System Verification and Validation Plan for Scanalyze AI

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

April 4, 2025

Revision History

Date	Author	Notes
4 November 2024	Harrison	Section 4
4 November 2024	Hamza	Section 5
4 November 2024	Gurnoor	Section 1-3
4 November 2024	Jared	Section 3, 5
4 November 2024	Ahmad	Section 5
4 April 2024	Jared, Harrison	Update VnVPlan to address feedback from Github Issues and TA feedback

Contents

List of Tables

1 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

This document outlines the verification and validation (VnV) plan for the project's proposed solution. It begins with a summary of the software's general functions, the objectives of the VnV plan (including in-scope and out-of-scope elements), and references relevant documentation.

It lists the VnV team members and describes the process for verifying the SRS, design, VnV plan, and implementation, including the automated testing tools used. The VnV roadmap starts with verifying the SRS, followed by the design, VnV plan, and implementation. After these steps, the software will undergo validation, with milestones achieved through automated tools and guidance from the project supervisor. It also shows the tests done (system and unit tests) to each major module to verify its functionality and output.

2 General Information

2.1 Summary

The software being developed is the **Chest Scan** AI diagnostic tool, designed to assist radiologists by automating the classification of chest X-ray images and generating structured diagnostic reports. The system leverages convolutional neural networks (CNNs) to detect and classify multiple thoracic diseases including pneumonia, cardiomegaly, and effusions within a single scan. This multi-class classification capability enhances diagnostic efficiency and helps radiologists manage large volumes of medical imaging. The tool will include user authentication, heatmap visualization, confidence scoring, and a diagnostic report through a web interface.

2.2 Objectives

The V&V Plan for the Chest Scan system targets three major goals: diagnostic reliability across disease categories, secure handling of sensitive clinical data, and intuitive system usability. Each objective is measurable and traceable to the system's core requirements. They guide testing for both functional correctness and compliance with healthcare regulations.

1. **AI Model Accuracy and Diagnostic Reliability**
    - Verify that the CNN model can classify chest X-rays across multiple disease labels (e.g., pneumonia, cardiomegaly, edema) with high precision and recall per class.
    - Confirm that class-specific false positive and false negative rates remain below thresholds defined in the SRS (e.g., $< 10\%$ FP for cardiomegaly).
    - Ensure that heatmaps and confidence scores correspond to the regions of the X-ray that support each prediction, enabling radiologist verification.
2. **Data Security and Access Control**
    - Confirm that all personally identifiable health data is encrypted both at rest and during transmission, using standards such as AES-256 and HTTPS.

- Verify that only authorized users (e.g., clinicians, researchers) can access system outputs via token-based authentication.
- Validate that the system is aligned with HIPAA and PIPEDA standards by conducting compliance reviews and verifying that audit logs, encryption, and access control procedures are implemented correctly.

3. **Usability and System Accessibility**
   - Verify that radiologists can intuitively upload images, receive multi-label predictions, view corresponding heatmaps, and download PDF reports.
   - Confirm that the UI supports readability, keyboard navigation, and mobile responsiveness for accessibility.
   - Validate seamless integration with imaging systems (e.g., PACS) for DICOM file upload and metadata retrieval.

2.2.1 Out of Scope for the V&V Plan

While essential to future development, the following areas are currently beyond the scope of this V&V plan:

- **Industry-Standard UI/UX Optimization:** The plan does not include formal usability heuristics testing or A/B interface comparisons.
- **Third-Party Library Validation:** The plan assumes correctness of well-established open-source ML libraries (e.g., PyTorch, TensorFlow).
- **Full Load and Stress Testing:** The MVP is scoped for single-user workflows; multi-user scalability testing will be deferred.

2.3 Challenge Level and Extras

This project is considered high challenge due to its simultaneous emphasis on medical correctness, model generalizability, regulatory compliance, and usability.

AI Complexity: Building a CNN capable of handling multi-label classification on chest X-rays requires careful dataset curation, bias control, and threshold tuning.

Healthcare Compliance: Ensuring secure data handling practices compliant with HIPAA and PIPEDA adds architectural and legal overhead.

Clinical Relevance: Presenting diagnostic results in a format interpretable by radiologists, including justification visualizations (e.g., heatmaps), imposes a high bar for usability and interpretability.

2.3.1 Extras

If project capacity permits, the team will pursue the following enhancements:

- **Usability Survey & feedback Report:** Designed and conducted a usability survey to gather user feedback on our website, then compiled and analyzed the responses in a structured report for deeper insights.
- **Research Paper and Documentation:** Full technical documentation and a research summary will be published detailing model design, architecture, training pipeline, and V&V outcomes.

2.4 Relevant Documentation

The following deliverables provide traceable, testable definitions for both requirements and architecture:

- **Software Requirements Specification (SRS):** Defines functional (e.g., FR1-FR6) and nonfunctional (e.g., SR1-SR3, AR1-AR3) requirements with fit criteria and rationale.
- **Problem Statement Document:** Articulates the motivation, target stakeholders, and scope of the CNN chest X-ray diagnostic system.
- **Module Guide (MG):** Describes the modular breakdown of the system, including the AI engine, API controller, frontend renderer, and report generator.
- **Module Interface Specification (MIS):** Specifies interfaces, state transitions, and exported functions of each module. Used to define V&V unit test boundaries.

3 Plan

In this section, we will lay out the roadmap for ensuring each part of the project meets the intended standards through verification and validation. Responsibilities are assigned to team members for each task within the V&V process. This plan will address how we will confirm that the project's requirements, design documents, V&V plan, and final implementation align with the specified criteria. Tools and automated testing solutions that support our verification efforts will also be outlined. Additionally, a detailed process for validating the software's functionality and performance will be included. Our verification tasks will be presented in a structured sequence, moving from requirements analysis to design inspection, implementation testing, and review of the V&V process itself. Requirements verification will continue throughout development, with design checks occurring as the code is produced. Validation of the implementation will be an ongoing effort, eventually leading to a proof-of-concept phase and final project demonstrations.

3.1 Verification and Validation Team

VnV Team Member Name	Summary of Role(s)
Amirhossein Sabour	Advisor; oversees V&V strategy, ensures documentation quality, and provides final validation input. Reviews model performance metrics and compliance with clinical AI validation standards.
Other Design Teams	Peer reviewers. Raise model validation issues (e.g., poor class-wise recall), documentation inconsistencies, or usability oversights through structured feedback and bug reports.
Gurnoor Bal	Leads verification of CNN model outputs, including confusion matrix analysis, precision/recall per disease class, and threshold sensitivity testing. Also supports validation of image upload and result display pipelines.
Jared Paul	Manages dataset quality verification tasks. Reviews for bias, annotation mismatches, and underrepresented disease categories. Conducts fairness and generalization checks.
Ahmad Hamadi	Responsible for backend logic and API validation. Verifies data flow integrity, image pre-processing, and robustness of prediction-serving endpoints. Also participates in Grad-CAM heatmap correctness validation.
Hamza Issa	Performs V&V of the web interface, focusing on frontend validation, usability compliance, and PDF generation. Participates in cross-checking classification results against reports.
Harrison Chiu	Oversees integration testing and multi-label output validation. Coordinates end-to-end test execution for model prediction, heatmap, report pipeline and leads usability testing sessions with stakeholders.

3.2 SRS Verification Plan

The SRS Verification Plan for the "Chest Scan" diagnostic tool ensures that all specified requirements are complete, accurate, and verifiable, particularly in the context of multi-label disease classification. This section outlines the approach to confirm that the SRS document meets quality standards, providing a solid foundation for subsequent design, implementation, and testing phases.

3.2.1 Objectives

The objectives of this SRS Verification Plan are to:

1. **Ensure Requirement Completeness:** Confirm that all essential functionalities, such as multi-class image classification, model confidence estimation, heatmap visualizations, and diagnostic report generation, are fully and clearly specified in the SRS.
2. **Validate Requirement Clarity:** Verify that each requirement, including classification performance metrics (e.g., precision, recall, F1-score), is unambiguous and

testable.

3. **Check Requirement Feasibility:** Assess that all technical features-especially those related to deep learning models-are realistically achievable given dataset availability and system constraints.
4. **Verify Traceability:** Ensure that each requirement can be traced through to design modules, test cases, and validation outputs.

3.2.2 Verification Checklist

To comprehensively verify the SRS, the following key elements will be checked:

1. **Functional Requirements**
   - Disease detection and classification across multiple diagnostic categories (e.g., pneumonia, pleural effusion, cardiomegaly, etc.).
   - Accuracy thresholds for class-wise detection defined using symbolic constants (e.g., `MIN_ACCURACY_THRESHOLD`).
   - Confidence score computation and annotation with classification results.
   - Integration of interpretability tools such as Grad-CAM heatmaps.
   - Diagnostic report generation with location-based annotations and condition summaries.
2. **Nonfunctional Requirements**
   - **Performance:** Symbolic benchmarks for classification latency, throughput, and expected accuracy ranges under load.
   - **Security:** HIPAA-compliant handling of sensitive data including encryption-at-rest, secure data transfer, and access control policies.
   - **Usability:** UI clarity, response times for clinical review, and interpretability of classification results.
   - **Reliability:** Uptime requirements, fallback behaviors in case of model failure or data corruption.
3. **Constraints**
   - Use of PyTorch and Grad-CAM as mandatory frameworks and tooling.
   - Compliance with public dataset licenses (e.g., CheXpert, MIMIC-CXR, NIH) and hospital system compatibility (PACS/HIS).
4. **Traceability and Identification**
   - All requirements assigned unique identifiers (e.g., `FR1`, `NFR2`) mapped to modules and future test cases.

