Verification and Validation Report: CXR

Team 27, Neuralyzers
Ayman Akhras
Nathan Luong
Patrick Zhou
Kelly Deng
Reza Jodeiri

March 7, 2025

1 Revision History

Date	Version	Notes
March 01	1.0	Nanthan completed section 3
March 02	1.1	Kelly completed section 4
March 05	1.2	Patrick completed section 2, 5
March 05	1.3	Aymen completed section 7

2 Symbols, Abbreviations, and Acronyms

This section records information for easy reference and aims to reduce ambiguity in understanding key concepts used in the project.

2.1 Table of Units

Throughout this document, SI (Système International d'Unités) is employed as the unit system. In addition to basic units, several derived units are used as described below. For each unit, the symbol is given, followed by a description of the unit and the SI name.

Symbol	**Unit**	**SI**
s	Time	Second
GB	Data	Gigabyte
MB	Data	Megabyte
LOC	Quantity	Lines of Code

2.2 Definitions

This subsection provides a list of terms that are used in the subsequent sections and their meanings, with the purpose of reducing ambiguity and making it easier to correctly understand the requirements:

- **Artificial Intelligence (AI) Model**: A program that analyzes datasets to identify patterns and make predictions. Used extensively in medical image analysis for automating diagnostics.
- **Convolutional Neural Network (CNN)**: A deep learning algorithm that processes images by assigning weights and biases, allowing it to identify patterns and features in medical images such as chest X-rays.
- **Detection Transformer (DETR)**: A transformer-based neural network model for object detection. It uses an encoder-decoder transformer architecture to directly predict bounding boxes and class labels from an image, simplifying the detection process.

- **DICOM (Digital Imaging and Communications in Medicine)**: The international standard for medical images, defining formats for image exchange that ensure clinical quality.

- **Containerized Application**: A portable version of an application that can be run on a container run-time, such as Docker.

- **Machine Learning (ML)**: A subset of AI focusing on using data and algorithms to mimic human learning, improving accuracy over time.

- **Picture Archiving and Communication System (PACS)**: A system for acquiring, storing, transmitting, and displaying medical images digitally, providing a filmless clinical environment.

- **PHI**: Personal Health Information - Private and confidential data that must be protected under the HIPPA act.

- **HIPPA**: Health Insurance Portability and Accountability Act, a set of standards protecting sensitive health information from disclosure without patient's consent.

- **AWS - Amazon Web Services**: A public cloud provider, offering all HIPAA-compliance cloud services that helps Neuralanalyzer host, manage, and scale our application.

- **AWS ECS**: AWS Elastic Cloud Service: - An AWS managed service for managing and maintaining application containers at run-time.

- **AWS ECR**: AWS Elastic Container Registry - An AWS managed service for storing and managing container images.

- **AWS Fargate**: An AWS managed service for running containerized applications.

- **AWS Cognito**: An AWS managed service for authentication logic, handling user and password management.

- **React**: A web front-end framework, written in Javascript.

- **Flask**: An HTTP-based server framework, written in Python.

- **Finite State Machine (FSM)**: A computation model that simulates sequential logic using state transitions, applied in processes like user authentication and backend workflows.

- **ROC Curve (Receiver Operating Characteristic Curve)**: A graph that shows the performance of a classification model by plotting the true positive rate against the false positive rate at various threshold levels.

- **Service-Level Agreement (SLA)**: Defines the guaranteed uptime of the system, such as ensuring the availability of the AI service for 99.99% of operational hours.

- **Software as a Medical Device (SaMD)**: Software classified as a medical device under regulatory frameworks, such as those defined by the Food and Drugs Act.

- **TorchXRAYVision**: An open-source library for classifying diseases based on chest X-ray images, offering pre-trained models to accelerate the development process.

- **X-ray**: A form of high-energy electromagnetic radiation used in medical imaging to produce images of the inside of the body, enabling the diagnosis of conditions through radiographic film or digital detectors.

- **MIT License**: An open-source software license that allows for the free use, modification, and distribution of software.

- **Training Data**: Refers to the dataset of labeled chest X-ray images used to train the AI model. In this project, the dataset size is approximately 471.12 GB.

