Test Case Assignment: Scientific Paper Analyzer: Cloud-Deployed Summarization and Q&A Tool

Team Members: Ali Talasaz, Evan Parra, Bharath Sai Vallabhaneni, Ashritha Aloori

1. Introduction

This document defines the formal test cases for validating the functionality, performance, and reliability of the Scientific Paper Analyzer system. The system enables users (students, researchers, faculty, and maintainers) to upload, summarize, and interact with academic documents through a cloud-deployed AI platform. Each test case follows a structured format describing objectives, preconditions, procedures, and expected outcomes to ensure all user, functional, and non-functional requirements are thoroughly validated.

2. User Requirement Test Cases

****
*** User – Student ***
****

TC-UR1.1 – Student/Researcher Authentication and Session Access

- **Objective:** Verify that students and researchers can securely sign in before accessing any document-related functions such as uploading, summarizing, or Q&A.
- **Scope:** Tests user authentication workflow and access control for student/researcher accounts.
- **Configuration:**
  - Authentication Method: Email + Password or Institutional Login (OAuth)
  - Platform: Web UI / Chat Interface
  - Session Timeout: 15 minutes of inactivity
- **Preconditions:**
  - User is registered in the system database.
  - Login services and session manager are active.
- **Procedure:**
  - Navigate to the login screen.
  - Enter valid credentials and submit.
  - Attempt to upload a PDF or ask a question before login (negative test).
  - Attempt again after successful authentication.
- **Expected Results:**
  - Unauthorized access attempts are blocked.
  - Successful login grants full access to document upload, summary, and Q&A features.
  - Session remains valid until timeout or explicit logout.
- **Postconditions:** Active session created and tracked in UserSession database; activity logs recorded for traceability.

TC-UR1.2 – Upload Academic Documents in PDF Format

- **Objective:** Validate that users can successfully upload academic papers in PDF format through the web interface.

- **Scope:** Covers file upload functionality, format validation, and storage in the backend system.
- **Configuration:**
  - Interface: Gradio / Web UI
  - Backend: Google Cloud Storage
  - File Types Supported: .pdf only
- **Preconditions:**
  - User has an active session.
  - PDF upload option available in the interface.
- **Procedure:**
  - Click the Upload Paper button.
  - Select a valid PDF document from the local machine.
  - Observe system response for upload confirmation.
  - Attempt to upload a non-PDF file (e.g., .docx) to verify format restriction.
- **Expected Results:**
  - PDF uploads succeed and are stored correctly.
  - Non-PDF formats are rejected with a clear error message.
  - Confirmation message appears with filename and document ID.
- **Postconditions:** Document stored in the system and indexed for summarization and retrieval.

TC-UR1.3 – Generate Concise Running Summaries of Uploaded Content

- **Objective:** Ensure that the system automatically generates concise summaries highlighting main findings from the uploaded document.
- **Scope:** Tests the summarization pipeline's accuracy and coherence in generating running summaries.
- **Configuration:**
  - Summarization Engine: NVIDIA LLM or equivalent
  - Backend: LangChain summarization chain
  - Output Fields: Running summary, key findings, loose ends
- **Preconditions:**
  - PDF document successfully uploaded and processed.
  - Summarization endpoint operational.
- **Procedure:**
  - After upload, click Generate Summary.
  - Observe progress and verify that a summary appears.
  - Review whether it includes 3–5 key findings.
- **Expected Results:**
  - Summary produced automatically within 10 s.
  - Key findings are clear, non-redundant, and factual.
  - Summary text stored for continuous updates as the session continues.
- **Postconditions:** Summary saved in the database; ready for contextual retrieval during Q&A.

TC-UR1.4 – Ask Natural-Language Questions and Receive Context-Grounded Answers

- **Objective:** Confirm that users can ask natural-language questions and receive citation-grounded answers derived from the uploaded document.
- **Scope:** Verifies the document-aware Q&A system and retrieval accuracy.
- **Configuration:**

  - Retrieval Engine: FAISS vector store
  - Language Model: NVIDIA or OpenAI API
  - Interface: Chat UI (Gradio or Discord)
- **Preconditions:**
  - Document already summarized and embedded.
  - User session active.
- **Procedure:**
  - Type a natural question such as “What dataset was used in the experiment?”
  - Observe the model response.
  - Ask a follow-up question to test context retention.
- **Expected Results:**
  - Responses are factually consistent with the paper.
  - Each answer includes supporting citations or section references.
  - Context continuity maintained across multiple questions.
- **Postconditions:** Q&A logs recorded for later validation and fine-tuning.