- Inclusion of backward and forward traceability from requirements to test outputs and system modules.

5. **Glossary and Terminology**
   - Standardization of clinical terms (e.g., infiltrates, nodules), technical terms (e.g., Grad-CAM, ROC-AUC), and abbreviations.
6. **Assumptions and Dependencies**
   - Availability of annotated chest X-ray datasets (with multi-class labels) assumed.
   - Dependence on high-performance hardware (e.g., GPU-enabled environments) for real-time inference.

3.2.3 Methods

The following methods will be used to verify the SRS:

- **Requirement Review Meetings:** Cross-functional team discussions with domain experts to confirm all requirements support multi-class clinical use cases.
- **Traceability Matrix Construction:** All SRS items mapped to functional and non-functional test cases for validation and unit testing.
- **Cross-Referencing with MG and MIS:** Ensures decomposition of requirements into software modules and well-defined interfaces for each responsibility.

3.2.4 Verification Criteria

The SRS will be considered verified if:

- Requirements meet completeness, consistency, and clarity standards suitable for medical AI tools.
- Each requirement is covered in the test case traceability matrix with associated validation criteria (e.g., precision, recall, Grad-CAM heatmap visibility).
- All feedback from domain reviewers and stakeholders (e.g., radiologists, AI engineers) is addressed and resolved.

By following this updated SRS Verification Plan, the team will ensure that the diagnostic system's architecture supports reliable, explainable, and secure multi-disease classification suitable for healthcare deployment.

3.3 Design Verification Plan

This Design Verification Plan focuses on verifying that the system's modular structure and interface specifications, as outlined in the Module Guide (MG) and Module Interface Specification (MIS), align with the requirements in the SRS. Each module's functionality, integra-

tion, and adaptability to potential changes will be reviewed to ensure the system is robust, maintainable, and ready for clinical use.

3.3.1 Objectives

The objectives of this Design Verification Plan are to:

1. **Verify Modular Consistency with Requirements:** Ensure that each module in the MG fulfills its corresponding SRS requirements, including image processing, multi-class classification, Grad-CAM visualization, and secure access.
2. **Validate Information Hiding and Encapsulation:** Confirm that modules encapsulate implementation details (e.g., CNN model parameters, data loaders) to support abstraction and maintainability.
3. **Check Interface Compatibility:** Verify that module interfaces in the MIS support accurate data exchange between components like the frontend UI, backend API, and CNN classification engine.
4. **Assess Anticipated Changes and Traceability:** Ensure that anticipated design changes (e.g., swapping the model architecture or modifying output schema) are localized and traceable.

3.3.2 Verification Checklist

The following checklist covers key design aspects to be verified based on the MG and MIS documents:

1. **Module Decomposition**
   - Modules follow the information-hiding principle and align with layered design (e.g., ML layer, data layer, UI layer).
   - CNN architecture module includes breakdown of input layers, convolutional stacks, activations (e.g., ReLU), pooling layers, and softmax/multi-label classifiers.
   - Traceability matrix maps SRS classification requirements (e.g., FR1-FR4) to CNN Model and Grad-CAM modules.
2. **Module Interfaces (MIS)**
   - CNN module input interface accepts preprocessed X-ray tensors with consistent shape (e.g., `B x 1 x 224 x 224`).
   - Output interface returns a multi-class probability vector, interpretable labels, and Grad-CAM heatmaps.
   - Modules export constants like `CLASS_LABELS`, `CONFIDENCE_THRESHOLD`, ensuring no hard-coded literals.
3. **Anticipated and Unlikely Changes**

- Anticipated: Change in CNN architecture (e.g., swapping DenseNet with ResNet) should be confined to the ML module.
- Unlikely: Change to data modality (e.g., CT scans instead of X-rays) would affect preprocessing and UI assumptions-documented as unlikely to reduce over-design.

4. **Traceability and Use Hierarchy**
   - Each design element traced to one or more SRS requirements using the traceability matrix.
   - UI modules depend on Backend API, which depends on ML model, which in turn depends on DataLoader and Preprocessing modules (clean downward hierarchy).
5. **User Interface Design (if applicable)**
   - Image upload UI and heatmap visualizations are verified for compatibility with backend output format.
   - Diagnostic report UI tested for multi-class display layout (i.e., stacked or tabular outputs for multiple disease labels).

3.3.3 Methods

The following methods will be used to verify the design:

- **Design Walkthroughs:** Systematic reviews with the team to inspect modular boundaries, data flow, and CNN structure alignment with requirements.
- **Consistency Checks with SRS:** Ensure module responsibilities match SRS requirements, including disease classification, heatmap generation, and frontend reporting.
- **Interface Testing:** Simulate API calls or UI interactions using mock data to verify each module interface behaves as expected.
- **Traceability Matrix Review:** Confirm each FR/NFR is mapped to design modules and test plans for coverage and traceability.

3.3.4 Verification Criteria

The design is considered verified if:

- The CNN model structure, activation pipeline, and outputs align with SRS expectations and clinical interpretability.
- Modules demonstrate proper encapsulation of model logic, UI routines, and data transformations.
- Interface definitions are consistent, conflict-free, and aligned with MIS documentation.
- All traceability links between requirements and modules are complete and validated.

This plan ensures the system's design is ready for implementation, supporting flexibility, traceability, and correctness for real-world clinical deployment.

3.4 Verification and Validation Plan Verification Plan

This Verification and Validation (V&V) Plan Verification Plan outlines the process to ensure that the V&V document itself is thorough, accurate, and effectively designed to support all project goals. By verifying this V&V document, we aim to confirm that all sections—objectives, scope, methods, and criteria—are clearly defined and cover the requirements, design, and implementation needs outlined in the SRS, MG, and MIS documents.

3.4.1 Objectives

The objectives of this V&V Plan Verification Plan are to:

1. **Ensure Completeness of Verification and Validation Processes:** Verify that this V&V Plan includes all necessary sections to thoroughly validate the tool's requirements, design, and implementation.
2. **Confirm Clarity and Traceability:** Ensure that each section is clearly written, unambiguous, and easy to follow for all project stakeholders, with traceability to project documents.
3. **Validate Feasibility of Methods:** Check that the methods and criteria specified in the V&V Plan are feasible given project resources and timelines.
4. **Identify Gaps and Overlaps:** Ensure that all elements of the system are addressed without unnecessary repetition, minimizing gaps or redundant validations.

3.4.2 Verification Checklist

The following checklist identifies key elements in the V&V Plan to verify for completeness and accuracy:

1. **Objectives and Scope**
    - Verify that the V&V Plan includes well-defined objectives and scope covering all essential elements (SRS, MG, MIS).
    - Confirm that the plan covers functional and nonfunctional requirements, with a clear boundary on what is out of scope.
2. **Verification and Validation Methods**
    - Ensure that each section (e.g., SRS Verification Plan, Design Verification Plan) includes clearly defined methods and criteria for verifying requirements, design, and implementation.
    - Confirm that the V&V methods align with the constraints and objectives laid out in the project's SRS.

3. **Traceability and Consistency**
    - Check that the V&V Plan is traceable to all primary project documents (SRS, MG, MIS) with appropriate references, ensuring alignment and consistency across documents.
    - Ensure that each V&V Plan section is traceable back to specific requirements or design aspects from the SRS and MG, ideally via traceability matrices.
4. **Feasibility and Practicality of Methods**
    - Confirm that each verification and validation method in the V&V Plan is achievable within the project's resources, timeline, and technical constraints.
    - Check that tools, data, and stakeholder inputs needed for each V&V process are specified and accessible.
5. **Criteria for Success**
    - Ensure that the V&V Plan defines clear criteria for determining the success of each verification and validation step.
    - Verify that these criteria are realistic and aligned with project requirements, design goals, and intended system functionality.

3.4.3 Methods

To verify the V&V Plan:

- **Document Review Meetings:** Conduct meetings with project stakeholders and team members to review each section of the V&V Plan for clarity, feasibility, and traceability.
- **Cross-Referencing with Project Documents:** Check each verification and validation method against the SRS, MG, and MIS to confirm alignment and completeness.
- **Feasibility Assessment:** Evaluate the methods and tools proposed in the V&V Plan to ensure they are practical and achievable with available resources.

3.4.4 Verification Criteria

The V&V Plan will be considered verified if:

- All sections are complete, clearly written, and aligned with project documents (SRS, MG, MIS).
- The verification and validation methods are feasible and cover all essential elements.
- Stakeholders confirm that the V&V Plan is practical and provides a robust approach for testing and validating the tool.

This Verification Plan ensures that the V&V Plan itself is reliable, thorough, and ready to support a comprehensive validation process for the "Chest Scan" diagnostic tool.

