System Verification and Validation Plan for Software Engineering

Team #1, Sanskrit Ciphers
Omar El Aref
Dylan Garner
Muhammad Umar Khan
Aswin Kuganesan
Yousef Shahin

October 27, 2025

Revision History

Date	Version	Notes
Date October 27	1.0	Adding NFR testing plan
Date October 27	1.1	Adding FR testing plan
Date October 27	1.2	Added section 2 and reflection
Date October 27	1.3	Added section 3.1 - 3.4
Date October 27	1.4	Added implementation, automated, and software verification

Contents

1 Symbols, Abbreviations, and Acronyms . . . iv

2 General Information . . . 1
- 2.1 Summary . . . 1
- 2.2 Objectives . . . 1
- 2.3 Challenge Level and Extras . . . 3
- 2.4 Relevant Documentation . . . 4

3 Plan . . . 5
- 3.1 Verification and Validation Team . . . 5
- 3.2 SRS Verification . . . 6
- 3.3 Design Verification . . . 8
- 3.4 Verification and Validation Plan Verification . . . 12
- 3.5 Implementation Verification . . . 13
- 3.6 Automated Testing and Verification Tools . . . 14
- 3.7 Software Validation . . . 14

4 System Tests . . . 15
- 4.1 Tests for Functional Requirements . . . 15
  - 4.1.1 Area of Testing 1 — User Authentication and Access . . . 15
  - 4.1.2 Area of Testing 2 — Fragment Upload and Orientation Normalization . . . 16
  - 4.1.3 Area of Testing 3 — Fragment Matching and Reconstruction . . . 18
  - 4.1.4 Area of Testing 4 — Script Identification and Transcription Assistance . . . 19
  - 4.1.5 Area of Testing 5 — Database Search and Exploration 20
- 4.2 Tests for Nonfunctional Requirements . . . 21
  - 4.2.1 System Performance Testing . . . 21
  - 4.2.2 AI Matching Assistance Quality . . . 23
  - 4.2.3 Usability Testing . . . 24
  - 4.2.4 System Availability and Reliability . . . 26
  - 4.2.5 Data Integrity and Security . . . 28
  - 4.2.6 Scalability Testing . . . 30
  - 4.2.7 Cross-Browser and Device Compatibility . . . 31
  - 4.2.8 Performance Benchmarking . . . 33

4.3 Traceability Between Test Cases and Requirements 34

5 Unit Test Description 35

5.1 Unit Testing Scope 36

5.2 Tests for Functional Requirements 36

5.2.1 Module 1 36

5.2.2 Module 2 37

5.3 Tests for Nonfunctional Requirements 37

5.3.1 Module ? 37

5.3.2 Module ? 38

5.4 Traceability Between Test Cases and Modules 38

6 Appendix 40

6.1 Symbolic Parameters 40

6.2 Usability Survey Questions? 40

List of Tables

1 V&V Roles and Responsibilities 5

2 Traceability for Functional Requirements 34

3 Traceability for Non-Functional Requirements 35

[Remove this section if it isn't needed —SS]

List of Figures

[Remove this section if it isn't needed —SS]

1 Symbols, Abbreviations, and Acronyms

symbol	description
T	Test

[symbols, abbreviations, or acronyms — you can simply reference the SRS (Author, 2019) tables, if appropriate —SS]

[Remove this section if it isn't needed —SS]

This document ... [provide an introductory blurb and roadmap of the Verification and Validation plan —SS]

2 General Information

2.1 Summary

The software system being tested is **Sanskrit Manuscript Reconstruction Platform**, a web-based research tool designed to assist scholars in reconstructing fragmented Buddhist manuscripts through computational analysis. The system integrates machine learning, computer vision, and database management to enable users to upload digitized manuscript fragments, perform automated preprocessing and orientation correction, and generate similarity based match suggestions across fragments.

The platform consists of a front-end interface for visualization and user interaction, a back-end API for data management and orchestration, and multiple AI/ML components responsible for text extraction, edge and damage pattern analysis, and handwriting classification.

The verification and validation process will ensure that all core functionalities, including fragment upload, orientation correction, transcription assistance, and match discovery, perform correctly, reliably, and consistently across diverse manuscript inputs. It will also verify that data integrity, user authentication, and safety-critical requirements identified in the Hazard Analysis are met.

Testing will be conducted incrementally across all major milestones (Proof of Concept, Revision 0, and Revision 1) using a combination of unit testing, integration testing, usability testing, and system-level validation to confirm that the software meets its specified functional and non-functional requirements as outlined in the SRS.

2.2 Objectives

The primary objective of this Verification and Validation (V&V) Plan is to ensure that the **Sanskrit Manuscript Reconstruction Platform** meets its defined functional and non-functional requirements, operates reliably under typical research conditions, and achieves sufficient usability for its intended audience of Buddhist Studies scholars. The V&V process aims to build confidence in the correctness, stability, and scholarly utility of the system.

In-Scope Objectives:

- **Functional correctness:** Verify that all major workflows (fragment upload, image normalization, fragment matching, OCR transcription, and session management) produce accurate and expected outcomes.
- **Reliability:** Validate that the system performs consistently across multiple sessions and input types, with appropriate handling of errors, interruptions, and data corruption.
- **Data integrity and security:** Ensure that user data, manuscript fragments, and annotations are stored and transmitted securely, as outlined in the Safety and Security Requirements.
- **Usability and accessibility:** Evaluate that scholars can intuitively use the system to complete core reconstruction tasks with minimal training, supported by the usability study and user manual deliverables.
- **Model transparency:** Confirm that AI-generated suggestions (match rankings, OCR outputs, classifications) include interpretable confidence levels and require explicit human confirmation before acceptance.
- **Traceability:** Establish end-to-end verification between requirements, test cases, and hazard mitigations to ensure all critical safety-related functionalities are validated.

Out-of-Scope Objectives:

- **External library verification:** The correctness of third-party libraries and frameworks (e.g., OpenCV, PyTorch, TensorFlow) will be assumed based on their established reliability and community validation.
- **Extensive performance benchmarking:** While basic response-time and load-handling tests will be conducted, large-scale benchmarking under production conditions is beyond the project's resource and time constraints.

- **Cross-platform hardware testing:** The V&V plan focuses on web deployment via modern browsers and does not include testing on specialized hardware or legacy systems.

- **Formal verification:** Mathematical proofs or model checking of algorithms are not feasible within the capstone timeline and are deferred to potential future work.

The objectives and their limitations reflect the practical constraints imposed on the team (mainly time and resources) while ensuring comprehensive coverage of functionality, usability, and safety-critical elements central to the platform's success.

2.3 Challenge Level and Extras

This year's capstone projects do not include a challenge level. Evaluation will focus on system completeness, correctness, usability, and overall technical quality.

Extras:

Two approved extras are included as part of this project to enhance the evaluation of usability and maintainability.

- **User Manual:** A comprehensive user manual will be developed to assist scholars in using the platform effectively. It will describe the main workflows such as fragment upload, preprocessing, matching, and transcription, and provide troubleshooting guidance. The manual will also include annotated screenshots and short tutorials covering both basic and advanced usage scenarios.

- **Usability Report:** A formal usability evaluation will be conducted to assess the system's ease of use, learnability, and effectiveness for non-technical users. Feedback will be collected from scholars in Buddhist Studies and related fields through observation and structured task completion. The resulting report will summarize key usability metrics, identify design pain points, and provide recommendations for improvement.

These extras directly support the project's primary goals of accessibility and scholarly usability by ensuring that end users can efficiently interact with complex AI-driven tools through a clear, guided interface.