TC-UR1.5 – Retrieve Key Sections or Highlights from Documents

- **Objective:** Validate that users can retrieve specific sections or highlighted portions (e.g., abstract, methods, results) upon request.
- **Scope:** Tests retrieval and segmentation capabilities within the document analysis pipeline.
- **Configuration:**
  - Retrieval Module: LangChain retriever with FAISS index
  - Commands: “Show methods section,” “Highlight main results”
- **Preconditions:**
  - Uploaded paper indexed and chunked into retrievable sections.
  - Chat or command interface accessible.
- **Procedure:**
  - Ask the chatbot “Retrieve the abstract section.”
  - Request “Show highlighted results or conclusions.”
  - Compare output text with the corresponding document section.
- **Expected Results:**
  - Returned sections match the user’s query precisely.
  - Highlights preserve document structure and original formatting.
  - Retrieval is completed in under 3 seconds.
- **Postconditions:** Retrieved excerpts cached for repeated access within the current session.

****
*** User – Faculty **
****

TC-UR2.1 – Faculty/Reviewer Authentication and Validation Privileges

- **Objective:** Ensure that faculty and reviewers can authenticate securely to validate generated summaries and responses.
- **Scope:** Covers authentication, access privilege enforcement, and reviewer-level actions.
- **Configuration:**
  - Authentication via Secure Faculty Portal (OAuth or Institutional SSO)
  - Role-Based Access Control (RBAC) enforced through API
- **Preconditions:**

  - Faculty user account with reviewer privileges exists.
  - System connected to reviewer verification service.
- **Procedure:**
  - Log in using faculty credentials.
  - Attempt to access validation dashboard without logging in (negative test).
  - Validate system summaries and submit review feedback.
- **Expected Results:**
  - Only authenticated faculty can access review tools.
  - Review actions (approve, flag, comment) are recorded with user ID.
  - Unauthorized users receive “Access Denied” message.
- **Postconditions:** Validation logs updated; reviewer actions tagged for QA analysis.

TC-UR2.2 – Validate Summaries and Answers for Correctness

- **Objective:** Verify that the system generates accurate summaries and responses consistent with the original paper content.
- **Scope:** Ensures factual alignment between the AI-generated output and the uploaded research document.
- **Configuration:**
  - Deployment: Cloud Run with NVIDIA LLM endpoint
  - Storage: Google Cloud Storage
  - Interface: Gradio Chat / Web UI
- **Preconditions:**
  - Faculty user logged in with an uploaded paper ready.
  - Summary and Q&A functions accessible through the main interface.
- **Procedure:**
  - Open a previously uploaded academic paper.
  - Click “Generate Summary” and review the generated content.
  - Ask two domain-specific questions about methods or findings.
  - Compare AI answers with the source document manually.
- **Expected Results:**
  - Summaries are concise, comprehensive, and free from factual errors.
  - Each answer correctly references information present in the original document.
- **Postconditions:** Faculty validation results logged for performance benchmarking.

TC-UR2.3 – Identify Methodologies, Results, and Key Insights

- **Objective:** Ensure the tool allows faculty to extract specific methodological and result-based insights efficiently.
- **Scope:** Tests the document segmentation and context retrieval capability of the RAG subsystem.
- **Configuration:**
  - LangChain RAG pipeline integrated with FAISS vector store.
  - Access via “Ask a Question” or “Highlight Sections” command.
- **Preconditions:**
  - Uploaded document indexed in the vector database.
  - Model ready to process section-level retrieval.
- **Procedure:**
  - Ask “Summarize the methodology section.”
  - Ask “List the primary results or conclusions.”
  - Observe whether responses maintain section context.