3.5 Implementation Verification Plan

This Implementation Verification Plan outlines the steps to confirm that the "Chest Scan" diagnostic tool's final implementation accurately reflects its design, fulfills all specified requirements, and operates as intended in realistic clinical environments. The plan emphasizes evaluating model correctness, user experience, dataset adaptability, and multi-label classification robustness. It also verifies that the system supports interpretability and bias-reduction techniques across various disease classes.

3.5.1 Objectives

The objectives of the Implementation Verification Plan are to:

- **Verify Functional Accuracy:** Ensure that each implemented feature—including disease detection, diagnostic report generation, and heatmap generation—performs according to its design and aligns with the SRS functional requirements.
- **Confirm Model Stability and Reliability:** Assess the CNN's stability across training sessions using different dataset splits, ensuring that performance (e.g., accuracy, precision, recall) consistently meets symbolic thresholds (e.g., `ACC_THRESHOLD`).
- **Validate Dataset Adaptability and Generalization:** Confirm that the model performs well on real-world datasets (e.g., CheXpert, MIMIC-CXR) beyond its training data, ensuring broad applicability.
- **Ensure Security and Usability Compliance:** Verify encryption protocols, access control logic, and that the user interface is intuitive and accessible for clinical users.
- **Evaluate Bias Mitigation:** Verify that training pipelines implement data balancing, augmentation, or stratification techniques to reduce biases from class imbalance or dataset skew.

3.5.2 Verification Methods

The following methods will be employed to verify implementation:

- **Unit Testing:** Each backend module, including image preprocessing, classification, report formatting, and UI display components, will be tested using automated unit tests tied to unique SRS identifiers.
- **Integration Testing:** Functional connections between subsystems (e.g., frontend upload $\rightarrow$ backend classifier $\rightarrow$ report UI) will be validated. These tests confirm correct data transfer, error handling, and system communication flows.
- **System Testing:** The entire platform will be tested in a near-production environment. Evaluation will include:
  - Accuracy metrics (precision, recall, F1-score) on a holdout dataset.
  - Classification confusion matrix per disease class.

- Model explainability checks using Grad-CAM overlays.
- Load testing with multiple uploads to verify concurrency.
- HIPAA-aligned encryption and access audits.

3.5.3 Verification Criteria

The implementation will be considered verified if:

- **Pass Rate:** 100% of unit and integration tests pass without critical or high-severity errors.
- **Performance Thresholds:** Multi-class CNN meets symbolic metrics such as `ACC_THRESHOLD`, `PREC_THRESHOLD`, and `RECALL_THRESHOLD` for top-priority disease classes.
- **Model Stability:** Results from multiple runs with different seeds and data splits show consistent classification performance within acceptable variance.
- **Bias Detection and Mitigation:** Validation shows no statistically significant performance drop between underrepresented and overrepresented classes (p-value ¡ 0.05).
- **Clinical Usability and Security:** Stakeholders confirm that the UI is intuitive, reports are interpretable, and data access follows security standards (AR2, SR3).

By following this Implementation Verification Plan, the project team will ensure that the final deployed system behaves reliably across use cases, meets clinical needs, adapts to real-world data, and remains interpretable and secure throughout operation.

3.6 Automated Testing and Verification Tools

Automated testing will ensure consistent verification of the "Chest Scan" diagnostic tool's functionality, classification accuracy, security, and fairness throughout its development. Given the complexity of a multi-class CNN in a clinical setting, automated tools will support reproducibility, precision monitoring, and protection against model drift or biased outputs. Tools will be selected to optimize testing efficiency, track model behavior, and enforce project constraints through a continuous integration pipeline.

3.6.1 Objectives

The goals of automated testing are to:

- **Enhance Testing Efficiency:** Use automated testing suites to reduce manual verification time and catch regressions early in the development process.
- **Support Continuous Integration and Deployment (CI/CD):** Automate testing workflows to verify code and model behavior on each update.
- **Track Model Performance and Fairness:** Implement evaluation scripts that monitor classification metrics (e.g., precision, recall), dataset bias, and output disparities across disease labels.

- **Evaluate Adversarial Robustness:** Test model performance against perturbed or low-quality images to ensure reliability under real-world imaging variability.

3.6.2 Tools and Methods

The following tools and methods will be employed during the project:

- **Unit Testing:**
  - `pytest`, `unittest`: Validate core functionalities such as preprocessing routines, model inference, and data formatting logic.
  - Test metrics include symbolic constants like `ACC_THRESHOLD` and `RECALL_THRESHOLD` defined in alignment with SR1.
- **Model Evaluation Scripts:**
  - Custom Python scripts to automatically compute:
    - Multi-class confusion matrix and per-class precision/recall/F1-scores.
    - Grad-CAM heatmap visual inspection and feature localization checks.
    - Bias detection using demographic parity or subgroup accuracy tests (for future scalability).
  - Tools like `scikit-learn`, `captum`, and `Fairlearn` will be used for metric calculation and fairness evaluation.
- **UI and Integration Testing:**
  - `Selenium`: Automated interaction testing for the frontend, validating image uploads, report visibility, and user flow.
  - `React Testing Library`: Unit tests for frontend components, such as result modals and image previews.
- **Continuous Integration Pipeline:**
  - `GitHub Actions`: Executes linting, test suites, and evaluation scripts on each pull request.
  - `Flake8`, `Black`, `ESLint`: Enforce Python/JS code style and formatting rules.
  - Auto-reporting system logs test results and accuracy drops to alert developers of performance degradation.

3.6.3 Success Criteria

Automated testing will be considered effective if:

- The CI pipeline completes without failures across unit, integration, and evaluation stages.

- Model performance consistently exceeds defined symbolic thresholds (`ACC_THRESHOLD`, `F1_THRESHOLD`) during automated testing.
- Bias and robustness test scripts identify performance disparities early and allow corrective interventions (e.g., resampling or retraining).
- Functional UI tests ensure that no deployment breaks the image upload or report display pipeline.

Through this automation framework, the project will maintain high-quality, reproducible testing while improving classification accuracy, system robustness, and fairness in clinical AI diagnostics.

3.7 Software Validation Plan

This Software Validation Plan outlines the approach to confirm that the "Chest Scan" tool meets all functional and nonfunctional requirements, particularly in its role as a multi-class classification system for chest X-ray diagnostics. The plan incorporates quantitative and qualitative methods to ensure accuracy, fairness, usability, and clinical applicability.

External datasets (e.g., MIMIC-CXR, CheXpert) will be used to validate classification performance across multiple disease categories, while user testing with radiologists will provide real-world feedback on system usability and workflow integration.

3.7.1 Objectives

The objectives of the Software Validation Plan are to:

- **Validate Requirement Alignment:** Confirm that all functional requirements (FR1-FR4) and nonfunctional requirements (SR0, SR1, SR3) are fulfilled through structured task-based evaluations and stakeholder reviews.
- **Assess Multi-Class Classification Performance:** Use independent validation sets to assess precision, recall, F1-score, and confusion matrix per disease class, ensuring that model outputs meet diagnostic standards.
- **Gather Stakeholder Feedback:** Leverage Rev 0 and Rev 1 demonstration sessions with supervisors, clinicians, and developers to incorporate iterative feedback and refine the tool.
- **Confirm Clinical Usability:** Validate that the web interface and reporting pipeline align with radiologist workflows, supporting intuitive review of AI-generated reports and Grad-CAM visualizations.

3.7.2 Validation Methods

The following methods will be employed to confirm software effectiveness:

- **Task-Based Inspections:** Perform test cases simulating real-world scenarios such as uploading chest X-rays, viewing disease predictions, and interpreting Grad-CAM heatmaps. These tasks ensure diagnostic and interpretability goals are met.
- **Model Validation on External Datasets:** Run the trained CNN on held-out CheXpert and MIMIC samples and compute multi-class confusion matrices, per-class F1-scores, and overall accuracy to evaluate alignment with SR0 and SR1 thresholds.
- **Demo and Review Sessions:** Present working implementations in Rev 0 and Rev 1, gathering feedback from supervisors and potentially clinical advisors to assess requirement completeness.
- **User Testing with Radiologists:** Conduct observational usability studies, asking participants to navigate the tool, interpret results, and report on clarity, confidence, and workflow impact.

3.7.3 Validation Criteria

The software will be considered validated if:

- CNN model achieves symbolic accuracy and precision thresholds (e.g., `ACC_THRESHOLD` ≥ 0.90) across disease classes on independent datasets (SR0).
- Feedback from stakeholders (Rev 0/1) confirms compliance with both functional and nonfunctional expectations.
- Radiologist usability feedback indicates that the interface is interpretable, reliable, and clinically usable in simulated workflows.

This validation plan ensures the tool's real-world readiness through comprehensive evaluation of both performance metrics and user experience, aligning with the project's clinical goals and SRS-defined criteria.

4 System Tests

This subsection presents the system test cases categorized by areas of functionality. Each test validates a key feature described in the functional requirements. Tests span input handling, multi-class disease identification, report generation, visualization, user access, data protection, and authorization. References to SRS functional requirements are included in the derivations.

