from the Rev 0 demo to the supervisor (or to potential users) should be highlighted. —SS]

8 Automated Testing

Automated testing for our project currently emphasizes maintaining high code quality through consistent linting and formatting checks. The existing automated tests ensure code quality, readability, and consistency across the entire codebase. Our project does not require extensive CI/CD automated testing because the team is small and focused. The codebase is also manageable and easily divided up in parts where each member could work on it separately.

8.1 Linting and Formatting

We utilize GitHub Actions workflows to automate linting and formatting checks for both frontend and backend components. These automated checks help maintain code consistency, improve readability, and minimize potential bugs.

8.2 Frontend (React.js / JavaScript)

We use two primary tools for frontend code quality:

- **ESLint:** ESLint analyzes JavaScript and React code, enforcing best practices and identifying errors or potential issues such as unused variables, improper imports, syntax errors, and inconsistent code patterns. Developers run ESLint locally before pushing code to identify and fix issues proactively. The ESLint configuration adheres to industry-standard rules optimized for React development, providing detailed error and warning messages to facilitate rapid fixes.
- **Prettier:** Prettier automatically formats JavaScript and React code, enforcing consistent indentation, spacing, bracket placement, and line lengths. This ensures that the frontend codebase remains uniform regardless of individual developer preferences. Automated checks verify adherence to the Prettier rules, failing commits or pull requests that do not comply.

8.3 Backend (Python)

The backend uses robust automated checks using the following tools:

- **Flake8:** Flake8 enforces adherence to Python's PEP 8 standards. It identifies potential syntax errors, unused imports or variables, and common coding pitfalls. By automating Flake8 checks, we ensure backend code quality remains consistently high, reducing manual code reviews and catching errors early in the development process.
- **Black:** Black automatically formats Python code to a unified and clear style, eliminating debates about formatting preferences. It handles line length, spacing, and structural formatting. Automated Black checks verify code conformity and clearly indicate any violations, ensuring readability and maintainability.

8.4 Running Automated Tests Locally

Developers can perform these tests locally before committing changes, ensuring issues are addressed early:

8.4.1 JavaScript Checks:

`npm run lint` and `npm run format:check`

- This command runs ESLint and Prettier, respectively, highlighting or automatically fixing formatting and linting issues.

`flake8 .` and `black --check .`

- Running these commands validates backend Python code adherence to style and linting standards.

8.5 GitHub Actions Integration

GitHub Actions workflows trigger these automated checks each time a developer makes a pull request or pushes code to the main branch. These workflows include:

- Linting Workflow: Runs ESLint (frontend) and Flake8 (backend). If linting errors are found, the workflow fails, clearly identifying issues that must be resolved before merging.
- Formatting Workflow: Checks formatting with Prettier (frontend) and Black (backend). Any deviations from the established formatting rules trigger a failed workflow, requiring immediate correction by developers.

These checks ensure only clean and consistently formatted code enters the main codebase.

The GitHub Actions workflows are configured to trigger automatically whenever:

- A pull request is opened or updated.
- Code is pushed to critical branches (e.g., main or development).

If a workflow fails, GitHub clearly indicates the exact nature and location of errors, allowing developers to quickly identify and fix issues. This rapid feedback cycle helps team productivity and helps maintain a robust and error-free codebase

9 Trace to Requirements

10 Trace to Modules

11 Code Coverage Metrics

Appendix — Reflection

The information in this section will be used to evaluate the team members on the graduate attribute of Reflection.

The purpose of reflection questions is to give you a chance to assess your own learning and that of your group as a whole, and to find ways to improve in the future. Reflection is an important part of the learning process. Reflection is also an essential component of a successful software development process.

Reflections are most interesting and useful when they're honest, even if the stories they tell are imperfect. You will be marked based on your depth of thought and analysis, and not based on the content of the reflections themselves. Thus, for full marks we encourage you to answer openly and honestly and to avoid simply writing "what you think the evaluator wants to hear."

Please answer the following questions. Some questions can be answered on the team level, but where appropriate, each team member should write their own response:

1. What went well while writing this deliverable?
2. What pain points did you experience during this deliverable, and how did you resolve them?
3. Which parts of this document stemmed from speaking to your client(s) or a proxy (e.g. your peers)? Which ones were not, and why?
4. In what ways was the Verification and Validation (VnV) Plan different from the activities that were actually conducted for VnV? If there were differences, what changes required the modification in the plan? Why did these changes occur? Would you be able to anticipate these changes in future projects? If there weren't any differences, how was your team able to clearly predict a feasible amount of effort and the right tasks needed to build the evidence that demonstrates the required quality? (It is expected that most teams will have had to deviate from their original VnV Plan.)