2.4 Relevant Documentation

This Verification and Validation Plan references all major project documents that define the system's motivation, requirements, development strategy, safety considerations, and user-facing deliverables for the **Sanskrit Manuscript Reconstruction Platform**. These documents collectively establish the foundation for testing and traceability throughout the project lifecycle.

- **Problem Statement and Goals Document** (Ciphers (2025d)) Defines the project motivation, objectives, and expected outcomes for reconstructing fragmented Buddhist manuscripts.
- **Development Plan** (Ciphers (2025b)) Outlines the workflow structure, team responsibilities, scheduling, coding standards, and continuous integration process.
- **Software Requirements Specification (SRS)** (Ciphers (2025e)) Specifies all functional and non-functional requirements, including detailed system components, interfaces, and prioritized behaviors.
- **Hazard Analysis Document** (Ciphers (2025c)) Identifies potential hazards, risk factors, and corresponding safety and security requirements that inform this V&V plan.
- **Design Documents (MG and MIS)** (Ciphers (2025a)) To be developed in later phases. These will provide architectural decomposition, module definitions, and design rationale.
- **User Manual** (Ciphers (2025g)) Provides detailed guidance for scholars on using the platform's core features, including uploading, preprocessing, matching, and transcription.
- **Usability Report** (Ciphers (2025f)) Summarizes results from usability testing and participant feedback to evaluate ease of use, learnability, and overall effectiveness.

Together, these documents establish a complete record of the project's development and verification process, ensuring traceability between requirements, design, testing, and evaluation.

3 Plan

This section presents the overall strategy for Verification and Validation (V&V) of the project. It outlines how we will organize people and activities, how key artifacts will be reviewed and tested, and how results will inform subsequent work. The goal is to provide a clear, practical roadmap for building confidence in the system from requirements through delivery.

3.1 Verification and Validation Team

This subsection defines the project roles responsible for planning, executing, and reviewing V&V activities. Each role has clear primary responsibilities and an assigned independent Reviewer to ensure separation of duties and objective appraisal. Roles may rotate by phase (requirements, design, implementation) to match expertise and workload, but accountability remains with the named owner for the current milestone. The table below summarizes the assignments.

Table 1: V&V Roles and Responsibilities

Role	**Primary Responsibilities**	**Person**	**Reviewer**
V&V Lead	Manages the development and execution of the V&V plan, oversees test protocols, ensures compliance, and leads root cause analysis for issues found during testing	Dylan Garner	Aswin Kuganesan
V&V Manager	Defines the overall V&V strategy, coordinates activities across the project, manages schedules and resources, and forecasts workloads	Yousef Shahin	Umar Khan
Test Architect	Builds Req↔Test traceability; drafts system and NFR tests	Omar El-Aref	Yousef Shahin
V&V Engineer	Develops and executes test plans and procedures, designs test cases, documents results, and troubleshoots issues.	Umar Khan	Dylan Garner
Quality Assurance	Often works closely with the V&V team to ensure the product meets quality standards throughout the development lifecycle	Aswin Kuganesan	Omar El-Aref
Supervisor	External reviewer; approves SRS baselining and risk mitigations	Dr. Shayne	

3.2 SRS Verification

This subsection defines how we will verify that the SRS is correct, complete, consistent, measurable, and feasible. We use three complementary approaches: guided reviews by TAs/peers, a supervisor-facing survey and interview that validate fundamentals in non-technical terms (translated into technical requirements), and continuous conformance checks during design/implementation against SRS use cases and fit criteria. Each approach includes instruments, process, and success metrics.

TA/Peer Guided Review (Structured Feedback)

Instrument. A checklist-driven review by peers and TAs to ensure we are staying in line with the fundemntals of our requirements. Issues are filed in GitHub as feedback from TAs and peers as well as feedback on Avenue which is more detailed from the TA
Process.

1. After handing in the SRS document our peers go through the document and highlight areas to improve on.
2. Once our peers have given us feedback we are to address the feedback
3. The TA will also go through the SRS document and provide more in depth feedback to help with the document overall
4. Once we get the feedback for the TA we create an Issue in GitHub and address the feedback given to us

Success metrics.

- **Measurability coverage:** $\geq$ 100% of feedback has been addressed
- **Ambiguity count:** $\leq$ 2 open ambiguity items after rework.
- **Defect turn-around time (TAT):** All feedback has been addressed within 72 hours of being given the feedback (with a grace period of 48 hours for the entire team)
- **Link/xref errors:** 0 build warnings for missing references.

Supervisor Validation via Survey plus Interview (Non-technical → Technical)

Instrument. A 10-question survey focused on *fundamental product expectations*, followed by a 30 min interview. Example survey items:

1. "The SRS describes how fragments will be *prepared* (orientation/cleaning) before analysis." (1–5)
2. "The SRS explains how *candidate matches* will be presented for scholarly judgment." (1–5)
3. "The SRS addresses *trust signals* (e.g., confidence scores, overlays) for suggested matches." (1–5)
4. "Privacy and access for *unpublished images* are clearly described." (1–5)
5. Open: "What essential research task is missing or under-specified?"

Process.

1. Send SRS and survey 48 hours in advance; collect responses.
2. Hold interview; capture clarifications and priorities (top 3 must-haves).
3. *Translation step:* Requirements Analyst converts each non-technical concern into one or more technical requirements (G/S with Fit Criteria), files issues, updates traceability.

Success metrics.

- **Coverage score:** average score ≥ 4.0 on survey.
- **Gap closure:** 100% of supervisor "missing/under-specified" items mapped to new/updated requirements within 1 week.
- **Traceability updates:** new items appear in Req→Test matrix with an assigned test type and owner.

Ongoing Conformance During Build (Design/Code → SRS)

Instrument. A checklist that is shared with the team and has each module mapped to a requirement.
Process.

1. For each S.4 scenario in the SRS, create at least one system test; link the implementing issue/Pull Request (PR).

2. On each PR, the author selects related requirement from the SRS document.

3. Monthly conformance audit: Each month the team will highlight what percentage of the requirements is done or being completed.

4. The team will also hihglight each month if they encountered a requirement that the SRS missed and make sure it is added within 48 hours of finding the missing requirement.

5. If code conflicts with SRS, open change request: either update SRS (with rationale) or refactor code.

Success metrics.

- **Traceability coverage:** $\geq$ 95% of modules or requirments can be linked to code module.

- **Use-case alignment:** 100% of G.5 use cases have a corresponding S.4 detailed scenario and at least one test case.

3.3 Design Verification

This subsection defines how we will verify that the *design* (architecture, detailed design, and model design) is **sound, implementable, and consistent** with the SRS. We use three complementary techniques with objective success criteria.

DV-1: Unified Modeling Language (UML) Design $\leftrightarrow$ Code Modules

Approach. Maintain a living UML class diagram for core modules (front-end, backend services, Machine Learning (ML) components). Verify code conformance on each milestone via static review (e.g., presence of classes/interfaces, key method signatures, dependency directions).

- **Procedure:**
    1. Identify *must-exist* types and interfaces in the UML for the current milestone.
    2. Map each UML element to its code module (file/path, namespace, class/interface name).
    3. Keep a checklist of the modules that have been mapped and the ones that still need too get created.
    4. As the design starts getting implemented we can start adjusting the UML diagram as needed.
- **Success metrics:**
    - *Coverage:* $\geq$ 95% of *must-exist* UML elements mapped to code modules.
    - *Directional integrity:* 0 violations of designated dependency rules.