4.1 Tests for Functional Requirements

4.1.1 Handling Input Data

FRTC1
Title: Chest X-ray Image Input Acceptance
Control: Manual

Initial State: The system is initialized and ready to receive input through the web interface.
Input: A valid chest X-ray image in supported formats (DICOM, JPEG, or PNG).
Output: The image is successfully accepted, read, and queued for analysis with no errors or crashes.
Test Case Derivation: The test confirms the system meets baseline input handling by successfully accepting valid file types. The expected output validates the upload and queuing logic for correctly formatted chest X-ray images without backend or UI errors.
How the test will be performed:

1. Launch the system and ensure it's ready to receive inputs.
2. Use the upload interface to provide a valid chest X-ray image.
3. Confirm the image is accepted into the system for processing.
4. Monitor for any frontend or backend anomalies during the upload.

FRTC2
Title: Invalid Chest X-ray Image Format Rejection
Control: Manual
Initial State: The system is initialized and awaiting file upload.
Input: A file in an unsupported or corrupted format (e.g., .txt, .docx, or corrupted .jpg).
Output: The system rejects the file with a descriptive error message and logs the failure.
Test Case Derivation: The test ensures the system handles invalid or malformed inputs gracefully by displaying clear user-facing errors and suppressing unintended backend behavior. The expected output confirms validation safeguards are enforced to block non-image files or corrupted uploads.
How the test will be performed:

1. Attempt to upload invalid image files.
2. Confirm that the UI blocks the upload and displays an appropriate error message.
3. Verify error handling is logged in the system backend.
4. Ensure no processing pipeline is triggered.

4.1.2 FR2

FRTC3
Title: Image Preprocessing Enforcement for CNN Compatibility
Control: Manual
Initial State: System is running and ready for image upload.
Input: A raw chest X-ray image with incorrect resolution and color format.
Output: The system resizes to MODEL_INPUT_DIMENSIONS, normalizes pixel values, converts to grayscale if needed, and logs errors for failures.
Test Case Derivation: The expected output is justified by the CNN model's strict input

requirements-ensuring valid data improves model accuracy and prevents runtime errors.
How the test will be performed:

1. Upload an unprocessed color X-ray image.
2. Confirm the system performs resizing, normalization, and grayscale conversion.
3. Upload a malformed image to verify error logging and halt in processing.

4.1.3 FR3

FRTC4
Title: Disease Classification with Confidence Scores
Control: Automatic (with manual verification)
Initial State: CNN model is deployed and active.
Input: Chest X-ray images from the validation dataset.
Output: For each image, the system returns disease predictions with confidence scores greater than 0 and less than 1; per-class accuracy meets or exceeds 60%.
Test Case Derivation: Predictions must include both labels and confidence scores to meet the fit criterion, while model accuracy above 60% ensures diagnostic reliability.
How the test will be performed:

1. Load a labeled validation dataset.
2. Run the CNN model to obtain disease predictions and confidence scores.
3. Verify the format and range of confidence scores.
4. Compare predictions against ground truth labels.
5. Calculate accuracy and confirm it meets or exceeds 60%.

4.1.4 FR4

FRTC5
Title: Ambiguous Image Classification (Multi-Label)
Control: Automatic (verified manually)
Initial State: Model and supporting UI loaded.
Input: Chest X-ray showing signs of multiple conditions (e.g., cardiomegaly + pleural effusion).
Output: Model correctly assigns multiple labels with balanced precision and recall across classes.
Test Case Derivation: Based on updated SRS scope (multi-label classification support).
How the test will be performed:

1. Use an ambiguous image with verified multiple diagnoses.
2. Run the classifier and extract all predicted labels.

3. Calculate precision, recall, and F1 for each label.
4. Ensure metrics meet performance thresholds $\geq ACCURACY\_THRESHOLD$.

4.1.5 FR5

FRTC6
Title: Display of Predictions and Heatmaps
Control: Manual
Initial State: System has completed classification for an uploaded image.
Input: A classified chest X-ray image.
Output: The interface displays disease labels, confidence scores, and heatmaps clearly within MAX_RENDER_TIME.
Test Case Derivation: Ensures readability and timeliness of visual outputs, matching fit criterion for interpretability.
How the test will be performed:

1. Upload an image and wait for classification to complete.
2. Observe the rendered output on the results screen.
3. Confirm labels, scores, and heatmaps are visible and rendered within MAX_RENDER_TIME.
4. Manually assess layout and readability.

4.1.6 FR6

FRTC7
Title: Diagnostic Report Download
Control: Manual
Initial State: A user is logged in and has a classified image result.
Input: User clicks "Download Report".
Output: Report file downloads with prediction, heatmap, and findings.
Test Case Derivation: Confirms output includes required contents in a retrievable format, supporting traceability.
How the test will be performed:

1. Log in and upload an image.
2. Wait for predictions and visuals.
3. Click "Download Report".
4. Open file and verify it contains predictions, heatmap, and text summary.

4.1.7 FR6

FRTC8
Title: Report Storage Security and Retrieval
Control: Manual
Initial State: A previous diagnostic session has been completed.
Input: User accesses dashboard and attempts to retrieve prior report.
Output: Report is retrieved from encrypted storage (AES-256) and associated with the session.
Test Case Derivation: Validates backend compliance with encryption and user-based indexing for secure access.
How the test will be performed:

1. Upload an image, generate a report, and log out.
2. Log back in using the same account/session.
3. Navigate to dashboard and access stored report.
4. Verify successful decryption and session-specific access.

4.1.8 FR7

FRTC9
Title: Authentication with Valid and Invalid Credentials
Control: Manual
Initial State: Login screen is presented.
Input: Correct and incorrect username/password combinations.
Output: Access granted for valid credentials; access denied and logged for invalid attempts.
Test Case Derivation: Ensures only authorized users gain entry and failed logins are audited.
How the test will be performed:

1. Enter valid credentials and confirm access.
2. Log out and enter incorrect credentials.
3. Confirm access is denied.
4. Check backend logs for authentication failure entry.

4.1.9 FR7

FRTC10
Title: Role-Based Access to Restricted Features
Control: Manual
Initial State: User is logged in.
Input: User with limited permissions attempts to access restricted modules.

Output: Features are hidden or blocked based on user role.
Test Case Derivation: Verifies that sensitive functions are only accessible by users with appropriate permissions.
How the test will be performed:

1. Log in with a basic user role.
2. Attempt to access admin-only dashboard or reports.
3. Confirm UI hides or disables access.
4. Attempt direct URL access and verify rejection.

4.1.10 FR8

FRTC11
Title: Secure API Image Classification Request
Control: Automatic (via script or client request)
Initial State: API endpoint is live and accepting requests.
Input: A valid POST request with a DICOM image and valid token.
Output: Returns JSON with disease labels and confidence scores.
Test Case Derivation: Confirms API behaves as expected for compliant clients, providing structured output.
How the test will be performed:

1. Send POST request with valid token and image.
2. Capture response payload.
3. Confirm JSON structure includes disease names and score values.
4. Validate response format and token usage.

4.1.11 FR9

FRTC9
Title: Invalid Upload Error Handling
Control: Manual
Initial State: Upload form is visible to the user.
Input: Unsupported or corrupted image file.
Output: System displays clear error message and allows retry.
Test Case Derivation: Ensures graceful error handling and user guidance on failure.
How the test will be performed:

1. Attempt to upload a .txt or corrupted .jpg file.
2. Observe UI for error message.
3. Confirm message explains the issue and retry option is presented.

4.1.12 FR10

FRTC10
Title: Diagnostic Result Timestamp Logging
Control: Manual
Initial State: User has completed multiple uploads.
Input: User opens dashboard to view past results.
Output: Each entry shows image, prediction, probability, timestamp, and links.
Test Case Derivation: Verifies historical tracking and completeness of displayed diagnostic data.
How the test will be performed:

1. Upload multiple images and generate predictions.
2. Open dashboard view.
3. Confirm each result includes timestamp, score, image, and report/heatmap link.

4.2 Tests for Nonfunctional Requirements

This subsection includes test cases that evaluate the system's appearance, usability, interpretability, and performance, as described in the SRS nonfunctional requirements. These tests are essential to ensuring the system is accessible, trustworthy, and usable in real-world settings.

4.2.1 Appearance Requirements

NFRTC1
Title: Verification of Calming Color Scheme
Control: Manual
Initial State: The web application is deployed and accessible from a browser.
Input: A user accesses the live system on a browser under default theme settings.
Output: The UI displays a cool and calming color palette, primarily using white and soft blue tones.
Test Case Derivation: A calming palette enhances usability by improving visual comfort and reducing fatigue, making it easier to interpret outputs.
How the test will be performed:

1. Open the application in a modern browser under standard lighting.
2. Review the color palette used across login, dashboard, and report views.
3. Record the dominant background and highlight colors.
4. Ask general users to rate visual comfort and clarity on a 5-point scale.
5. Pass criteria: At least 80% of users rate visual experience 4 or higher.