2.3 Abbreviations and Acronyms

Symbol	Description
SRS	Software Requirements Specification
AI	Artificial Intelligence
CNN	Convolutional Neural Network
DICOM	Digital Imaging and Communications in Medicine
DETR	Detection Transformer
ViT	Vision Transformer
VnV	Verification and Validation
ML	Machine Learning
PACS	Picture Archiving and Communication System
SaMD	Software as a Medical Device
ROC	Receiver Operating Characteristic Curve
SLA	Service-Level Agreement
FR	Functional Requirement
NFR	Non-Functional Requirement
FSM	Finite State Machine
CXR	Chest X-Ray Project
POC	Proof of Concept
TM	Theoretical Model
AWS	Amazon Web Services
ECS	Elastic Container Service
ECR	Elastic Container Registry
CI/CD	Continuous integration and Continuous deployment
HTTP	Hypertext Transfer Protocol

Contents

List of Tables

List of Figures

3 Functional Requirements Evaluation

1. FR1

   Initial State: The system is set up to accept chest X-ray images as input from authorized users.

   Input/Condition: Calling HTTP Server Module API to get a presigned URL for uploading the image with and without an authorization token.

   Expected Output/Result: Presigned URL does not return if the user is not authorized.

   Actual Output: Presigned URL does not return if the user is not authorized.

   Result: Pass

2. FR2

   Initial State: The system allows users to input additional patient symptoms.

   Input/Condition: Execute integration tests on the Medical Record Management Module to include additional patient symptoms.

   Expected Output/Result: Additional patient symptoms are included in the medical record from the Data Persistent Module.

   Actual Output: Additional patient symptoms are included in the medical record from the Data Persistent Module.

   Result: Pass

3. FR3

   Initial State: The system analyzes chest X-ray images to detect the presence or absence of specific diseases.

   Input/Condition: Execute Validation Script on the Disease Detection Module to check the accuracy of the AI model.

   Expected Output/Result: The accuracy of the AI model is 85

   Actual Output: The accuracy of the AI model is 85

   Result: Pass

4. FR4

   Initial State: The system determines whether a patient's condition has improved, worsened, or remained stable.

   Input/Condition: Execute integration tests on the Disease Progression Module with pre-defined patient conditions.

   Expected Output/Result: The system outputs "improved," "worsened," or "remained stable."

   Actual Output: The system outputs "improved," "worsened," or "remained stable."

   Result: Pass

5. FR5

   Canceled

6. FR6

   Initial State: The system generates structured, human-readable reports.

   Input/Condition: Execute integration tests on the Internal Report Generation Service with pre-defined patient conditions.

   Expected Output/Result: The output includes key findings, disease detection results, and progression status.

   Actual Output: The output includes key findings, disease detection results, and progression status.

   Result: Pass

7. FR7

   Initial State: The system stores patient data securely.

   Input/Condition: Execute integration tests on the Medical Record Management Module to create a medical record with images and reports.

   Expected Output/Result: Medical records, findings, and X-ray images are stored in the Data Persistent Module.

   Actual Output: Medical records, findings, and X-ray images are stored in the Data Persistent Module.

Result: Pass

8. FR8

   Initial State: The system provides alerts when significant changes in a patient's condition are detected.

   Input/Condition: Manually create a medical record on the Disease Progression Module with significant changes in a patient's condition.

   Expected Output/Result: Significant changes appear in the UI of the Patient List View Module and Patient Overview Module.

   Actual Output: Significant changes appear in the UI of the Patient List View Module and Patient Overview Module.

   Result: Pass

9. FR9

   Initial State: The system allows healthcare professionals to adjust treatment plans based on X-ray analysis.

   Input/Condition: Manually edit a medical record on the X-Ray Report View Module with adjusted treatment plans.

   Expected Output/Result: Adjusted treatment plans appear in the UI of the Patient List View Module and Data Persistent Module.

   Actual Output: Adjusted treatment plans appear in the UI of the Patient List View Module and Data Persistent Module.

   Result: Pass

10. FR10

    Canceled

11. FR11

    Initial State: The system displays confidence levels for disease detection and progression analysis.

    Input/Condition: Manually create a medical record on the X-Ray Report View Module with known disease confidence levels.