DV-2: Structured Walkthroughs (Code/Design)

We will use lightweight, structured walkthroughs for code and design implementations

Author

- **What they do:** Presents the change (PR), states intent, scope, and known risks; answers questions.
- **How it works:** In a meeting they will share their screen and present the changes they made as well as the logic behind their changes
- **Why it rotates:** The author is simply whoever made the change for that task; authorship necessarily varies by feature or document section.

Reviewer

- **What they do:** Performs the detailed critique using a checklist (correctness, testability, traceability, compliance).
- **How it works:** Reviews asynchronously before the session; in the session, walks through findings; files issues or suggests concrete edits.
- **Why it rotates:** Review expertise should match the change (e.g., ML for matching model code, accessibility for User Interface (UI) text), so the most relevant teammate reviews.

Walkthrough Lead

- **What they do:** Facilitates the session, keeps time, ensures the checklist is followed, resolves scope creep, and states clear outcomes.
- **How it works:** Opens with goals and exit criteria; keeps discussion to evidence; confirms owner and due date for each action.
- **Why it rotates:** Lead alternates between V&V Lead and Test Architect depending on topic (documents vs. tests/code), ensuring domain-appropriate facilitation.

Audience / Notetaker

- **What they do:** Attends for situational awareness; the designated notetaker records decisions, action items, and deferments.
- **How it works:** Notes are captured and action items become GitHub issues (`blocker/major/minor`) with assignees and deadlines.
- **Why it rotates:** Anyone can take notes; rotating spreads context and avoids overloading a single person.

Process (applies to both code and document walkthroughs).

1. **Prep (T − 24h):** Author posts link(s) to PR , a 1–2 paragraph summary, and relevant checklists.
2. **Asynchronous review:** Reviewer annotates items and runs applicable checks.

3. **Session (30 minutes):** Lead states goals & exit criteria; Author presents; Reviewer walks through findings; decisions and actions are recorded.

4. **Exit criteria:** All *blocker/major* items assigned with due dates; PR updated or issues filed; next steps acknowledged by owners.

5. **Follow-up (72h):** Author closes items; Reviewer verifies fixes; Lead updates the review log.

Why roles change per task. Changes originate in different parts of the system (UI, backend, ML). The *author* is the contributor for that change; the *reviewer* is selected for the best subject-matter fit; the *lead* alternates to match topic; the *notetaker* rotates to share context. This rotation ensures expertise, fairness in workload, and higher defect detection while remaining lightweight and feasible.

DV-3: ML Design Soundness (Regularization/Robustness)

Approach. Verify that AI/ML components are designed to generalize: regularization and robustness are evaluated with stress tests.

- **Controls implemented:** dropout (or equivalent), data augmentation, early stopping, and (where applicable) ensembles.

- **Ablation plan:** compare baseline vs. dropout, augmentation, ensemble; report mean±sd over k-fold Cross Validation.

- **Robustness:** evaluate on mildly perturbed inputs (rotation, illumination, small occlusion); report metric deltas.

- **Success metrics:**

  - Will mainly be measured by if we found a better more robust model. Regardless of the outcome it will be a success since we have tested the models ability to generalize.

- **Design outcome:** if criteria fail, document change in an ADR (e.g., add augmentation X, adjust dropout p, switch loss) and retest.

3.4 Verification and Validation Plan Verification

The V&V plan is itself an artifact that must be verified and improved over time. We will use peer/TA reviews, a structured test-plan walkthrough & retrospective focused on functional and nonfunctional tests

PV-1: Classmate and TA Reviews

What: External review by classmates and a course TA to assess clarity, completeness, and feasibility.
How:

- Have a completed version of the V&V plan submitted.
- Class peers review the plan and give feedback in an Issue on the GitHub.
- TA reviews the plan and confirms scope, grading-aligned expectations, and missing evidence.
- Each team memeber will create an Issue on the GitHub and fix the comments they were given on their respective parts.

Success: $\geq$ 95% of actions from peer/TA reviews closed within 7 days.

PV-2: Test-Plan Walkthrough & Retrospective (Functional & Nonfunctional)

What: A structured meeting to walk through current system tests (functional) and NFR tests (accessibility, privacy, performance, usability), emphasizing what has worked, what has not, and how to adapt. This will be quite similar to the walkthrough mentioned ion susection 3.1.
Inputs: Current test catalog, pass/fail history, known pain points, and a walkthrough checklist covering: specificity of steps/data, pass/fail criteria, traceability to SRS Req IDs, reusability, and stakeholder relevance.
How:

1. Lead presents the V&V Plan document and walks through the sections asking if the team has any doubts with the currecnt section.
2. If a team member does have doubts about a section they would then specify. (If not the Lead moves on to the next section).

3. The team comes up with a solution to address the doubt and clear up any misunderstadning.
4. Team records adjustments in an Issue on the GitHub with owner and due date.

Success:

- Executability: 100% of sampled tests can be executed by a third party without clarification after fixes.
- Specificity: Each test specifies input data, exact steps, expected outputs, and pass thresholds.
- Adaptation throughput: $\geq$ 95% of improvement actions closed by next walkthrough.

3.5 Implementation Verification

Implementation verification will ensure that all parts of the system follow the functional requirements described in the SRS. Verification testing will include a variety of integration tests and static verification tests.

Each module will have tests associated with it to verify correctness of input/output behavior, error handling, and data integrity. These tests will be defined in the testing plan and maintained in the continuous integration pipeline.

Static Verification:

- **Code walkthroughs and inspections:** Conducted by peers to ensure code quality, maintainability, and adherence to design principles.
- **Static analysis:** Automated tools will identify potential memory leaks, unused variables, and security vulnerabilities prior to runtime testing.
- **Code review sessions:** Implemented using GitHub pull requests to ensure that every commit has at least one peer review before being merged to the main branch.

3.6 Automated Testing and Verification Tools

Automated verification will be supported through system testing and quality assurance tools integrated into the project's continuous integration pipeline.

Tools and Test Frameworks:

- **System Testing:** For back-end, testing frameworks such as pytest will be used, and Jest will be used for front-end components.
- **Continuous Integration:** GitHub Actions for automated build, test, and deployment verification.
- **Static Analysis:** Coding standards will be enforced using static code analysis tools such as pylint, and static type checkers such as mypy will be used for type checking.
- **Test Coverage:** To measure the percentage of lines covered by automated tests, coverage.py can be used for the back-end and Jest will be used for the front-end.
- **Dependency and Security Scanning:** For the back-end, security scanners such as bandit will be used and npm audit will be used for frontend libraries to identify known vulnerabilities.

Test coverage metrics will be summarized in the continuous integration dashboard and reviewed regularly to ensure that important components such as data management and authentication satify quality metrics. Linting and static checks will be automatically run when the pull request is created.

3.7 Software Validation

Validation ensures that the implemented system meets the supervisor's requirements and satisfies its intended purpose in the academic research context. Validation will focus on confirming that reconstructed manuscripts and system functionality align with requirements.

Validation Activities:

- **Supervisor Review Sessions:** Periodic demonstrations to the project supervisor and relevant stakeholders in the field to confirm that features meet research and usability expectations.

- **Usability Testing:** Stakeholders will try reconstructing manuscripts to observe system functionality and performance.
- **Requirements Comparison Review:** Comparing system behavior with the functional and non-functional requirements in the SRS to confirm all requirements are satisfied.
- **Manuscript Validation:** Comparison of fragment matches generated by the system with manuscript pairings that are proven to be accurate from the same dataset.

The Rev 0 demonstration will be used to validate the requirements through feedback provided from supervisors and peers to identify software usability issues and determine final improvements prior to the project's completion.