- **Expected Results:**
    - Retrieved text includes correct section-level summaries (methods, results).
    - Responses highlight critical steps or findings clearly.
- **Postconditions:** Faculty feedback saved for fine-tuning summarization and retrieval accuracy.

TC-UR2.4 – Save Interactions for Later Reference

- **Objective:** Validate that faculty can save, retrieve, and review prior interactions for peer review or future teaching use.
- **Scope:** Ensures persistence of chat sessions and stored outputs in the knowledge base.
- **Configuration:**
    - Cloud-hosted KnowledgeBase with session persistence enabled.
    - API endpoints: `/save_session` and `/load_session`.
- **Preconditions:**
    - Active session with summary and Q&A history.
    - User logged in with faculty access.
- **Procedure:**
    - Complete several Q&A exchanges with the chatbot.
    - Click "Save Session" from the interface menu.
    - Log out and back in.
    - Click "Load Previous Session."
- **Expected Results:**
    - Previous conversations, summaries, and questions restored accurately.
    - No data corruption or loss observed during retrieval.
- **Postconditions:** Session metadata stored for academic recordkeeping and future analysis.

****
*********************************** Developer / Maintainer ***************************************
****

TC-UR3.1 – Developer/Maintainer Authentication and System Access

- **Objective:** Verify that developers and maintainers can securely authenticate to perform administrative or maintenance operations.
- **Scope:** Tests privileged access control, system configuration security, and logging of developer sessions.
- **Configuration:**
    - Admin Portal: `/admin/dashboard` (protected endpoint)
    - Authentication: 2FA (username + OTP)
    - Logging: Cloud Run access logs
- **Preconditions:**
    - Developer account created with admin privileges.
    - Two-Factor Authentication enabled.
- **Procedure:**
    - Attempt to access `/admin/dashboard` without logging in.
    - Authenticate with username, password, and OTP.
    - View and update model settings or logs.
- **Expected Results:**
    - Unauthorized users are redirected to login.

    - Developers can view logs, retrain models, or restart services only after authentication.
    - All administrative actions are logged for audit.
- **Postconditions:** Secure admin session established; logs updated with timestamp, IP, and activity type.

TC-UR3.2 – Manage User Sessions and System Logging

- **Objective:** Verify that the system effectively manages active user sessions and logs all key interactions for traceability, debugging, and performance monitoring.
- **Scope:** Covers user session persistence, log integrity, and audit trace generation in the deployed cloud environment.
- **Configuration:**
    - Deployment: Google Cloud Run (managed service)
    - Logging: Google Cloud Logging with structured JSON format
    - Session store: Firestore or equivalent backend for state persistence
- **Preconditions:**
    - Multiple user sessions active simultaneously.
    - Logging service connected and permissions enabled for Cloud Run service account.
- **Procedure:**
    - Initiate several concurrent chat sessions under different user IDs.
    - Perform standard actions — document upload, summarization, and Q&A.
    - Review system logs for user identifiers, timestamps, and request responses.
    - Terminate one session and restart it.
- **Expected Results:**
    - Each session retains continuity across reconnects.
    - Logs include detailed metadata (timestamp, user ID, request type, execution time).
    - No missing or duplicated log entries detected.
- **Postconditions:** Session and log data stored securely; system logs retrievable via Cloud Console or API.

TC-UR3.3 – Maintain Scalability for Multiple Concurrent Users

- **Objective:** Ensure the system can efficiently support multiple users without degradation in performance or latency.
- **Scope:** Validates the scalability of backend components including Cloud Run autoscaling, FAISS retrieval, and model-serving endpoints.
- **Configuration:**
    - Environment: Google Cloud Run with concurrency set to ≥50
    - Load testing tool: Apache JMeter or Locust
    - LLM service via NVIDIA endpoint
- **Preconditions:**
    - System deployed with autoscaling enabled.
    - Load generator configured with 50+ simultaneous virtual users.
- **Procedure:**
    - Launch load test simulating 50–100 concurrent users performing uploads and Q&A.
    - Record metrics: latency, error rate, and resource utilization.
    - Gradually increase concurrent requests to test scaling thresholds.