    Expected Output/Result: Confidence levels are accurately shown on the UI of the X-Ray Report View Module.

Actual Output: Confidence levels are accurately shown on the UI of the X-Ray Report View Module.

Result: Pass

12. FR12

    Canceled

13. FR13

    Canceled

14. FR14

    Canceled

15. FR15

    Canceled

16. FR16

    Initial State: The system supports regular updates to the AI model to improve accuracy.

    Input/Condition: Execute integration tests on the AI Model Update Module to ensure updates are applied correctly.

    Expected Output/Result: The AI model's accuracy improves over time with new data.

    Actual Output: The AI model's accuracy improves over time with new data.

    Result: Pass

4 Nonfunctional Requirements Evaluation

4.1 Look and Feel

1. NFR-LF1

   Initial State: The system must be installed and accessible.

Input/Condition: Adjust window/level settings. Navigate between other tabs. Apply basic tools (zoom, pan, measure). Export or save images from the viewer.

Expected Output/Result: The system responds to user interactions.

Actual Output: The system responds to user interactions promptly and correctly.

Result: Pass

2. NFR-LF2

Initial State: The system under test is installed and accessible on the target device. The monitor brightness is set to a comfortable level to ensure consistency during the test.

Input/Condition: User interaction with the interface for typical workflows.

Expected Output/Result: Color contrast ratio is 4.5:1 or higher for text. Font sizes at least 12-14 points for normal text.

Actual Output: All systemm interface showed valid color contrast ratioand appropriate font size

Result: Pass

4.2 Usability and Humanity

1. NFR-UH1

Initial State: User accounts and access credentials are prepared for all healthcare professionals participating in the test. Any required files or images are pre-loaded and accessible for the tasks.

Input/Condition: Test user given a list of specified tasks and user accounts.

Expected Output/Result: User successfully login each user account and perform: upload a chest X-ray image to the system, view analysis results or processed reports, and export the analysis or report to a local folder.

Actual Output: User can login as doctor / patient and upload chest X-ray image to the system, view analysis results or processed reports.

Result: Pass

2. NFR-UH2
   Canceled

4.3 Performance

1. NFR-PR1

   Initial State: User is logged in. Standard chest X-ray images are available for testing in PNG/JPG format (depending on system requirements).

   Input/Condition: Test user interacts with the system to upload an X-Ray image.

   Expected Output/Result: AI-analyzed results of the X-Ray image is displayed in user interface within 1 minute of uploading the X-ray image.

   Actual Output: Disease probabilities was produced by AI model in user interface along with summaries.

   Result: Pass

2. NFR-PR2

   Initial State: A monitoring system is set up to log availability metrics, such as downtime events, mean time to recovery, and uptime percentage.

   Input/Condition: Server outages, network issues, high number of concurrent users.

   Expected Output/Result: Uptime percentage; downtime events (log the number, duration, and cause of any disruptions or downtime); mean time to recovery.

   Actual: Uptime percentage of the system is above 99

   Result: Pass

3. NFR-PR3

   Initial State: All dependencies (database, network, storage, and image processing engine) are configured and operational. Monitoring tools are available to track system performance, including CPU and memory usage, image processing times.

   Input/Condition: Collections of 20 identical or varied images, multiple user accounts. Baseline for each image processing is 20 seconds.

   Expecetd Output/Result: Processing time (float), system resource utilization (float), images uploading success/fail (boolean).

   Actual Output: Image uploading function is working. Processing time is within 20 seconds for all images. System resource utilization showed 15

   Result: Pass.

4.4 Operational and Environmental

1. NFR-OE1

   Initial State: At least 90common tasks (e.g., uploading images, viewing analysis results).

   Input/Condition:

   Expected Output/Result: Boolean indicating whether the PACS successfully stores the AI-processed results with correct format (annotated image or report). The stored result is associated with the correct patient ID and instance number in the PACS.

   Actual Outout: The AI system was not able to connect to a hospital network.

   Result: Fail

2. NFR-OE2

   Initial State: The network is configured normally with minimal latency and no packet loss initially. Necessary patient data and chest X-ray studies are available for retrieval.

Input/Condition: Retrieve and process a chest X-ray image, store the processed result, and send the data to external.