NFRTC2
Title: Consistency of Simple and Uncluttered Style
Control: Manual
Initial State: Web app is operational and fully loaded with all components.
Input: Navigation across multiple interface modules (login, dashboard, image viewer, report viewer).
Output: UI layout remains consistent and free of clutter; visual hierarchy is preserved.
Test Case Derivation: Consistency in layout and spacing ensures users can navigate the app without confusion, improving overall usability.
How the test will be performed:

1. Access each core module of the UI as a user.
2. Evaluate font styles, spacing, element alignment, and visual consistency across modules.
3. Perform a walkthrough with 3 representative users.
4. Track time taken to locate a core function (e.g., report viewer).
5. Confirm no modules exhibit layout shifts, overlap, or off-brand UI elements.
6. Success metric: All users navigate without confusion and UI passes visual QA checklist.

4.2.2 Usability and Humanity Requirements

NFRTC3
Title: Ease of Performing Core Functionality Without Guidance
Control: Manual
Initial State: The web application is deployed and accessible via a browser; users are interacting with it for the first time.
Input: Untrained users attempt to upload a chest X-ray and request classification using the tool.
Output: At least 80% of users complete the task within MAX_TASK_DURATION without requiring assistance.
Test Case Derivation: The test confirms that intuitive UI design enables users to perform core functions with minimal learning.
How the test will be performed:

1. Recruit 5-7 first-time users (not involved in design).
2. Instruct them to upload a chest X-ray image and trigger the diagnosis feature without guidance or tutorial.
3. Time the task and record completion success/failure.
4. Observe confusion points or UI obstacles.
5. Validate success if at least 80% complete the task in under MAX_TASK_DURATION unaided.

6. Collect optional feedback via Likert-scale survey (e.g., "The task was easy to complete.").

4.2.3 Usability and Humanity Requirements

NFRTC4
Title: Intuitiveness and Memorability of the Software
Control: Manual
Initial State: A cohort of users has completed 1-2 guided sessions with the software.
Input: Users attempt to navigate and use the system independently after a one-month break.
Output: Users are able to recall navigation and perform core actions (e.g., upload, view report) with no external aid.
Test Case Derivation: The output reflects long-term usability, as users who remember workflows are less likely to rely on retraining or support.
How the test will be performed:

1. Provide 1-2 structured walkthroughs of the app to a test group.
2. Wait 4 weeks without further exposure.
3. Ask users to independently upload an image and access the report dashboard.
4. Observe interactions and task success.
5. Survey user experience, focusing on recall of navigation flow and interface elements.
6. Pass criteria: At least 80% of participants complete all steps correctly with minimal hesitation.

NFRTC5
Title: Learning Curve Assessment
Control: Manual
Initial State: System is deployed with a functional user interface, and no help prompts are enabled.
Input: First-time users try to perform a full diagnosis cycle (upload, classify, view result).
Output: Users complete the cycle independently within MAX_TASK_DURATION, with high confidence.
Test Case Derivation: Successful completion without help demonstrates that minimal training is required to use the system effectively.
How the test will be performed:

1. Introduce new users to the platform without providing any tutorials or hints.
2. Ask them to diagnose a chest X-ray image.
3. Record time taken and whether they required external help.

4. Collect post-task confidence ratings (e.g., "How confident were you in your actions?" on a 5-point scale).
5. Define success if majority complete within MAX_TASK_DURATION and rate confidence ≥ 4.

NFRTC6
Title: Realism of Generated Chest X-Ray Images
Control: Manual (with user evaluation)
Initial State: The diffusion model is operational and capable of generating synthetic chest X-ray images.
Input: A set of generated chest X-ray images using the trained model.
Output: Users are unable to consistently distinguish between real and generated images, indicating high visual realism.
Test Case Derivation: Accurate visual mimicry ensures synthetic images serve as valid training or testing inputs, making the system trustworthy for downstream tasks.
How the test will be performed:

1. Generate a dataset of at least 20 synthetic chest X-ray images.
2. Mix with 20 real images from public datasets.
3. Ask 3 or more users to label each image as "Real" or "Generated."
4. Compute accuracy of identifying source.
5. Success if correct classification rate $\leq 60\%$.

4.2.4 Performance Requirements

NFRTC7
Title: Image Generation Speed Test
Control: Automatic
Initial State: The system is running in its production environment.
Input: A request to generate a synthetic chest X-ray image.
Output: The image is rendered in the interface within `LATENCYTHRESHOLD` of request submission.
Test Case Derivation: Quick output ensures user efficiency and supports timely review during clinical workflows.
How the test will be performed:

1. Initiate an image generation request.
2. Log start and end timestamps.
3. Compute total latency.
4. Confirm it does not exceed `LATENCYTHRESHOLD`.

5. Repeat 20 times and average.

NFRTC8
Title: System Stability and Uptime Monitoring
Control: Automatic (Continuous)
Initial State: The system is deployed and uptime tracking enabled.
Input: Continuous automated health checks.
Output: Uptime exceeds `UPTIMETHRESHOLD` with no more than `MAXDOWNTIME` per week.
Test Case Derivation: System must remain online reliably to support consistent user access.
How the test will be performed:

1. Ping the system every 5 minutes for 30 days.
2. Log successes and failures.
3. Calculate uptime and flag excessive downtime.

NFRTC9
Title: Single Image Generation Capacity Enforcement
Control: Manual and Automatic
Initial State: The system is processing an image generation request.
Input: Two image generation requests submitted concurrently.
Output: The second request is rejected with a message: "Only one generation job allowed at a time."
Test Case Derivation: Concurrency limits preserve backend stability and ensure fairness in usage.
How the test will be performed:

1. Submit Request A and wait.
2. Submit Request B before A completes.
3. Verify B is rejected with a clear message.
4. Ensure A completes successfully.

4.2.5 Security Requirements

NFRTC10
Title: Public Accessibility Without Authentication
Control: Manual
Initial State: The web application is deployed publicly.
Input: Access the app from multiple devices/IPs without logging in.
Output: Core features remain accessible without login prompts.
Test Case Derivation: Unauthenticated access supports quick demo/testing and avoids

blocking users behind credentials.
How the test will be performed:

1. Open the app in incognito mode on multiple devices.
2. Try all core features without logging in.
3. Confirm consistent, unrestricted access.

NFRTC11
Title: Verification of Non-Collection of Personal Data
Control: Manual and Automatic
Initial State: Application is live and traffic can be monitored.
Input: User actions across sessions.
Output: No personally identifiable information (PII) is collected or stored.
Test Case Derivation: No collection of PII ensures compliance with privacy regulations and avoids legal risk.
How the test will be performed:

1. Perform user interactions and monitor requests.
2. Check network traffic for PII.
3. Verify logs, local storage, and databases for any user-specific data.

4.2.6 Supportability Requirements

NFRTC12
Title: Availability and Sufficiency of User Documentation
Control: Manual
Initial State: Application includes user-facing documentation.
Input: Users attempt tasks using only documentation.
Output: At least 80% complete core workflows without extra support.
Test Case Derivation: Complete, clear documentation enables users to operate the system independently.
How the test will be performed:

1. Provide only the documentation to test users.
2. Ask them to complete image generation and viewing.
3. Track completion rates and clarity feedback.

4.2.7 Cultural Requirements

NFRTC13
Title: Functionality of Browser-Based Language Translation
Control: Manual

Initial State: App UI is in English and live.
Input: Use browser translation to switch to other languages.
Output: UI renders properly and all features remain functional.
Test Case Derivation: Translation compatibility ensures global usability and avoids UI issues during localization.
How the test will be performed:

1. Open the app in Chrome or Edge.
2. Activate built-in translation and switch languages.
3. Verify UI updates and core functions work.
4. Repeat for at least 3 languages.

4.3 Untested Non-Functional Requirements

Some non-functional requirements are acknowledged but excluded from validation due to current project constraints. These constraints include lack of long-term deployment, restricted access to infrastructure, and project scope boundaries. The following are recognized but untested:

- **NF-SER0 (Scalability or Extensibility Requirements):** Testing for horizontal/vertical scaling, modularity, or plugin compatibility is outside current implementation scope.
- **NF-LR0 (Longevity Requirements):** Validating performance and reliability over a three-year timeline is infeasible within this capstone cycle.
- **NF-MR0 (Maintenance Requirements):** Assessing ease of future updates, bug fixes, and dependency upgrades requires an operational phase post-deployment.
- **NF-EPE0 (Expected Physical Environment):** The software is environment-agnostic and runs entirely within a digital infrastructure. No real-world physical testing required.
- **NF-RIAS0 (Interfacing with Adjacent Systems):** No external system integration exists in the current design. Therefore, interfacing behavior falls outside of scope.
- **NF-IR0 (Integrity Requirements):** The system outputs anonymized visual data only. Since no patient identifiers or modifiable inputs are used, specific data integrity testing was deemed unnecessary.
- **NF-AR0 (Audit Requirements):** While security is enforced via encryption and role-based access, no formal third-party audit or audit trail requirement was defined for this phase.