- **Expected Results:**
    - 95% of requests completed within 3 seconds.
    - No request failures or API throttling observed.
    - Autoscaler triggers new container instances automatically.
- **Postconditions:** Performance report generated; scaling thresholds logged for further optimization.

TC-UR3.4 – Update Vector Stores, Models, and Interfaces

- **Objective:** Validate the ability to update embeddings, retrievers, or LLM endpoints without interrupting live user sessions or causing data loss.
- **Scope:** Tests system's modular deployment structure for zero-downtime updates to AI components.
- **Configuration:**
    - Vector storage: FAISS hosted on persistent disk or cloud bucket
    - Model endpoints: NVIDIA NIM or equivalent
    - Deployment: Blue-Green rollout via Cloud Run revisioning
- **Preconditions:**
    - System running with active users.
    - Updated vector store or model version ready for deployment.
- **Procedure:**
    - Trigger Cloud Run revision update using the new image version.
    - Monitor existing user sessions during deployment.
    - After deployment, revalidate summarization and Q&A operations.
    - Compare retrieval accuracy and model latency before and after update.
- **Expected Results:**
    - Update completes with zero downtime.
    - Vector store retains all previous embeddings.
    - System interfaces (ChatAPI, UI) remain accessible and responsive.
- **Postconditions:** New model revision marked active; prior version archived for rollback if needed.

TC-UR3.5 – Verify Local vs. Cloud Deployment Consistency

- **Objective:** Ensure application behavior is identical when run locally (e.g., via Docker) versus deployed on Cloud Run.
- **Scope:** Tests deployment parity for developers.
- **Configuration:**
    - Local: Docker; Cloud: Google Cloud Run with same container image.
- **Preconditions:**
    - Container built; Test scripts for upload/summarize/Q&A.
- **Procedure:**
    - Run tests locally, then deploy and run on Cloud Run.
    - Compare outputs and latencies.
- **Expected Results:**
    - 100% functional match; Latency difference <20%.
- **Postconditions:** Deployment logs compared.

TC-UR3.6 – Test Blue-Green Deployment for Zero-Downtime Updates

- **Objective:** Validate updating services (e.g., new model version) without interrupting

active sessions.
- **Scope:** Tests revision management on Cloud Run.
- **Configuration:**
    - Deployment: Blue-Green via Cloud Run revisions; Traffic: Tag-based routing.
- **Preconditions:**
    - Active service with users; New revision ready.
- **Procedure:**
    - Deploy new revision with tag; Route test traffic to it.
    - Switch production traffic after validation.
- **Expected Results:**
    - No downtime; Sessions persist across switch.
- **Postconditions:** Old revision archived.

TC-UR3.7 – Test Cloud Build Integration for CI/CD

- **Objective:** Verify automated builds and deployments via Cloud Build for code changes.
- **Scope:** Tests CI/CD pipeline for maintainers.
- **Configuration:**
    - Trigger: GitHub push; Build: Docker image to Artifact Registry.
- **Preconditions:**
    - Repo connected; Test code change prepared.
- **Procedure:**
    - Push change; Monitor build/deploy to Cloud Run.
    - Run post-deploy tests.
- **Expected Results:**
    - Build succeeds; Deploys automatically; Tests pass.
- **Postconditions:** Build logs stored.

3. Functional Requirement Test Cases

TC-FR1 – Summarize a User-Provided Document

- **Objective:** Verify that the system can automatically generate accurate and concise summaries from uploaded academic documents.
- **Scope:** Tests the end-to-end summarization process including parsing, chunking, and summary generation.
- **Configuration:**
    - Backend: NVIDIA LLM via LangChain summarization chain
    - Interface: Gradio / Chat Web UI
    - File Type: PDF
- **Preconditions:**
    - User has successfully uploaded a valid PDF document.
    - Summarization service is active and connected to LLM endpoint.
- **Procedure:**
    - Upload a PDF file.
    - Click “Generate Summary”.
    - Observe the summarized text displayed.
    - Validate summary coherence and factual consistency.
- **Expected Results:**
    - Summary generated within 10 seconds.
    - Summary captures key ideas without errors.
- **Postconditions:** Summary saved in session memory for reference in subsequent

queries.