Expected Output/Result: Latency is within a reasonable time (190ms 220ms), logs include latency and packet loss statistics.

Actual Output: Logs showing that latency is around 190ms and no packet loss detected.

Result: Pass

4.5 Security and Privacy

1. NFR-SR1

   Initial State: AES-256 encryption libraries (such as OpenSSL or Cryptography in Python) are configured for the system. Patient data (including DICOM images and reports) is stored on the system or transmitted over the network.

   Input/Condition: X-ray image and diagnostic report with patient ID and instance number. Secret key and initialization vector (IV) for AES-256 encryption.

   Expected Output/Result: All patient data, including images and reports, is encrypted using AES-256 both during storage and transmission.

   Actual Output: HTTPS transmission from backend to frontend (vise versa) securely.

   Result: Pass

2. NFR-SR2
   Canceled

4.6 Maintainability and Support

1. NFR-MS1

   Initial State: The AI system code-base is deployed in a version-controlled environment. Documentation for each module (e.g., README files, API references, inline comments) is available in the repository.

Input/Condition: All modules within the code repository are accessible.

Expected Output/Result: All key functionalities (e.g., data ingestion, inference, reporting) are encapsulated in separate, well-defined modules. Modules can function independently with minimal coupling. Dependencies between modules are well-documented.

Actual Output: All modules are separated and follows SOLID principles. Modules are self-contained that can be modified without breaking other parts.

Result: Pass

2. NFR-MS2

Initial State: Code repository contains the entire AI system, automated testing framework (e.g., JUnit, Pytest) is installed and configured in the project. Tests include unit tests, integration tests, system tests, and end-to-end tests. Code coverage tool (e.g., coverage.py, JaCoCo) is integrated.

Input/Condition: Unit tests, integration tests, and end-to-end tests.

Expected Output/Result: Code coverage log.

Actual Output: Code coverage log included as part of CI/CD pipeline which will be triggered everytime there's a new commit.

Result: Pass

4.7 Cultural

1. NFR-CR1
   Initial State: The language options (English and French) are available in the settings menu.

   Input/Condition: Switch language between English and French. Generate report in English and French.

   Expected Output/Result: No untranslated or misaligned content should appear.

   Actual Output: System does not show French content.

   Result: Fail

4.8 Legal

1. NFR-LR1

   Initial State: HIPAA and PIPEDA documentation are in place, development artifacts and source code are available.

   Input/Condition: Development artifacts, system design documentation.

   Expected Output/Result: System design aligns with the requirements of HIPAA (US) and PIPEDA (Canada). If not, all software risks are identified, evaluated, and mitigated.

   Actual output: System design fully followd HIPAA (US) and PIPEDA (Canada).

   Result: Pass

2. NFR-LR2

   Initial State: ISO 13485 documentation is in place, development artifacts are available (verification and validation plans, risk analysis reports, requirements, design documents, test plans, etc.).

   Input/Condition: Development artifacts, processes (software lifecycle management process, design review process, etc.).

   Expected Output/Result: All development processes align with the requirements of ISO 13485 and if not, all software risks are identified, evaluated, and mitigated.

   Actual Output: Development artifacts and processes follow SO 13485 guidelines.

   Result: Pass

4.9 Health and Safety

1. NFR-HS1

   Initial State: User roles are configured within the system, including Radiologist, Clinician, etc.

Input/Condition: AI-generated report, Radiologist user account.

Expected Output/Result: Logs including radiologist's review action (confirm/reject) with a timestamp.

Actual Output: Logs including doctor's action were shown in docker as the system is running.

Result: Pass

2. NFR-HS2

Initial State: The AI system is deployed and accessible to radiologists, clinicians, and other users.

Input/Condition: Access the AI-generated diagnostic report and view the user interface displaying AI results.

Expected Output/Result: Disclaimer is prominently displayed and easy to read.

Actual Output: A disclaimer box is displayed.