4 System Tests

This section describes the system-level tests for both functional and non-functional requirements. Functional requirement tests verify that the system implements the specified features correctly. Nonfunctional requirement tests validate quality attributes including performance, usability, reliability, security, scalability, and compatibility. Each test case specifies the test type, initial state, input conditions, expected output, and detailed procedures for execution.

4.1 Tests for Functional Requirements

This section details the functional tests derived from the system's high-priority functional requirements defined in sections S.2 and S.4 of the Requirements Standard Plan. Each test validates that the platform performs its intended scholarly and technical functions.

4.1.1 Area of Testing 1 — User Authentication and Access

This set of tests verifies that the authentication system satisfies requirements S.4.1 from the SRS, ensuring secure login, user access control, and proper error handling. These tests confirm compliance with input validation constraints defined in the data constraints table of the SRS.

1. **Test ID:** test-fr-auth-1
   Control: Automatic
   Initial State: System running; no user session active.
   Input: Valid username and password.
   Expected Output: User successfully authenticated and directed to dashboard.
   Test Case Derivation: Confirms secure credential verification using backend authentication service.
   How test will be performed: Automated UI test verifying successful login and redirection.
   Success criteria: User receives a valid session token and is redirected to the dashboard within 2 s; authentication response is HTTP 200 and no error message is shown.
   Requirement addressed: The system shall authenticate users through a secure login interface and only allow authorized access.

2. **Test ID:** test-fr-auth-2
   Control: Automatic
   Input: Invalid username and/or password.
   Expected Output: Access denied; error message displayed "Invalid username or password."
   How test will be performed: Automated UI test submitting incorrect credentials.
   Success criteria: Login attempts with invalid credentials return HTTP 401/403 and display the correct error message; no session is created.
   Requirement addressed: Invalid credentials must be rejected.

4.1.2 Area of Testing 2 — Fragment Upload and Orientation Normalization

These tests ensure that the system fulfills requirements S.4.2 of the SRS, covering input handling for image uploads, file format constraints, and automated orientation correction. Reference is made to SRS Table 3 (Data Constraints) for permissible image formats and size limits.

1. **Test ID:** test-fr-upload-1
   Control: Automatic

Initial State: Authenticated session.
Input: Valid image file (less than 50 MB).
Expected Output: File successfully uploaded and preprocessing begins.
Test Case Derivation: Confirms compliance with supported file formats and upload limits.
How test will be performed: Automated API and UI tests with sample image files.
Success criteria: Server accepts the file (HTTP 200/201), file is stored in the expected location, and the preprocessing pipeline is invoked within 5 s of upload.
Requirement addressed: The system shall allow users to upload images (JPG, PNG, TIFF) for processing.

2. **Test ID:** test-fr-upload-2
   Control: Automatic
   Initial State: Uploaded image with random rotation.
   Input: Rotated manuscript image.
   Expected Output: Processed image aligned horizontally; confidence score
   *geq*0.9.
   How test will be performed: Compare processed image orientation using OpenCV-based angle detection.
   Success criteria: The corrected image orientation is within
   *pm*5
   deg of horizontal and the orientation confidence score is
   *geq*0.9.
   Requirement addressed: The system shall automatically regularize image orientation within
   *pm*5
   deg.

4.1.3 Area of Testing 3 — Fragment Matching and Reconstruction

This section addresses requirements S.4.3 of the SRS, ensuring that the system can identify and reconstruct fragment matches based on edge and content similarity. The tests validate AI model performance, ranking algorithms, and interactive assembly functionality.

1. **Test ID:** test-fr-fragment-matching-1
   Control: Automatic
   Initial State: Database populated with preprocessed fragments.
   Input: Selected fragment ID.
   Expected Output: Ranked list of at least three probable matches with similarity scores.
   Test Case Derivation: Validates integration between AI/ML cluster and backend orchestration service.
   How test will be performed: Automated backend tests with known fragment pairs; check if top match $>$ 0.8 accuracy.
   Success criteria: At least three candidate matches are returned, the top-ranked match has similarity score $>$ 0.8, and results are returned within the expected latency (e.g., 5 s).
   Requirement addressed: The system shall generate ranked match suggestions for fragments based on edge, damage, and content similarity.

2. **Test ID:** test-fr-fragment-matching-2
   Control: Manual
   Initial State: Logged-in user with fragments loaded.
   Input: Drag-and-drop fragments onto canvas.
   Expected Output: Fragments move and snap to nearby edges; arrangement saved to session.
   How test will be performed: Manual verification during demonstration; supplemented with UI automation for drag-drop response $<$ 200 ms.
   Success criteria: Fragments snap correctly to compatible edges, the arrangement is persisted to the session store and can be reloaded, and UI drag/drop feedback latency is under 200 ms.
   Requirement addressed: The system shall allow scholars to visually

arrange fragments in an interactive workspace.

4.1.4 Area of Testing 4 — Script Identification and Transcription Assistance

The following tests verify that the system meets requirements S.4.4 and S.4.5 from the SRS, confirming accurate script classification and OCR-based transcription performance. These functions form the core of the text analysis pipeline and are critical for scholarly accuracy.

1. **Test ID:** test-fr-script-identification-1
   Control: Automatic
   Initial State: Uploaded fragment with visible text.
   Input: Image containing text of known script.
   Expected Output: Script classification with
   geq 70
   Test Case Derivation: Validates accuracy of handwriting classification model.
   How test will be performed: Model tested on labeled dataset of fragments; results compared to ground truth.
   Success criteria: The model assigns the correct script label with confidence
   geq 70
   Requirement addressed: The system shall identify the paleographic script type (e.g., Gupta, Tibetan, Chinese).

2. **Test ID:** test-fr-script-identification-2
   Control: Automatic
   Input: Fragment image with legible text.
   Expected Output: Generated text file with
   geq 70
   How test will be performed: Compare OCR output to manually transcribed ground truth using Levenshtein distance metric.
   Success criteria: OCR output achieves character accuracy
   geq 70
   Requirement addressed: The system shall perform OCR transcrip-

tion tuned for Sanskrit manuscripts.

4.1.5 Area of Testing 5 — Database Search and Exploration

These tests correspond to requirements S.4.6 of the SRS, which specify functionality for database search, filtering, and result export. The tests confirm that the query system performs efficiently and exports accurate data.

1. **Test ID:** test-fr-database-search-1
   Control: Automatic
   Initial State: Database with metadata indexed.
   Input: Query "Gupta script, > 5 lines."
   Expected Output: Filtered result set within 3 s response time.
   Test Case Derivation: Validates search performance and accuracy against known dataset.
   How test will be performed: API testing using Postman and performance profiling with JMeter.
   Success criteria: Query returns the correct filtered set and response time is under 3 s for the test dataset; results match the expected ground-truth set.
   Requirement addressed: The system shall enable multi-criteria search and filtering (by script type, line count, fragment size).

2. **Test ID:** test-fr-database-search-2
   Control: Manual
   Initial State: Search results displayed.
   Input: Click "Export".
   Expected Output: Downloaded CSV file containing fragment metadata.
   How test will be performed: Manual verification of downloaded file structure.
   Success criteria: Exported CSV conforms to the documented schema, contains the filtered results, and downloads without error in common browsers.
   Requirement addressed: The system shall support exporting query results to CSV or PDF.

4.2 Tests for Nonfunctional Requirements

The following subsections detail tests for system performance, AI matching quality, usability, reliability, data integrity, security, scalability, cross-browser compatibility, and performance benchmarking. These tests align with the nonfunctional requirements specified in the SRS document and provide measurable criteria for validating system quality attributes.