4.4 Traceability Between Test Cases and Requirements

Functional Requirement	**Test Cases**
FR1: Upload supported chest X-ray images	FRTC1
FR2: Preprocess images (resize, normalize, grayscale)	FRTC2
FR3: CNN classification with confidence scores	FRTC3
FR4: Multi-disease classification per image	FRTC5
FR5: Display predicted labels, confidence scores, and heatmaps	FRTC6
FR6: Downloadable diagnostic report with visuals	FRTC4
FR7: Role-based access control	FRTC7, FRTC8
FR8: Secure API endpoint for classification	FRTC11
FR9: Handle invalid uploads with error messaging	FRTC9
FR10: Dashboard view of predictions with timestamps	FRTC10

Table 2: Traceability Matrix - Functional Requirements to System Test Cases

4.5 Traceability Between Test Cases and Requirements

Table 3: Traceability Between Non-Functional Requirements and Test Cases

Non-Functional Requirement	**Test Cases**
NF-AR1: Calming and professional color scheme	NFRTC1
NF-SR1: Minimalistic and functional design	NFRTC2
NF-SR2: Clean, organized layout	NFRTC2
NF-EUR0: Efficient task completion	NFRTC3
NF-EU2: Intuitive post-training use	NFRTC4
NF-LR0: Learnability and uptime expectations	NFRTC5, NFRTC8
NF-UPR0: Realism of synthetic images	NFRTC6
NF-SLR0: Latency under 60 seconds	NFRTC7
NF-RFTR0: Fault tolerance for corrupted uploads	FRTC9
NF-CR0: One-at-a-time image generation capacity	NFRTC9
NF-CR1: Cultural and language support via translation	NFRTC13
NF-PR0: No personal data collection	NFRTC11
NF-IR0: Access limited to authenticated sessions	FRTC7, NFRTC11
NF-AUR0: Logs of access and configuration	FRTC8
NF-RIAS0: Integration with PACS/EHR	*Untested in this version*
NF-REL0: MVP feature delivery	FRTC1, FRTC3, FRTC4
NF-REL1: Logging and API configuration options	FRTC11
NF-MR0: Modular architecture	*Documented, not tested*
NF-MR1: Git-based version control	*Documented, not tested*
NF-SCR0: Compliance with HL7/FHIR/DICOM	*Reviewed, not tested*
NF-SER0: Scalability and modularity	*Untested in this version*

5 Unit Test Description

This section outlines the approach and rationale for the unit tests designed to verify each module within the chest X-ray diagnostic application. The overall philosophy for test case selection centers on ensuring each module functions correctly in isolation, covering both

standard functionality and potential edge cases. This approach provides a robust foundation to identify and address any issues at the unit level before integrating modules into the broader system.

To streamline testing and avoid redundancy, each module's tests are designed to validate both typical and boundary conditions. This strategy includes creating a baseline test case for normal expected behavior, accompanied by additional edge cases to capture unexpected or extreme inputs. This allows for comprehensive testing without unnecessary detail for every possible scenario.

For efficiency, test cases prioritize high-risk functions and core functionalities. Tests are organized to:

- Validate standard functionality with a single test for typical cases.
- Test critical edge cases that reflect likely and realistic error conditions (e.g., handling of empty data, invalid formats, and unexpected large data inputs).
- Ensure modules perform within expected limits for nonfunctional requirements, such as performance and security, through specialized tests.
- Include module-specific performance tracking across variable dataset conditions to ensure stability and reliability.
- Evaluate modular interactions such as dataset loading, model persistence, and API response validation.

Where possible, detailed input/output descriptions are omitted in favor of referring to well-documented unit testing code, which maintains readable and meaningful test names. This approach not only saves space but also facilitates future maintenance and expansion, as new edge cases can be added to the codebase directly.

5.1 Unit Testing Scope

Unit testing for this project targets the core components of the chest X-ray diagnostic pipeline, ensuring correctness, resilience, and modularity of each individual module. The scope explicitly addresses all critical stages from data ingestion and transformation to model training and persistence, aligned with the architecture described in the updated SRS and design documents.

Each module is tested in isolation under both standard and edge-case conditions. These unit tests are critical for identifying logic flaws, integration risks, and regressions early in the development lifecycle. By verifying core functionalities before system-wide testing, we ensure each module behaves predictably and performs as expected across varied dataset conditions, such as missing files, malformed inputs, and irregular class distributions.

Unit tests span the following core components:

- **DataPreparation Module** - Handles transformation of raw dataset metadata into structured model inputs. Tests verify:

  - Label binarization and class alignment (e.g., `test_binarize_labels`).
  - Stratified data splitting and sample preservation (`test_split_data`).
  - Transformation pipeline integrity for data augmentation and normalization.
- **DataRetrieval Module** - Automates NIH dataset download and extraction. Unit tests confirm:
  - Archive integrity and target directory structure (`test_extract_dataset`).
  - Network download stability and file caching (`test_download_dataset`).
- **ModelArchitecture Module** - Encapsulates CNN layer definitions and model assembly:
  - Validation of model structure and final output layer shape.
  - Confirmation of feature extractor freezing behavior.
  - Forward pass shape and range correctness.
- **Training Module** - Manages training loop, early stopping, and model saving:
  - Training loop correctness and early error detection (`test_train_step`).
  - Prediction output distribution and confidence scoring (`test_prediction`).
- **MLBackend Module** - Provides inference API and model loading:
  - Connectivity test and response latency (`test_connection`).
  - Inference error handling and JSON format validation.
- **ModelInterface Module** - UI-triggered logic for uploads, predictions, and report export:
  - Verification of image format validation and preview rendering.
  - Confirmation that prediction and confidence outputs are rendered correctly.
  - Robustness of report generation and download functionality.
- **AuthClient and Authorization Modules** - Implements frontend and backend login logic:
  - Token creation and validation for protected API routes.
  - Error handling for failed logins and permission mismatches.
- **Config Module** - Backend boot configuration including JWT and CORS:
  - Test security filter chain setup.
  - Validate backend-to-backend communication setup.

Modules not included in this unit testing scope:

- **Third-party libraries or frameworks** (e.g., Flask, Spring Security, PyTorch): These are assumed to be externally validated and are only tested through integration points.
- **Static assets** such as UI CSS or icon libraries, which do not affect logic.

5.2 Tests for Functional Requirements

6 Unit Test Description

6.1 Unit Testing Scope

All internal modules developed by the team are within the scope of unit testing. External libraries (e.g., PyTorch, Albumentations, JWT libraries) are considered out of scope and tested via integration. Priority is given to `ModelInterface`, `DataPreparation`, `MLBackend`, and `Training` due to their centrality in image processing and prediction.

6.2 Tests for Functional Requirements

6.2.1 ModelInterface (M1)

This module is tested via frontend test suites (React + Jest). Tests cover input validation, backend interaction, and PDF generation logic.

1. **MI-test-upload-valid**
Type: Automatic
Initial State: No image uploaded
Input: Valid image file (e.g., .jpg)
Output: Image preview renders in UI
Test Case Derivation: Valid images should trigger preview render for verification.
How test will be performed: Simulate file input and assert image tag is inserted into DOM.

2. **MI-test-analyze-error**
Type: Automatic
Initial State: Image uploaded
Input: API call fails (mock 500 error)
Output: Error message appears to user
Test Case Derivation: Server-side issues should produce user-readable feedback.
How test will be performed: Mock failed request and confirm displayed error.

3. **MI-test-report-download**
Type: Automatic
Initial State: Analysis completed and UI visible
Input: Click on "Download Report"
Output: File download initiated
Test Case Derivation: Download trigger must call blob creation and trigger anchor link.
How test will be performed: Mock browser APIs and confirm link click occurs.

6.2.2 AuthClient (M2)

Frontend module tested using UI simulation and API mocking. Test cases ensure proper error handling and form state behavior.

1. **AC-test-register-valid**
Type: Automatic
Initial State: Registration form shown
Input: Valid username, email, and password
Output: Success message and redirect to login
Test Case Derivation: Valid input should create account and guide user to next step.
How test will be performed: Simulate form fill and submit; assert redirect and toast.

2. **AC-test-login-invalid**
Type: Automatic
Initial State: Login form active
Input: Invalid password for known user
Output: Authentication error shown
Test Case Derivation: Failed login must provide helpful error without crashing.
How test will be performed: Submit credentials and verify error state.

6.2.3 DataRetrieval (M3)

Focuses on verifying NIH image archive download and extraction operations.

1. **DR-test-download-skipped-if-exists**
Type: Automatic
Initial State: Partial image set exists
Input: prepareData() call
Output: Logs skipping already-downloaded files
Test Case Derivation: Should avoid re-downloading existing content.
How test will be performed: Pre-create dummy tar.gz and assert skip path.

2. **DR-test-extraction-failure**
Type: Automatic
Initial State: Corrupt archive present
Input: prepareData() call
Output: Raises FileIOError
Test Case Derivation: Corrupted archives must halt cleanly and raise traceable error.
How test will be performed: Replace archive with dummy data and invoke.

6.2.4 DataPreparation (M4)

This is one of the most testable components. Tests span label parsing, stratified splitting, and augmentation.

1. **DP-test-preprocess-valid**
Type: Automatic

Initial State: Raw CSV loaded
Input: DataFrame with label values
Output: Filtered rows with valid disease strings
Test Case Derivation: Ensures rows with invalid/no labels are removed.
How test will be performed: Feed test CSV and verify rows retained.