TC-FR2 – Extract and Display Document Metadata

- **Objective:** Ensure that metadata such as title, authors, and publication date are correctly extracted and displayed.
- **Scope:** Covers document parsing, metadata recognition, and display components.
- **Configuration:**
  - DocumentParser class using PyPDFLoader or equivalent
  - Metadata fields: title, authors, DOI, abstract
- **Preconditions:**
  - Document uploaded and readable.
- **Procedure:**
  - Upload a document and view metadata section.
  - Compare extracted metadata to the source PDF.
- **Expected Results:**
  - All metadata fields correctly populated.
  - Display formatted clearly in the UI.
- **Postconditions:** Metadata stored for retrieval and citation validation.

TC-FR3 – Identify and Highlight 3 Key Findings

- **Objective:** Validate that the system identifies and highlights the top three findings or conclusions.
- **Scope:** Tests LLM summarization refinement and output structuring.
- **Configuration:**
  - RSummarizer module with `key_findings=True` parameter
- **Preconditions:**
  - Summary generated successfully.
- **Procedure:**
  - Review generated summary output.
  - Check the “Key Findings” section for correctness.
- **Expected Results:**
  - Three major findings extracted and clearly highlighted.
  - Findings align with the paper’s results section.
- **Postconditions:** Highlighted results available for faculty validation.

TC-FR4 – Answer User Questions Based on the Document

- **Objective:** Confirm that the system provides fact-based answers to user queries grounded in uploaded documents.
- **Scope:** Tests integration between retrieval, augmentation, and LLM components.
- **Configuration:**
  - RAG pipeline with FAISS vector retrieval
  - LLMService for answer generation
- **Preconditions:**
  - Document indexed and embeddings created.
- **Procedure:**
  - Ask a factual question related to the uploaded paper.
  - Observe system response for correctness.
- **Expected Results:**

    - Answers derived from document content.
    - Includes in-text citation or reference snippet.
- **Postconditions:** Answer stored in KnowledgeBase for conversation continuity.

TC-FR5 – Retrieve Relevant Context for Each Question

- **Objective:** Ensure retrieval mechanism correctly returns relevant document chunks per user query.
- **Scope:** Tests context retrieval accuracy and semantic similarity search.
- **Configuration:**
    - FAISSVectorStore integrated with LangChain retriever
- **Preconditions:**
    - Document embeddings exist in vector store.
- **Procedure:**
    - Ask a contextual question (e.g., “What methods were used?”).
    - Observe the context text fetched prior to LLM response.
- **Expected Results:**
    - Retrieved text is contextually aligned.
    - No unrelated sections included.
- **Postconditions:** Retrieved context stored temporarily for query audit.

TC-FR6 – Track and Use Past Conversation History

- **Objective:** Validate that the chatbot retains memory of prior exchanges in ongoing sessions.
- **Scope:** Covers state persistence and conversational continuity.
- **Configuration:**
    - KnowledgeBase or session memory storage
- **Preconditions:**
    - Active chat session with previous Q&A logs.
- **Procedure:**
    - Ask follow-up questions referencing earlier responses.
    - Observe continuity in conversation flow.
- **Expected Results:**
    - System recalls past context correctly.
    - Responses remain consistent with previous turns.
- **Postconditions:** Session logs appended to persistent memory.

TC-FR7 – Maintain a Running Summary Across Updates

- **Objective:** Confirm that the system updates summaries dynamically as new content or questions are introduced.
- **Scope:** Tests incremental summarization and context merging.
- **Configuration:**
    - RSummarizer with DocumentSummaryBase model
- **Preconditions:**
    - Initial summary generated.
- **Procedure:**
    - Ask follow-up questions about new document areas.
    - Regenerate summary.
- **Expected Results:**

    - Running summary reflects new insights.
    - No redundancy or loss of prior context.
- **Postconditions:** Summary version history maintained for audit trail.

TC-FR8 – Present Open Questions from the Paper

- **Objective:** Ensure the system identifies open research questions or suggested future work in the document.
- **Scope:** Tests natural-language analysis and topic classification.
- **Configuration:**
    - PromptAugmentor with “future work” extraction mode
- **Preconditions:**
    - Full document loaded and indexed.
- **Procedure:**
    - Ask: “What future work is mentioned?”
    - Verify extracted statements.
- **Expected Results:**
    - All open questions extracted correctly.
    - Responses clearly attributed to paper sections.
- **Postconditions:** Extracted items saved for display in summary results.