4.2.1 System Performance Testing

System performance testing validates that the platform meets specified performance requirements under expected workloads. These tests measure processing throughput for large fragment collections and system responsiveness under concurrent user load.

Collection Processing Performance These tests verify the system can process large collections of manuscript fragments within acceptable timeframes and handle concurrent research use as specified in the SRS.

1. **test-nfr-perf-1**

   Type: Dynamic, Automated, Performance

   Initial State: System deployed with empty database; sample dataset of 500 representative fragments from the British Library collection prepared for ingestion

   Input/Condition: Execute bulk fragment processing pipeline with sample dataset; monitor processing time, resource utilization, and completion rate

   Output/Result: Processing completes within 1.15 hours for 500 fragments; all fragments successfully processed with metadata extracted and indexed; system logs show no critical errors; extrapolated processing rate indicates full 21,000-fragment collection would process within $\leq$ 48 hours (per SRS NFR criteria)

   How test will be performed: Automated script will initiate the bulk processing workflow on the 500-fragment sample. Performance monitoring tools (e.g., time command, system resource monitors) will track execution time, CPU usage, memory consumption, and I/O operations. Upon completion, validation queries will verify that all fragments are searchable with extracted metadata. Test passes if: (1) processing time $\leq$ 1.15 hours for 500 fragments (extrapolates to $\leq$ 48 hours for full 21,000-fragment collection, meeting SRS requirement), (2) success rate $\geq$ 99%, and (3) no memory leaks or resource exhaustion detected. Processing rate will be calculated (fragments/hour) and documented for scaling projections.

2. **test-nfr-perf-2**

   Type: Dynamic, Automated, Load Testing

   Initial State: System fully operational with sample dataset (500-1000 fragments) loaded; load testing framework configured

   Input/Condition: Simulate 15 concurrent users performing typical research workflows (searches, filtering, canvas operations) over a 20-minute period to verify system can support 15+ concurrent users (per SRS NFR criteria)

   Output/Result: System maintains responsiveness throughout test period with 15 concurrent users; search queries return results within 3 seconds; canvas operations respond within 500ms; 95% of operations complete successfully without errors; no server crashes or performance degradation

   How test will be performed: Load testing tool (e.g., Locust or k6) will simulate 15 concurrent users executing scripted realistic research workflows. Test will ramp up from 1 to 15 users over 2 minutes, then maintain 15 concurrent users for 18 minutes. Test scenarios include: (1) search by fragment ID (10% of operations), (2) metadata filter queries (40%), (3) canvas workspace creation with 5-10 fragments (30%), (4) session save/restore operations (20%). Performance metrics (response times, error rates, throughput, server resource utilization) will be logged continuously. Test passes if: (1) mean search response time $\leq$ 2 seconds under load, (2) 95th percentile search time $\leq$ 3 seconds, (3) canvas operations complete within 500ms, (4) error rate $\leq$ 2%, (5) system successfully handles 15 concurrent users without crashes (meeting SRS requirement), and (6) no unhandled exceptions or out-of-memory errors occur. Results will include response time graphs, throughput metrics, and resource utilization charts.

4.2.2 AI Matching Assistance Quality

AI matching quality tests evaluate whether the system's machine learning-based fragment matching suggestions provide practical utility for scholarly research workflows.

Matching Suggestion Utility This test validates that AI-generated matching suggestions are sufficiently accurate and useful to assist researchers in identifying potential fragment connections.

1. **test-nfr-ai-1**

   Type: Dynamic, Manual, Expert Evaluation

   Initial State: System operational with AI matching model trained and deployed; test set of 20 fragment queries prepared with known ground truth (10 fragments with confirmed matches, 10 without matches in collection)

   Input/Condition: For each test fragment, request AI matching suggestions; record top 5 suggestions and their confidence scores

   Output/Result: For fragments with known matches: true match appears in top 5 suggestions in $\geq$ 60% of cases; for fragments without matches: system correctly identifies low confidence or provides diverse alternatives; overall suggestion relevance rated as ”potentially useful” by domain expert(s) in $\geq$ 50% of cases

   How test will be performed: Project supervisor and/or 1-2 domain experts (e.g., faculty member, graduate student in relevant field) will evaluate AI matching suggestions for the 30-fragment test set. For each fragment, evaluators will complete a standardized evaluation form.

   Evaluation Form (per fragment):

   - Fragment ID: [ID]
   - Known Ground Truth: [Has match in collection: Yes/No] [If yes, Fragment ID of match: ___]
   - Top 5 AI Suggestions: [List 5 suggestion IDs and confidence scores]
   - Ground Truth in Top 5?: [Yes/No]
   - Visual Feature Relevance: Do suggestions show similar edge patterns, damage, or script characteristics? [Yes/Somewhat/No]

- Overall Utility Rating: Would these suggestions be useful for scholarly investigation? [Useful / Potentially Useful / Not Useful]
- Comments: [Brief explanation of rating]

Results will be documented in a spreadsheet with one row per fragment. Test passes if: (1) known matches appear in top 5 for $\geq 60\%$ of test cases (6 out of 10 fragments with known matches), and (2) overall utility ratings are ”useful” or ”potentially useful” in $\geq 50\%$ of all 20 cases (10 or more fragments rated positively).

4.2.3 Usability Testing

Usability testing evaluates whether the system interface enables scholars to effectively perform manuscript reconstruction research tasks. These tests combine task-based user testing and systematic interface evaluation.

Research Workflow Task Completion Task-based usability testing measures whether users can successfully complete core research workflows and assesses subjective ease of use.

1. **test-nfr-usability-1**

   Type: Dynamic, Manual, User Testing

   Initial State: System fully operational with sample dataset loaded; test participants (3-4 users: project supervisor and/or 2-3 classmates) recruited; standardized task scenarios prepared

   Input/Condition: Each participant performs 5-6 representative research tasks: (1) search for specific fragment by ID, (2) apply metadata filters, (3) create canvas workspace, (4) arrange 3-5 fragments on canvas, (5) save session, (6) restore previous session

   Output/Result: Task completion rate $\geq 80\%$ across all participants and tasks (per SRS NFR criteria); average task completion time within specified targets; no more than 1 critical usability issue preventing task completion; participants rate overall ease of use as ”easy” or ”very easy” for $\geq 70\%$ of tasks

   How test will be performed: Participants will complete tasks in a moderated usability testing session (30-45 minutes per participant).

For each task, observers will record: completion status (success/fail/partial), time-on-task, number of errors, assistance required, and participant comments. Task-level success criteria: Task 1-2 (search/filter): ≤ 2 minutes each; Task 3-4 (canvas): ≤ 3 minutes each; Task 5-6 (save/restore): ≤ 1 minute each.

Post-test questionnaire (5-point Likert scale for each task):

- Q1-Q6: ”How easy was it to complete Task [1-6]?” (1=Very Difficult, 2=Difficult, 3=Neutral, 4=Easy, 5=Very Easy)
- Q7: ”What was the most confusing aspect of the system?” (open-ended)
- Q8: ”What feature did you find most helpful?” (open-ended)
- Q9: ”Would you recommend this system to colleagues for manuscript research?” (Yes/No/Maybe)

Test passes if: (1) overall task completion rate $\geq 80\%$ (aligns with SRS NFR requirement), (2) $\geq 70\%$ of individual task ratings (Q1-Q6) are 4 or 5 (”easy” or ”very easy”), and (3) no more than 1 critical usability issue identified that completely blocks task completion.