Result: Pass

5 Comparison to Existing Implementation

5.1 Existing Projects

- harrisonchiu/xray
- N8THEPL8/ChestLenseAI
- PLAN-Lab/CheXRelFormer

5.2 Project Structure

Project Name	**FrontEnd**	**BackEnd**	**Data Base**	**Cloud**	**Deployment**
Our Capstone	React.js	Flask API	Amazon S3	AWS	Docker
xray	N/A	N/A	N/A	N/A	Jupyter Notebook
ChestLenseAI	HTML & CSS	Flask API	N/A	N/A	Python
CheXRelFormer	N/A	N/A	N/A	N/A	Shell Command

5.3 Datasets

Project Name	Dataset	Size	Classes	Link
Our Capstone	MIMIC-CXR-JPG 2.0.0	557.6 GB	9: Lung Opacity, Pleural Effusion, Atelectasis, Enlarged Cardiac Silhouette, Pulmonary Edema/Hazy Opacity, Pneumothorax, Consolidation, Fluid Overload/Heart Failure, Pneumonia. 3: No Change, Improved, Worsened.	https://physionet.org/content/mimic-cxr-jpg/2.0.0/
xray	Chest-xray14	42.0 GB	14: Atelectasis, Cardiomegaly, Consolidation, Edema, Effusion, Emphysema, Fibrosis, Hernia, Infiltration, Mass, Nodule, Pleural Thickening, Pneumonia, Pneumothorax.	https://nihcc.app.box.com/v/ChestXray-NIHCC/folder/37178474737
ChestLenseAI	MIMIC-CXR-JPG 2.0.0	557.6 GB	6: Atelectasis, Cardiomegaly, Consolidation, Edema, No Finding, Pleural Effusion.	https://physionet.org/content/mimic-cxr-jpg/2.0.0/
CheXRelFormer	MIMIC-CXR-JPG 2.0.0, MIMIC-III 1.4	557.6 GB, 6.2 GB	3: No Change, Improved, Worsened	https://physionet.org/content/mimic-cxr-jpg/2.0.0/

5.4 Architectures

Project Name	Neural Network	Configuration	Link (graph) (paper)
Our Capstone	DETR	MLP	https://viso.ai/wp-content/uploads/2024/02/DETR-Architecture.jpg https://arxiv.org/pdf/2005.12872
xray	ResNet	ResNet-50	https://i.ytimg.com/vi/woEs7UCaITo/maxresdefault.jpg https://arxiv.org/pdf/1512.03385
ChestLenseAI	DenseNet	DenseNet-121	https://pytorch.org/assets/images/densenet1.png https://arxiv.org/pdf/1608.06993
CheXRelFormer	ViT	MLP	https://www.researchgate.net/publication/383905431/figure/fig3/AS:11431281290331182@1731595235002/Structure-of-the-backbone-PVTv2.ppm https://arxiv.org/pdf/2106.13797

5.5 Summary

Project Name	Summary	Reference Project	Paper
Our Capstone	This project encompasses the entire product lifecycle, integrating Frontend, Backend, Cloud Infrastructure, and a CI/CD pipeline. It is designed to support the detection of diseases, track the progression of conditions over time, and generate AI-driven diagnostic reports for healthcare professionals.	https://github.com/McMasterAIHLab/CheXDetector	https://papers.miccai.org/miccai-2024/paper/3269_paper.pdf
xray	This project only contains backend structure for the AI model, which uses a smaller dataset compared to other projects and uses the resnet-50 network for disease classification.	https://github.com/LalehSeyyed/CheXclusion	https://arxiv.org/pdf/2003.00827v2
ChestLenseAI	This project uses a minimal Frontend and applies the DenseNet-121 pre-trained model to detect diseases in chest X-ray images.	https://github.com/LaurentVeyssier/Chest-X-Ray-Medical-Diagnosis-with-Deep-Learning/tree/main	https://arxiv.org/pdf/1711.05225
CheXRelFormer	This project is purely focused on the backend ViT model, it does not use a pre-trained network for disease progress between two x-ray images. It is a very advanced (PhD) project which is developed with no prior related research.	N/A	N/A

6 Unit Testing

7 Changes Due to Testing

8 Automated Testing

To run automated tests within a developer's local environment, a developer can execute a command to build the project. This process ensures that all dependencies are correctly set up and runs the automated tests for both Python and React.js.

If a developer wants to run only the tests, they can navigate to the appropriate directory where the tests are located and execute the relevant test command. For Python, this typically involves running pytest or Flake8, while for React.js, Jest is used.