TC-FR9 – Validate Document Answers with Citations

- **Objective:** Validate that each system-generated answer includes at least one citation or document reference.
- **Scope:** Tests post-processing and response validation pipeline.
- **Configuration:**
    - ResponseFormatter module with citation tagging enabled
- **Preconditions:**
    - User has submitted a Q&A query.
- **Procedure:**
    - Submit a question to the chatbot.
    - Review whether citations or section markers appear.
- **Expected Results:**
    - Every response includes traceable references.
    - Citation links functional in display output.
- **Postconditions:** Citation log appended to QA record.

TC-FR10 – Support Input Format in PDF

- **Objective:** Ensure only PDF input format is supported and properly handled.
- **Scope:** Covers file upload validation and format filtering.
- **Configuration:**
    - Frontend: File Upload Widget
    - Backend: DocumentParser class
- **Preconditions:**
    - User attempts to upload multiple file types.
- **Procedure:**
    - Upload .pdf, .docx, and .txt files sequentially.
    - Observe which files are accepted or rejected.
- **Expected Results:**

    - Only .pdf accepted.
    - Other formats rejected with clear error message.
- **Postconditions:** Uploaded PDF passed to parsing and summarization modules.

TC-FR11 – Validate Embedding Generation and Vector Storage

- **Objective:** Ensure that text chunks are correctly converted to embeddings and stored in the FAISS vector store for accurate retrieval.
- **Scope:** Tests the EmbeddingEngine and FAISSVectorStore components for semantic embedding quality.
- **Configuration:**
    - Embedding Model: NVIDIA nv-embed-v1; Vector Store: FAISS hosted on cloud bucket.
- **Preconditions:**
    - Document chunked and ready for embedding; Embedding endpoint active.
- **Procedure:**
    - Upload a sample PDF and trigger embedding.
    - Retrieve embeddings for a known chunk and compare similarity scores with expected similar/dissimilar texts.
- **Expected Results:**
    - Embeddings generated without errors; Similarity search returns relevant chunks with cosine similarity >0.8 for matches.
- **Postconditions:** Embeddings indexed in FAISS; logs record embedding latency and size.

TC-FR12 – Evaluate RAG Retrieval Precision and Recall

- **Objective:** Verify that the RAG pipeline retrieves relevant document chunks with high precision and recall for query augmentation.
- **Scope:** Tests retrieval accuracy in the VectorStoreManager and PromptAugmentor.
- **Configuration:**
    - Retriever: LangChain with FAISS; Metrics: Precision@K and Recall@K (K=5).
- **Preconditions:**
    - Document indexed; Sample queries with ground-truth relevant chunks prepared.
- **Procedure:**
    - Submit 10 test queries (e.g., "What methods were used?").
    - Calculate precision (relevant retrieved / retrieved) and recall (relevant retrieved / total relevant).
- **Expected Results:**
    - Precision and recall ≥85%; No irrelevant chunks in top results.
- **Postconditions:** Retrieval metrics logged for model tuning.

TC-FR13 – Test Multi-Turn Conversational Memory in RAG

- **Objective:** Validate that conversational history is used to refine RAG prompts across multiple queries.
- **Scope:** Tests KnowledgeBase and ChatPromptBuilder for context retention.
- **Configuration:**
    - Memory: FAISS-based ConvStore; Test: 5-turn conversation chain.
- **Preconditions:**
    - Initial query asked.

- **Procedure:**
  - Ask follow-up questions referencing prior responses (e.g., "Elaborate on that method").
  - Verify if context is augmented correctly.
- **Expected Results:**
  - Responses consistent with history; No context loss.
- **Postconditions:** Conversation stored for session reload.