2. **test-nfr-usability-2**

   Type: Static, Manual, Interface Evaluation

   Initial State: System interface fully implemented and accessible for evaluation; usability evaluation criteria checklist prepared

   Input/Condition: 2 team members (who did not design the specific interface components being evaluated) independently evaluate the interface against core usability principles

   Output/Result: Consolidated report identifying usability issues categorized by principle and severity (critical, major, minor); no more than 3 critical violations identified; prioritized list of recommended improvements documented

   How test will be performed: Each team member evaluator will spend 1-2 hours systematically exploring the interface and documenting usability issues. Evaluators will use a standardized checklist covering core usability principles:

   Evaluation Checklist:

(a) **Clarity**: For each major interface element (search bar, filter panel, canvas controls), is it clear what actions are available and what they do? Rate: Pass/Fail with examples.

(b) **Feedback**: When performing actions (search, drag fragment, save session), does the system provide clear visual or textual feedback? Rate: Pass/Fail with examples.

(c) **Consistency**: Are similar operations (e.g., all "save" actions, all "delete" actions) performed in similar ways? Rate: Pass/Fail with examples.

(d) **Error Prevention & Recovery**: Can users prevent errors (confirmation dialogs) and recover from mistakes (undo, clear error messages)? Rate: Pass/Fail with examples.

(e) **Efficiency**: Can users accomplish core tasks (search → filter → canvas → save) with minimal unnecessary steps? Count steps for each workflow. Rate: Pass/Fail.

(f) **Learnability**: Can a new user understand how to perform basic search and canvas operations without documentation? Rate: Pass/Fail with observations.

For each issue identified, document: (1) specific location/feature, (2) principle violated, (3) severity rating (critical/major/minor), (4) screenshot or description, (5) suggested fix. Severity definitions: critical (prevents task completion), major (causes significant frustration or confusion), minor (cosmetic or minor inconvenience). Team will meet to consolidate findings and create prioritized issue list. Test passes if: (1) no more than 3 critical violations identified, (2) evaluation report completed documenting all findings with specific examples, and (3) critical issues have documented remediation plans before final release.

4.2.4 System Availability and Reliability

Reliability testing validates that the system maintains consistent availability for academic research and can recover from failures without data loss.

Uptime and Availability Testing These tests measure system uptime and verify recovery capabilities to ensure the platform reliably supports ongoing research activities.

1. **test-nfr-reliability-1**

   Type: Dynamic, Automated, Reliability Testing

   Initial State: System deployed to test/staging environment; GitHub Actions workflow configured for automated monitoring

   Input/Condition: Continuous operation over 7-day period with automated health checks every 30 minutes via GitHub Actions scheduled workflow; simulated user activity via automated test script

   Output/Result: System availability $\geq$ 95% during the test period (per SRS NFR criteria); no more than 2 unplanned outages lasting > 1 hour; system recovers automatically or can be restarted successfully after any failures

   How test will be performed: GitHub Actions scheduled workflow (using cron syntax) will run every 30 minutes to perform health checks: (1) HTTP request to home page (expect 200 status), (2) execute test search query via API (verify JSON response), (3) verify database connectivity through health endpoint. Each workflow run logs results to a file committed to repository with timestamp and status. Additionally, a separate workflow runs hourly to simulate realistic user activity (perform searches, create canvas session, save/restore operations) and verify end-to-end functionality. All workflow runs are recorded in GitHub Actions history. For any failed health check: workflow creates GitHub issue automatically with failure details and timestamp. Test passes if: (1) uptime $\geq$ 95% over 7 days (calculated as successful health checks / total health checks; allowing maximum 8.4 hours downtime), (2) no more than 2 unplanned outages lasting > 1 hour, and (3) system recovers within 1 hour of failure detection (verified through subsequent health check passes). Results documented in availability report generated from workflow logs showing: total checks, successful checks, failed checks, outage periods, and uptime percentage.

2. **test-nfr-reliability-2**

   Type: Dynamic, Manual, Disaster Recovery

   Initial State: System operational with sample dataset loaded; backup procedures documented and tested; test environment available for recovery simulation

Input/Condition: Simulate critical failure scenario (e.g., accidental database deletion, container/server crash); initiate recovery procedures using documented backup strategy

Output/Result: System restored to functional state within 2 hours; sample dataset and user data successfully recovered with no loss; all core functionality (search, canvas, authentication) verified operational post-recovery

How test will be performed: Recovery test will be conducted in isolated test environment to avoid impacting production. Test scenario: (1) create backup of current system state (database, uploaded images, configuration), (2) simulate failure by stopping services and removing/corrupting critical data, (3) execute documented recovery procedures from backup, (4) measure time from ”failure” to full restoration. Post-recovery verification checklist: database accessible and contains expected records, search returns correct results for sample queries, canvas workspace can be created and manipulated, user authentication works, uploaded images accessible. Test passes if: (1) recovery completed within 2 hours, (2) all verification checks pass, (3) no data loss detected (verified by record count and sample data spot-check), and (4) recovery procedure documentation is sufficient for team member unfamiliar with process to execute successfully.

4.2.5 Data Integrity and Security

Data integrity and security testing ensures that fragment metadata and provenance information are preserved accurately and that the system protects against common security vulnerabilities.

Data Preservation and Attribution These tests verify that all fragment data maintains referential integrity and proper attribution throughout the system lifecycle.

1. **test-nfr-data-1**

   Type: Functional, Automated, Data Validation

   Initial State: British Library collection metadata and user-submitted images loaded into system

Input/Condition: Execute comprehensive data validation checks across entire collection

Output/Result: 100% of fragments retain original British Library catalog information (shelfmark, collection, provenance); all user-submitted images maintain uploaded attribution; no corruption or loss of metadata fields

How test will be performed: Automated validation script will query all fragments in the database and verify metadata integrity.

Required Metadata Fields:

- **Fragment ID**: Unique identifier (non-null, unique constraint)
- **Collection Source**: "British Library" or user-uploaded indicator (non-null)
- **Shelfmark**: Original catalog reference (required for British Library fragments)
- **Provenance**: Geographic/historical origin information (if available in source)
- **Image URL/Path**: Location of fragment image (non-null, valid path)
- **Upload Attribution**: Username or source for user-submitted fragments (non-null for user uploads)
- **Timestamp**: Date added to system (non-null)

Validation checks: (1) all required fields present and non-null, (2) British Library fragments cross-referenced against source catalog CSV (shelfmark, collection matches), (3) user-uploaded fragments have valid attribution field, (4) referential integrity verified (foreign keys valid, no orphaned records). Script generates exception report listing any fragments with missing/inconsistent data. Manual spot-check of 100 randomly selected fragments verifies field accuracy. Test passes with 100% of fragments having all required fields populated and $\geq$ 99% matching source data exactly.

2. **test-nfr-data-2**

   Type: Static, Manual, Code Review

Initial State: Data handling, authentication, and API code modules identified for review; security review checklist prepared based on OWASP guidelines

Input/Condition: Security-focused code review conducted by team members not primarily responsible for implementing the reviewed modules, supplemented by automated security scanning tools

Output/Result: Review checklist completed with findings documented; no critical security vulnerabilities identified; common security best practices verified (input validation, SQL injection prevention, authentication, authorization, API security)

How test will be performed: Two team members (preferably those who did not write the security-critical code) will conduct peer code review focusing on security aspects. Review checklist will cover: (1) input validation and sanitization for user-provided data, (2) SQL injection prevention (parameterized queries/ORM usage), (3) authentication mechanisms (password hashing, session management), (4) authorization controls (proper access checks before sensitive operations), (5) secure handling of API keys and secrets (not committed to repository), (6) protection against common web vulnerabilities (XSS, CSRF). Additionally, run automated security scanning tool appropriate for tech stack (e.g., npm audit for Node.js, Bandit for Python, pip-audit, or built-in IDE security warnings). All findings documented with severity ratings (critical/high/medium/low). Test passes if: (1) no critical or high-severity vulnerabilities identified in manual review, (2) checklist completed for all security-critical modules (authentication, data access, API endpoints), (3) automated scanning shows no critical issues, and (4) any medium/low findings have documented remediation plans or accepted risk justifications.