2. **DP-test-binarize-labels**
Type: Automatic
Initial State: DataFrame contains pipe-separated label column
Input: e.g., "Pneumonia—Infiltration"
Output: Columns pneumonia=1, infiltration=1
Test Case Derivation: Multi-label parsing must yield one-hot vector.
How test will be performed: Validate output columns after run.

3. **DP-test-stratified-split**
Type: Automatic
Initial State: DataFrame has labels
Input: Call splitData()
Output: 80/20 split with class distribution retained
Test Case Derivation: Split should preserve label proportion across sets.
How test will be performed: Compare $value_counts$ across sets.

4. **DP-test-transform**
Type: Automatic
Initial State: Transform pipeline defined
Input: Image tensor
Output: Transformed tensor of correct shape
Test Case Derivation: Valid transformations must preserve shape and normalization.
How test will be performed: Pass dummy image and inspect result.

5. **DP-test-training-stats**
Type: Automatic
Initial State: Training dataset loaded
Input: binarized label matrix
Output: posWeights and avgPos count
Test Case Derivation: Loss balancing terms must reflect true label sparsity.
How test will be performed: Manually compute and compare.

6.2.5 Authorization (M5)

Covers backend login, token issuance, and registration flows.

1. **AUTH-test-token-generation**
Type: Automatic
Initial State: Valid user credentials
Input: Authenticated session
Output: JWT returned

Test Case Derivation: Login should result in signed token creation.
How test will be performed: Use mock auth and assert token schema.

2. **AUTH-test-invalid-token**
Type: Automatic
Initial State: Token from expired session
Input: validateToken() call
Output: False or exception raised
Test Case Derivation: Expired/malformed tokens should not validate.
How test will be performed: Manually craft bad token.

6.2.6 MLBackend (M6)

Python Flask server responsible for model inference.

1. **MLB-test-healthcheck**
Type: Automatic
Initial State: Backend server running
Input: GET /test
Output: 200 OK + "API connected"
Test Case Derivation: Reachability check verifies backend is online.
How test will be performed: Use test client to make GET call.

2. **MLB-test-valid-prediction**
Type: Automatic
Initial State: Model loaded
Input: Valid chest X-ray image
Output: JSON with predictions
Test Case Derivation: API must return class-wise probabilities.
How test will be performed: Use file upload with sample image and parse output.

3. **MLB-test-missing-file**
Type: Automatic
Initial State: POST /predict triggered without image
Input: No file
Output: HTTP 400 with error
Test Case Derivation: Missing files must be caught and responded to.
How test will be performed: Empty POST request.

6.2.7 ModelArchitecture (M7)

1. **MA-test-model-construction**
Type: Automatic
Initial State: None
Input: Call assembleFullModel(13)
Output: nn.Module with 13 output units

Test Case Derivation: Classification head must output correct dimensions.
How test will be performed: Create model and inspect final layer.

2. **MA-test-layer-freeze**
Type: Automatic
Initial State: Base model instantiated
Input: freezeBaseLayers()
Output: Base layers require_grad = False
Test Case Derivation: Prevent gradient updates in transfer learning.
How test will be performed: Loop through layer flags.

6.2.8 Training (M8)

1. **TRAIN-test-single-batch**
Type: Automatic
Initial State: Model + loader initialized
Input: One training batch
Output: Gradient computed
Test Case Derivation: Confirms training loop runs per-batch loss
How test will be performed: Forward, loss, backward.

2. **TRAIN-test-eval-shape**
Type: Automatic
Initial State: Model in eval mode
Input: Batch of images
Output: Tensor (B, 13) with logits
Test Case Derivation: Output should be interpretable for all labels.
How test will be performed: Dummy input, inspect shape.

3. **TRAIN-test-margin-loss**
Type: Automatic
Initial State: Tensors prepared
Input: logits, labels
Output: Scalar margin loss
Test Case Derivation: Checks added regularization term.
How test will be performed: Call loss function on fixed input.

6.3 Tests for Nonfunctional Requirements

6.3.1 MLBackend (M6)

1. **NF-M6-bulk-prediction-timing**
Type: Automatic
Initial State: ML model loaded with GPU enabled
Input: Batch of 32 chest X-rays
Output: Inference time under 10 seconds

Test Case Derivation: Ensures latency is acceptable for real-time use.
How test will be performed: Log start and end time of batch inference.

2. **NF-M6-no-crash-on-noisy-data**
Type: Automatic
Initial State: Model ready for prediction
Input: Corrupted or low-quality images
Output: Prediction result or warning, no crash
Test Case Derivation: Validates fault-tolerance to suboptimal inputs.
How test will be performed: Feed known noisy inputs and observe log/output.

6.3.2 ModelArchitecture (M7)

1. **NF-M7-heatmap-integrity**
Type: Automatic (Visual and IOU score)
Initial State: Trained model with intermediate layers accessible
Input: Image and expected anatomical region
Output: Heatmap correctly focuses relevant areas
Test Case Derivation: Verifies interpretability of model through Grad-CAM.
How test will be performed: Compare heatmap overlay with expected lung region.

6.3.3 Training (M8)

1. **NF-M8-bias-evaluation**
Type: Automatic
Initial State: Model trained with metadata access
Input: Dataset with demographic splits
Output: No major metric disparity across subgroups
Test Case Derivation: Checks demographic fairness of model behavior.
How test will be performed: Evaluate recall/specificity per subgroup.

6.3.4 Authorization (M5)

1. **NF-M5-invalid-token-access**
Type: Automatic
Initial State: Endpoint secured by JWT middleware
Input: Request with malformed/expired token
Output: 403 Forbidden with log entry
Test Case Derivation: Enforces secure access and auditability.
How test will be performed: Simulate invalid request and verify log/status.

6.3.5 Config (M6)

1. **NF-M6-api-connectivity-test**
Type: Automatic
Initial State: Config boot logic active

Input: Attempt to ping Flask backend
Output: Successful response from ML backend
Test Case Derivation: Ensures connection across stack is reliable.
How test will be performed: Run connectivity ping on init and assert success.

subsectionTraceability Between Test Cases and Modules

Table 4: Legend for Module Labels

Module Label	**Description**
M1	DataPreprocessing - Cleans metadata, binarizes labels, applies transforms, and performs stratified splits.
M2	ChestXrayDataset - Custom PyTorch dataset for loading X-ray images and binary labels.
M3	ModelArchitecture - Builds MobileNetV2-based CNN and classification head for disease prediction.
M4	Training - Manages training loop, loss calculation, and metric evaluation.
M5	ModelPersistence - Handles saving and loading model checkpoints.
M6	DataRetrieval - Downloads and extracts NIH datasets from public URLs.
M7	MLBackend - Flask backend for API communication and image inference.
M8	Config - Spring Boot config logic (security, JWT, REST communication).
M9	Authorization - Auth/login/signup logic for protected user access control.

Table 5: Traceability Matrix for Functional Requirement Test Cases (FRTC)

FRTC	**M1**	**M2**	**M3**	**M4**	**M5**	**M6**	**M7**	**M8**	**M9**
FRTC1	X	X							
FRTC2	X						X		
FRTC3	X	X					X		
FRTC4			X	X			X		
FRTC5			X	X			X		
FRTC6							X		
FRTC7									X
FRTC8									X
FRTC9								X	X
FRTC10								X	X

Table 6: Traceability Matrix for Non-Functional Requirement Test Cases (NFRTC)

NFRTC	M1	M2	M3	M4	M5	M6	M7	M8	M9
NFRTC1								X	
NFRTC2								X	
NFRTC3								X	
NFRTC4								X	
NFRTC5	X	X						X	
NFRTC6			X	X					
NFRTC7							X		
NFRTC8							X		
NFRTC9							X		
NFRTC10								X	
NFRTC11							X		
NFRTC12								X	
NFRTC13								X	

7 Appendix

In this section, additional information that complements the V&V Plan is included.

7.1 Symbolic Parameters

The following symbolic parameters are used throughout the system and unit test cases to avoid hard-coded values. This approach improves clarity, consistency, and maintainability.

Symbolic Name	Definition or Description
`MIN_CLASSIFICATION_ACCURACY`	Minimum per-class accuracy required for CNN-based disease prediction (previously 60%).
`MIN_MULTILABEL_PERFORMANCE`	Minimum precision and recall required for multi-label classification tasks (previously ACCURACY_THRESHOLD).
`DISPLAY_LATENCY_THRESHOLD`	Maximum allowed time to display predictions and heatmaps on the UI (previously MAX_RENDER_TIME).
`MIN_VALIDATION_ACCURACY`	Required accuracy across disease classes on validation datasets (previously 90%).
`MAX_BATCH_PREDICTION_LATENCY`	Maximum duration for batch predictions over multiple X-rays (previously under 10 seconds).
`MAX_USER_DETECTION_ACCURACY`	Maximum rate users can identify synthetic vs. real X-rays (used in realism test, previously 60%).
`USER_SUCCESS_THRESHOLD`	Minimum user success rate on tasks or UI usability favorability (previously at least 80%).