4. Non-Functional Requirement Test Cases

TC-NFR1 – Response Time (≤ 3 Seconds on Average)

- **Objective:** Verify that the system provides responses to user queries within an average of 3 seconds under normal operating conditions.
- **Scope:** Measures end-to-end latency for document summarization and Q&A workflows.
- **Configuration:**
  - Environment: Google Cloud Run (Autoscaled)
  - Monitoring: Cloud Trace or Postman API timing
  - Dataset: 10 sample PDFs (5–10 pages each)
- **Preconditions:**
  - Model and retriever endpoints are active.
  - System idle baseline established.
- **Procedure:**
  - Submit 20 user queries across five sessions.
  - Measure total latency per request.
  - Compute mean response time.
- **Expected Results:**
  - Average response ≤ 3 seconds.
  - No individual query exceeds 5 seconds.
- **Postconditions:** Performance metrics stored for evaluation dashboard.

TC-NFR2 – Concurrent User Handling (≥ 50 Users)

- **Objective:** Ensure the system supports at least 50 concurrent active users without degradation or timeout.
- **Scope:** Validates backend concurrency and autoscaling capability.
- **Configuration:**
  - Load tool: Locust / JMeter
  - Deployment: Cloud Run (min instances = 1, max instances = 10)
- **Preconditions:**
  - Cloud Run autoscaling enabled.
  - Database connections pooled.
- **Procedure:**
  - Simulate 50–100 simultaneous chat sessions.
  - Observe latency, error rate, and resource metrics.
- **Expected Results:**
  - ≥ 95% of requests succeed.
  - Response time degradation ≤ 10%.
- **Postconditions:** Scalability data logged for capacity planning.

TC-NFR3 – Response Accuracy (≥ 90%)

- **Objective:** Confirm that system-generated answers maintain factual accuracy of ≥ 90% against source documents.
- **Scope:** Evaluates correctness of retrieved information and LLM output grounding.
- **Configuration:**
    - Validation: Faculty review set of 100 Q&A pairs
    - Metric: Accuracy = (correct answers / total answers) × 100
- **Preconditions:**
    - Documents indexed and citations enabled.
- **Procedure:**
    - Ask 100 domain questions on uploaded papers.
    - Faculty manually verify correctness.
- **Expected Results:**
    - Accuracy ≥ 90%.
    - No unsupported or hallucinated claims.
- **Postconditions:** Accuracy statistics appended to quality-assurance log.

TC-NFR4 – Preservation of Formatting and Structure

- **Objective:** Verify that retrieved information preserves the original document's structure, headings, and paragraph alignment.
- **Scope:** Tests response post-processing and display modules.
- **Configuration:**
    - Renderer: Markdown / HTML Formatter
    - Test Set: PDF documents with sectioned headings
- **Preconditions:**
    - PDF parsed and chunked.
- **Procedure:**
    - Retrieve key sections ("Methods," "Results").
    - Compare layout with original.
- **Expected Results:**
    - Hierarchical formatting retained.
    - Tables / bullets preserved in output.
- **Postconditions:** Formatted output cached for subsequent sessions.

TC-NFR5 – Accessibility for Non-Technical Users

- **Objective:** Validate that the user interface is intuitive and accessible to users with no technical background.
- **Scope:** Covers interface design, navigation simplicity, and error feedback.
- **Configuration:**
    - Platform: Web chat UI (Gradio / FastAPI frontend)
    - Evaluation Group: 10 non-technical testers
- **Preconditions:**
    - Test users briefed on system purpose.
- **Procedure:**
    - Observe users upload PDF and ask a question without assistance.
    - Record navigation errors and completion time.
- **Expected Results:**
    - 100% of participants complete task unaided.
    - No critical UI confusion or input error.
- **Postconditions:** Usability feedback collected for design iteration.

TC-NFR6 – Session-Level State Maintenance

- **Objective:** Confirm that the system maintains dialogue continuity and session context across user interactions.
- **Scope:** Validates persistence and retrieval of conversation state.
- **Configuration:**
  - Session store: Firestore / Redis
  - Timeout: > 15 minutes of inactivity
- **Preconditions:**
  - Multiple messages exchanged within one session.
- **Procedure:**
  - Ask a question, wait 10 minutes, then follow up.
  - Verify system recalls previous context.
- **Expected Results:**
  - Conversation context retained until explicit logout.
  - No context loss between turns.
- **Postconditions:** Session archived for audit and analysis.