4.2.6 Scalability Testing

Scalability testing analyzes how system performance changes as the fragment collection size increases, providing data to support future scaling decisions.

Fragment Collection Scalability This test measures system performance with varying dataset sizes to project scalability to larger fragment collections.

1. **test-nfr-scale-1**

   Type: Dynamic, Manual, Scalability Analysis

   Initial State: System operational with sample dataset (500 fragments); ability to add additional test data

   Input/Condition: Test system performance with incrementally larger datasets: 500 fragments (baseline), 2,000 fragments, 5,000 fragments

   Output/Result: Performance metrics documented for each dataset size showing how system scales; search response times remain acceptable ($\leq$ 5 seconds) at 5,000 fragments; documented analysis of expected performance at full 21,000-fragment scale based on observed trends

   How test will be performed: Using existing sample data (500 fragments), create duplicate test data to reach 2,000 and 5,000 fragments (duplicates acceptable for testing purposes). At each dataset size, execute standardized queries: (1) fragment ID search (10 queries, measure average time), (2) single metadata filter (10 queries, measure average time), (3) multi-criteria filter with 2-3 filters (5 queries, measure average time). Record response times and database query execution times. Document results in simple table showing dataset size vs. average response time for each query type. Analyze trend (linear, logarithmic, exponential) and extrapolate to 21,000 fragments. Test passes if response times remain usable ($\leq$ 5 seconds) at 5,000 fragments and trend analysis suggests acceptable performance at full scale.

4.2.7 Cross-Browser and Device Compatibility

Cross-browser compatibility testing ensures that the web-based interface functions correctly across major desktop browsers used by researchers.

Interface Compatibility Testing This test verifies that core functionality operates consistently across Chrome and at least one additional modern browser.

1. **test-nfr-compat-1**

   Type: Manual, Dynamic, Compatibility Testing

   Initial State: System deployed and accessible via URL

Input/Condition: Access system and perform core workflows on multiple browser and device combinations: Chrome, Firefox, Safari (desktop); Chrome, Safari (mobile); tablets

Output/Result: All core functionality (search, filter, canvas) operates correctly across tested platforms; visual layout renders appropriately; no critical browser-specific bugs identified

How test will be performed: Test team will execute standardized test suite on each browser/device combination.

Test Suite (perform on each platform):

(a) **Search Test**: Enter fragment ID "BL-12345" in search bar, verify results display correctly

(b) **Filter Test**: Apply metadata filter (e.g., "Script: Brahmi"), verify filtered results

(c) **Canvas Creation**: Click "New Canvas", verify canvas workspace opens

(d) **Drag-and-Drop**: Drag 2 fragment images onto canvas, verify they appear and can be repositioned

(e) **Image Rendering**: Verify fragment images load at full resolution, zoom controls work

(f) **Session Save**: Save canvas session, verify success message

(g) **Session Restore**: Reload page, restore saved session, verify fragments reappear in correct positions

(h) **Responsive Layout**: Resize browser window to mobile width (375px), verify layout adapts without breaking

Target Platforms: Chrome (latest, primary development browser), Firefox (latest, secondary browser), Safari (latest macOS if team has access).

For each test-platform combination, record: Pass/Fail, any errors/warnings, visual issues, performance notes. Issues documented with severity (critical/major/minor) and browser-specific details. Test passes if all critical functionality works on Chrome and ≥ 1 additional browser. Critical issues on Chrome must be resolved; issues on secondary browsers documented for future work. Mobile testing is optional/out of scope for this project phase.

4.2.8 Performance Benchmarking

Performance benchmarking establishes baseline metrics for system response times that can be used for regression testing in future development.

Response Time Measurement This test documents average response times for common operations to establish performance baselines.

1. **test-nfr-bench-1**

   Type: Dynamic, Manual, Benchmark Testing

   Initial State: System operational with sample dataset (500-1000 fragments)

   Input/Condition: Execute standardized set of operations representing typical usage patterns: (1) fragment ID search, (2) single metadata filter, (3) multi-criteria filter (2-3 criteria), (4) canvas operations with 5-10 fragments

   Output/Result: Performance report documenting average response times for each operation type; simple table showing mean response time for each operation; baseline metrics documented for future regression testing

   How test will be performed: For each operation type, execute 10 test runs and record response times using browser developer tools or simple timing script. Calculate average (mean) response time for each operation type. Document results in simple table format: Operation Type — Average Response Time — Notes. Example acceptable targets: ID search $\leq$ 1 second, single filter $\leq$ 2 seconds, multi-filter $\leq$ 3 seconds, canvas operations $\leq$ 2 seconds. This benchmark establishes baseline performance metrics for future comparison rather than strict pass/fail criteria. Results provide data for identifying performance regressions in future releases.

4.3 Traceability Between Test Cases and Requirements

Req. ID	Description	Test ID(s)	Method
S.4.1	Secure login and authentication	test-fr-auth-1 to test-fr-auth-5	Manual UI
S.4.2	Fragment upload and preprocessing	test-fr-upload-1 to test-fr-upload-4, test-fr-preprocess-1 to test-fr-preprocess-3	Manual and automated
S.4.3	Fragment matching and reconstruction	test-fr-match-1 to test-fr-match-6	Manual UI
G.4.3, S.4.3	Interactive canvas workspace	test-fr-canvas-1 to test-fr-canvas-8	Manual UI
S.4.4	Script identification	test-fr-script-1 to test-fr-script-4	Automated and manual
S.4.5	Transcription assistance	test-fr-transcript-1 to test-fr-transcript-7	Automated OCR and manual
S.4.6	Database search and exploration	test-fr-search-1	Manual API and UI

Table 2: Traceability for Functional Requirements

Req. ID	**Description**	**Test ID(s)**	**Method**
S.6, E.3	System performance	test-nfr-perf-1, test-nfr-perf-2	Automated
S.6	AI matching quality	test-nfr-ai-1	Manual expert eval
S.6, S.2.1	Usability	test-nfr-usability-1, test-nfr-usability-2	Manual testing
S.6, S.2.2	Reliability	test-nfr-reliability-1, test-nfr-reliability-2	Automated and manual
E.6, S.6, E.3	Data integrity and security	test-nfr-data-1, test-nfr-data-2	Automated and code review
E.3, S.6	Scalability	test-nfr-scale-1	Manual analysis
S.2.1	Cross-browser compatibility	test-nfr-compat-1	Manual
S.6	Performance benchmarking	test-nfr-bench-1	Manual benchmark

Table 3: Traceability for Non-Functional Requirements

5 Unit Test Description

[This section should not be filled in until after the MIS (detailed design document) has been completed. —SS]

[Reference your MIS (detailed design document) and explain your overall philosophy for test case selection. —SS]

[To save space and time, it may be an option to provide less detail in this section. For the unit tests you can potentially layout your testing strategy here. That is, you can explain how tests will be selected for each module. For instance, your test building approach could be test cases for each access program, including one test for normal behaviour and as many tests as needed for edge cases. Rather than create the details of the input and output here, you could point to the unit testing code. For this to work, you code needs to be well-documented, with meaningful names for all of the tests. —SS]