Table 7: Symbolic parameters used in V&V test definitions.

The definition of test cases will call for certain symbolic constants. Their values are defined

in this section for easy maintenance. See the following table for reference.

Symbolic Name	Definition or Description
DATASET-NAME	The chest X-ray imaging datasets used for development and testing. Details are as follows:
MIMIC-CXR	A standard labeled chest X-ray dataset commonly used for medical imaging tasks.
Chest ImaGenome	An extension of MIMIC-CXR, with detailed annotations including 29 bounding boxes on anatomical regions. This dataset links textual report segments to specific bounding boxes, supporting tasks in segmentation and disease localization.

7.2 Usability Survey Questions?

The following survey should be filled out by users after using the system for 5 to 15 minutes. This feedback helps evaluate the user experience and identifies areas for improvement.

7.2.1 User Experience Survey

Time using system: Please provide a rating between 0 and 10 for each of the following categories, with 0 being very difficult or undesirable, and 10 being very easy or desirable.

1. Ease of Use: 0 1 2 3 4 5 6 7 8 9 10
   (0 = very difficult to use, 10 = very easy to use)
2. Navigation: 0 1 2 3 4 5 6 7 8 9 10
   (0 = hard to find what you're looking for, 10 = easy to find what you're looking for)
3. Readability: 0 1 2 3 4 5 6 7 8 9 10
   (0 = hard to understand information, 10 = very easy to understand)
4. Look and Feel: 0 1 2 3 4 5 6 7 8 9 10
   (0 = visually unappealing, 10 = visually appealing)

7.2.2 Additional Feedback:

Notes: (Space for users to leave any additional comments or suggestions about the system.)

Appendix — Reflection

Current Content: Summarizes lessons learned and challenges.
Update Needed: Expand reflections to cover **challenges in classification generalization, optimizing hyperparameters, and improving model transparency for clinical adoption**.

7.3 Question 1

What went well while writing this deliverable?

Harrison Chiu: Planning out the modules for functional testing went pretty smoothly. Reviewing SOLID principles helped a lot and I was able to structure each module with a clear purpose, which made creating the tests feel straightforward. I even tried some initial tests on my own machine to catch edge cases early on, and it gave me confidence that I was covering all the important scenarios. That fuzz testing experiment helped me feel like I was on the right track.

Hamza Issa: Working on identifying tests for functional requirements felt really intuitive. After we reached a consensus regarding the functional requirements and our ability to implement it, it was quite simple to devise tests that were complete and meaningful, as part of creating functional requirements is that the expectation action or output is typically discrete.

Gurnoor Bal: Working on the SRS verification was actually pretty satisfying because I got to make sure everything was crystal clear. I found a few areas in the SRS that needed improvement, and being able to fix those early felt good. Plus, setting up the traceability matrix was helpful since it kept things organized and made it easy to link each requirement with its specific test case. It felt like things were clicking into place.

Jared Paul: I liked working on the design verification part because it let me focus on making sure the system was organized and easy to work with. I emphasized modularity and made sure each module was self-contained, which will make the system easier to maintain down the road. I also created a checklist to make sure each module followed solid design practices, so I felt like I was covering the important aspects without missing anything.

Ahmad Hamadi: Writing the validation plan was rewarding because I got to think about how people would actually use the tool, especially radiologists. I set up task-based testing and feedback sessions, so we'll get a real sense of how practical the tool is. Using actual datasets to check the model's accuracy felt like a solid choice too. It felt like I was creating a plan that would make sure the tool was both accurate and user-friendly.

7.4 Question 2

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

Hamza Issa: I found that in order to create a V&V that was really complete meant that I needed to not just need a basic abstract understanding of the theory around Diffusion

models, rather I needed to actually understand the theory, how it is typically implemented today in research and industry, so much so that I needed to find out the type of data structure to expect as output which would be a rather complex matrix. This entire process of learning was really challenging as I was a novice in the subject. But I managed to solve this by purchasing a course on Udemy regarding Theory and Implementation of Diffusion models which helped me really better understand and write a V&V plan that actually made sense in the context of diffusion models.

Harrison Chiu: A bit of a challenge was figuring out just how detailed each test case needed to be. I kept wondering if I was getting too specific or not specific enough. To fix this, I made a quick checklist of essential edge cases based on the project requirements and ran it by the team. Getting their input reassured me that I was covering the right bases without going overboard.

Gurnoor Bal: The main problem was with some of the requirements that were a bit too general or vague. It was hard to nail down exactly what needed verifying. I decided to check in with our project supervisor to get a better handle on what was expected, and I left a few notes in the document to flag areas we might need to clarify later. That way, the process stayed clear without me having to guess on details.

Jared Paul: One thing that wasn't easy was verifying the interfaces for each module. Some of the connections needed more detail to line up with the rest of the design. To sort this out, I went back and added specific inputs and outputs for each module, which made sure everything matched up well with the Module Guide. It took a bit more time, but in the end, it made the design much clearer.

Ahmad Hamadi: The tricky part was making sure the tests weren't just covering easy scenarios. I wanted to be sure they were meaningful. I ran into a few issues with environment consistency, which caused random test failures. To fix this, I standardized the setup and added a few retries for tests that occasionally flaked. Then I went back to the requirements to make sure the tests were focused on the important functions. This way, the tests ended up being both reliable and relevant.

7.5 Question 3

What knowledge and skills will the team collectively need to acquire to successfully complete the verification and validation of your project? Examples of possible knowledge and skills include dynamic testing knowledge, static testing knowledge, specific tool usage, Valgrind etc. You should look to identify at least one item for each team member.

Several different and diverse skills are required in order to create the verification and validation plan for this Chest X-ray Diffusion Model Research project.

Software Specification Design
Our team will need a strong understanding of software specification design to grasp the project context and outline the required functions and modules. By defining each function's purpose, inputs, and edge cases, we can plan the implementation of functions accurately

according to the specifications. This design phase will serve as the foundation for later development stages.

Unit Test Planning
Developing robust unit tests will be crucial. Our team must create comprehensive tests that ensure each function achieves its intended purpose reliably. This will require us to account for all relevant edge cases to avoid misleading results and ensure the accuracy and completeness of our unit tests as a core skill for our V&V plan.

Software Design Principles
To maintain a flexible and maintainable codebase, we'll need to prioritize software design principles, particularly the SOLID principles. For example, we plan to structure modules to follow the Single Responsibility Principle, which will help ensure clarity and cohesion in the design, allowing future engineers to work with the code more effectively.

Diffusion Model Concepts and Implementation
Understanding both the theoretical and practical aspects of diffusion models will be essential for designing an accurate V&V plan. Our team will need knowledge of the inputs, expected outputs, and common data structures used in similar applications to develop thorough and effective tests.

7.6 Question 4

For each of the knowledge areas and skills identified in the previous question, what are at least two approaches to acquiring the knowledge or mastering the skill? Of the identified approaches, which will each team member pursue, and why did they make this choice?

1. **Software Specification Design**
   Revise existing papers on software specification design, for example material on MIS specifications, and work done by David Parnas Reflect on previous Mcmaster Software Engineering course on software design (i.e. 3A04)
2. **Unit Test Planning**
   Revise testing blogs produced by large tech companies such as Meta, Google and Amazon. These blogs devise the strategies these companies adopt in their creation of effective unit tests. Reflect on previous Mcmaster Software Engineering course, 3S03 Software Testing
3. **Software Design Principles**
   Revise SOLID principles via informational blogs such as DigitalOcean, Medium, and FreecodeCamp Implement personal projects that are small but correctly implement each of the design principles.
4. **Diffusion Model Concepts/Implementation**
   Utilize Introduction to Diffusion Models course on Udemy. This goes over theory and implementation in practice today. Read existing papers on experiments with diffusion models by other researchers that have similar applications

Now, each team member selected a particular method to pursue, and stated their justifications below:

Jared Paul: I'm going to revise more formal and professional software specification design practices by looking at existing papers, such as that from David Parnas. This is because I recall being introduced to him in 2nd year, and found the content really straightforward and informative the, so I thought it would be able to get me up to speed quite quickly today.

Hamza Issa: I chose to revise existing engineering blogs from big tech companies like Amazon, as although there are papers and lots of theory out there for devising unit tests. I think that in designing and creating the best unit tests we can look to companies that are highly dependent on them, to find tests that are not just complete but not overly verbose.

Ahmad Hamadi: I plan on revising my SOLID principles using popular blogs like I've seen from FreeCode camp. I think this would be the simplest way to learn all the principles in a manner that is easy to digest and also easy to implement myself for when I want to experiment and practice.

Harrison Chiu: I am going to try and complete the Udemy course on Introduction to Diffusion models. I have some experience looking at research papers for Diffusion models, but they have been really difficult to understand due to the advanced mathematics. I think using a Udemy course would make it simpler to learn and easily transferable to our project

Gurnoor Bal: I am going to investigate existing research papers in the Diffusion Models space that have some similarity to our application. I think this would not only help me learn the required concepts, but also show me how to actually produce research and an application catered for this specific project. If i were to just watch the Udemy course, I'll understand the concepts, but I would still need to review papers to understand how to go about it for this particular project.