TC-NFR7 – Query and Response Logging for Future Improvement

- **Objective:** Ensure all user queries and system responses are logged for model evaluation and continuous improvement.
- **Scope:** Covers logging mechanism, storage security, and retrievability.
- **Configuration:**
  - Logging Service: Google Cloud Logging + BigQuery sink
  - Data Fields: timestamp, user ID (hash), query, response ID
- **Preconditions:**
  - Logging policy enabled.
- **Procedure:**
  - Perform 5 Q&A sessions.
  - Verify that each interaction appears in the log.
- **Expected Results:**
  - 100% of events captured with no loss.
  - Logs accessible only to authorized admins.
- **Postconditions:** Logs stored securely for periodic analysis and model tuning.

TC-NFR8 – Test for Hallucinations in LLM Responses

- **Objective:** Ensure LLM-generated summaries and answers contain no unsupported or fabricated information beyond the document.
- **Scope:** Evaluates factual grounding in the LLMService and ResponseFormatter.
- **Configuration:**
  - LLM: NVIDIA mixtral-8x7b-instruct; Test Set: 20 queries with known document facts.
- **Preconditions:**
  - Document uploaded; RAG enabled.
- **Procedure:**
  - Ask queries outside document scope (e.g., "What is the author's email?").
  - Manually or automatically check responses against document for extraneous claims.
- **Expected Results:**

    - 0% hallucinated content; All claims traceable to citations.
- **Postconditions:** Hallucination logs stored for fine-tuning.

TC-NFR9 – Assess Model Robustness to Input Variations

- **Objective:** Confirm the system handles noisy, long, or varied inputs without degrading ML performance.
- **Scope:** Tests robustness in summarization and Q&A pipelines.
- **Configuration:**
    - Input Variations: PDFs with OCR errors, >50 pages, or non-English abstracts.
- **Preconditions:**
    - Varied test documents prepared.
- **Procedure:**
    - Upload varied PDFs and generate summaries/Q&A.
    - Measure accuracy drop compared to clean inputs.
- **Expected Results:**
    - Accuracy degradation <10%; System flags or handles errors gracefully.
- **Postconditions:** Robustness report generated.

TC-NFR10 – Evaluate Auto-Scaling Triggers and Resource Utilization

- **Objective:** Ensure Cloud Run auto-scales based on load and monitors CPU/memory efficiently.
- **Scope:** Tests scaling in production-like conditions.
- **Configuration:**
    - Cloud Run: Concurrency ≥50, autoscaling enabled; Tool: Locust for load.
- **Preconditions:**
    - Baseline metrics established.
- **Procedure:**
    - Ramp up to 100 concurrent requests.
    - Monitor instance count and resource usage.
- **Expected Results:**
    - Scales to multiple instances; CPU <80%, no throttling.
- **Postconditions:** Scaling metrics logged in Cloud Monitoring.

TC-NFR11 – Assess Cloud Security and Error Handling

- **Objective:** Confirm secure deployment (HTTPS, auth) and graceful handling of cloud errors (e.g., timeouts).
- **Scope:** Tests security and resilience in Cloud Run.
- **Configuration:**
    - Auth: OAuth; Error Simulation: High load or API failures.
- **Preconditions:**
    - Service deployed.
- **Procedure:**
    - Attempt unauthorized access; Simulate LLM endpoint failure.
    - Observe retries and user feedback.
- **Expected Results:**
    - Access denied for unauth; Errors handled with fallbacks (e.g., cached responses).
- **Postconditions:** Security audit logs updated.

5. Conclusion

This verification plan provides a structured framework of detailed test cases covering user, functional, and non-functional requirements of the Scientific Paper Analyzer System. The outlined tests validate all critical aspects of the platform, from document ingestion, summarization, and context-aware Q&A to scalability, performance, and session persistence, within a cloud-deployed environment. By rigorously evaluating accuracy, response time, usability, and reliability, the plan ensures that the system meets academic-grade standards for research and educational use. Successful completion of these test cases confirms system stability, readiness for production deployment, and establishes a foundation for iterative improvement and future AI-driven research extensions.