We have tests that are executed using AWS services, with results and logs stored in Amazon S3 for easy access and review. This setup ensures that test results are centralized and accessible across for the team. The automation helps maintain code quality, catch errors early, and streamline the deployment process.
From the repository's perspective, tests are executed using Github CI/CD to maintain code quality and stability. Both linting and compilation are performed using the same commands that a developer would execute in their local environment. Specifically, we use Flake8 for Python linting and Jest for testing React.js components.

These tests are triggered when a new pull request is made to the main branch. If any test fails, merging is blocked until the issues are resolved. This ensures that only verified, high-quality code is integrated. Additionally, a compiler workflow runs after merging into the main branch to detect any errors or unintended changes in code behavior.

If any test or workflow fails, the detailed logs provide valuable insights into the reason for failure. This allows developers to quickly diagnose and address issues, preventing regressions and ensuring a smooth development process. By enforcing these automated checks, we maintain code consistency, reduce

bugs, and improve overall project reliability.

9 Trace to Requirements

10 Trace to Modules

For additional information about our modules, please refer to our MG.

Requirement ID	Test ID
M1	PR2,OE3
M2	PR2,OE2
M3	PR1, PR3, MS2
M4	PR1, PR3, MS2
M5	UH2
M6	LF1, LF2, UH1, CR1
M7	LF1, LF2, UH1, CR1
M8	LF1, LF2, UH1, CR1
M9	LF1, LF2, UH1, CR1
M10	MS1, CR1, HS1, HS2
M11	PR1, PR3, MS1, MS2
M12	PR1, PR3, MS1, MS2
M13	MS2
M14	UH2, OE1, SR1, LR1, LR2

Table 5: Trace to Modules Table

11 Code Coverage Metrics

References

Appendix — Reflection

What went well while writing this deliverable?

One aspect that went well in this deliverable was evaluating the functional and non-functional requirements. This process was relatively simple because we could trace these requirements back to our Verification and Validation (VnV) Plan and Software Requirements Specification (SRS) report. This ensured consistency and alignment with our initial project documentation.

What pain points did you experience during this deliverable, and how did you resolve them?

One major challenge was handling unexpected deviations from the VnV Plan. Some tests and validation activities required adjustments because real-world implementation differed from our initial assumptions. To address this, we removed certain tests and modified others to better fit the project scope. Another challenge was testing non-functional requirements (NFRs). To resolve this, our team decided to use an external tester via a Google Form. Additionally, creating unit tests for the code was a significant task. However, by effectively distributing the workload among team members, we successfully developed and passed all necessary tests.

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?

Certain parts of this document were directly influenced by discussions with clients or proxies. For example, verifying functional and non-functional requirements was guided by user needs, and stakeholder feedback shaped some test cases and validation methods to account for real-world constraints.
However, not all sections were based on client input. Technical implementation details, such as internal testing methodologies and the choice of specific testing frameworks, were determined by the development team. This distinction exists because clients typically focus on high-level functionality and usability, while the development team makes technical decisions regarding implementation and testing strategies.

In what ways was the Verification and Validation (VnV) Plan different from the activities that were actually conducted for VnV?

Our VnV Plan differed from the actual activities conducted for several reasons:

- Some verification steps outlined in the plan proved impractical due to technical constraints, requiring modifications to testing approaches.
- As development progressed, we identified gaps in the original plan, leading to the addition of test cases for overlooked edge cases.
- The order of validation activities changed due to development dependencies and evolving priorities.

These deviations were necessary due to evolving project requirements, stakeholder feedback, and resource limitations, all of which influenced the scope and execution of testing efforts.
While deviations were unavoidable, they provided valuable insights for improving future VnV planning. To better anticipate such changes, our team could:

- Incorporate iterative planning.
- Maintain flexibility in testing approaches.
- Establish contingency measures for unexpected challenges.
- Conduct more frequent stakeholder consultations to identify potential adjustments earlier.

If no modifications were required, it would indicate that our team accurately predicted the necessary effort and tasks to generate sufficient evidence of system quality. However, most teams experience some level of deviation from their original VnV Plan, making adaptability a key factor in ensuring successful validation and verification.