5.1 Unit Testing Scope

[What modules are outside of the scope. If there are modules that are developed by someone else, then you would say here if you aren't planning on verifying them. There may also be modules that are part of your software, but have a lower priority for verification than others. If this is the case, explain your rationale for the ranking of module importance. —SS]

5.2 Tests for Functional Requirements

[Most of the verification will be through automated unit testing. If appropriate specific modules can be verified by a non-testing based technique. That can also be documented in this section. —SS]

5.2.1 Module 1

[Include a blurb here to explain why the subsections below cover the module. References to the MIS would be good. You will want tests from a black box perspective and from a white box perspective. Explain to the reader how the tests were selected. —SS]

1. test-id1

   Type: [Functional, Dynamic, Manual, Automatic, Static etc. Most will be automatic —SS]

   Initial State:

   Input:

   Output: [The expected result for the given inputs —SS]

   Test Case Derivation: [Justify the expected value given in the Output field —SS]

   How test will be performed:

2. test-id2

   Type: [Functional, Dynamic, Manual, Automatic, Static etc. Most will be automatic —SS]

Initial State:

Input:

Output: [The expected result for the given inputs —SS]

Test Case Derivation: [Justify the expected value given in the Output field —SS]

How test will be performed:

3. ...

5.2.2 Module 2

...

5.3 Tests for Nonfunctional Requirements

[If there is a module that needs to be independently assessed for performance, those test cases can go here. In some projects, planning for nonfunctional tests of units will not be that relevant. —SS]

[These tests may involve collecting performance data from previously mentioned functional tests. —SS]

5.3.1 Module ?

1. test-id1

   Type: [Functional, Dynamic, Manual, Automatic, Static etc. Most will be automatic —SS]

   Initial State:

   Input/Condition:

   Output/Result:

   How test will be performed:

2. test-id2

   Type: Functional, Dynamic, Manual, Static etc.

Initial State:

Input:

Output:

How test will be performed:

5.3.2 Module ?

...

5.4 Traceability Between Test Cases and Modules

[Provide evidence that all of the modules have been considered. —SS]

References

Author Author. System requirements specification. https://github.com/..., 2019.

Team Sanskrit Ciphers. Design documents (mg and mis). https://github.com/DylanG5/sanskrit-cipher/blob/43484da11e4fb0f879f7bbf2377c21900e5b9838/docs/Design/SoftDetailedDes/MIS.pdf, 2025a. To be developed in later phases; describes system architecture and module decomposition.

Team Sanskrit Ciphers. Development plan. https://github.com/DylanG5/sanskrit-cipher/blob/43484da11e4fb0f879f7bbf2377c21900e5b9838/docs/DevelopmentPlan/DevelopmentPlan.pdf, 2025b. Describes workflow, scheduling, task assignments, and CI/CD strategy.

Team Sanskrit Ciphers. Hazard analysis document. https://github.com/DylanG5/sanskrit-cipher/blob/43484da11e4fb0f879f7bbf2377c21900e5b9838/docs/HazardAnalysis/HazardAnalysis.pdf, 2025c. Identifies potential hazards, risks, and mitigation strategies.

Team Sanskrit Ciphers. Problem statement and goals document. https://github.com/DylanG5/sanskrit-cipher/blob/43484da11e4fb0f879f7bbf2377c21900e5b9838/docs/

ProblemStatementAndGoals/ProblemStatement.pdf, 2025d. Defines project motivation, scope, and expected outcomes.

Team Sanskrit Ciphers. Software requirements specification (srs). https://github.com/DylanG5/sanskrit-ciphers-requirements/blob/9938018b682709551d0a6248a0a9f6e5794112ee/index.pdf, 2025e. Specifies functional and non-functional requirements for the system.

Team Sanskrit Ciphers. Usability report. https://github.com/DylanG5/sanskrit-cipher/tree/43484da11e4fb0f879f7bbf2377c21900e5b9838/docs/Extras/UsabilityTesting, 2025f. Details findings from surveys highlighting effectiveness of platform.

Team Sanskrit Ciphers. User manual. https://github.com/DylanG5/sanskrit-cipher/blob/43484da11e4fb0f879f7bbf2377c21900e5b9838/docs/UserGuide/UserGuide.pdf, 2025g. Guides end users on how to operate the platform.

6 Appendix

This is where you can place additional information.

6.1 Symbolic Parameters

The definition of the test cases will call for SYMBOLIC_CONSTANTS. Their values are defined in this section for easy maintenance.

6.2 Usability Survey Questions?

[This is a section that would be appropriate for some projects. —SS]

Appendix — Reflection

[This section is not required for CAS 741 —SS]

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

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

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

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

1. What went well while writing this deliverable?

   The fact that we had a solid base from our SRS to work from was one of this deliverable's strongest points. We were able to directly map the SRS's structured sections and clearly defined functional requirements into test areas, which greatly streamlined the planning process. Assigning ownership for test design and documentation was also simple because we had already finished the Development Plan and each of us knew our general responsibilities. This helped us stay organized. Throughout this phase, we worked effectively as a team, checked each other's sections for coherence, and developed trust that the verification plan related to our earlier work.

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

   Time management was our biggest challenge during this phase, as the deliverable overlapped with midterms and major assignments in other courses. We initially underestimated how much time would be required to properly structure the test cases and trace them back to the requirements. Since our platform integrates multiple components (front end, back end, and multiple AI/ML services), defining the boundaries between modules for testing was another significant challenge. This took longer than anticipated. In order to resolve this, we had brief group discussions to determine which system components would be verified at the unit level as opposed to the system level. These discussions also made sure that everyone understood the differences between validation and verification in the context of our platform. By the end, we were more assured of how we divided up the work and how we interpreted the testing scope.

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.
   To carry out the next stages of verification effectively, our team identified several areas for skill development. We need to strengthen our understanding of automated testing frameworks, particularly PyTest and coverage.py, to ensure consistent and maintainable test execution. We also plan to expand our familiarity with static and dynamic analysis tools like Ruff, MyPy, and Bandit for enforcing code quality and identifying potential vulnerabilities. Another important learning area will be validating the accuracy and robustness of our machine learning models — for example, evaluating OCR performance using accuracy metrics, confusion matrices, and ground truth comparison datasets. Collectively, these skills will help us achieve more objective, measurable verification results and reduce the likelihood of regression errors in later phases.

To carry out the next stages of verification effectively, our team identified several areas for skill development. We need to strengthen our understanding of automated testing frameworks, particularly PyTest and coverage.py, to ensure consistent and maintainable test execution. We also plan to expand our familiarity with static and dynamic analysis tools like Ruff, MyPy, and Bandit for enforcing code quality and identifying potential vulnerabilities. Another important learning area will be validating the accuracy and robustness of our machine learning models, for example, evaluating OCR performance using accuracy metrics, con-

fusion matrices, and ground truth comparison datasets. Collectively, these skills will help us achieve more objective, measurable verification results and reduce the likelihood of errors in later phases.

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?

   As a team, we plan to take a mixed approach that combines self-learning, collaborative practice, and direct application within the project. To establish a technical foundation, everyone in the group will adhere to the PyTest, coverage.py, and static analysis tools' official documentation and focused tutorials. Then, using GitHub Actions to run automated tests continuously and track coverage over time, we will further solidify this learning by implementing them early in the development process. We intend to incorporate brief knowledge-sharing periods into our weekly team meetings, where a team member can show the others a testing or verification method. In order to guide our validation setup for the AI components, we plan to review open-source evaluation scripts from Optical Character Recognition(OCR) and similarity-based matching research papers. This collaborative, iterative approach ensures that every member not only understands the testing tools but also applies them meaningfully within their assigned subsystem.