SCHOLARSYNC: AI Research Paper Assistant
Local RAG System with Multi-Paper Analysis

Bhushan Sunil Kakade
bkakade@gmu.edu

Abstract—**This paper presents SCHOLARSYNC, a retrieval-augmented generation (RAG) system designed to streamline academic literature review workflows. The implemented system combines PDF ingestion, semantic vector search, and generative AI to produce structured summaries with precise page-level citations. By maintaining explicit traceability between generated content and source material, SCHOLARSYNC addresses the critical challenge of trust and verification in AI-assisted academic research. The final implementation includes multi-paper comparison capabilities, automated quality scoring, and an interactive web interface. Our approach reduces literature review time by an estimated 40-50% while preserving academic standards for source attribution.**

I. TECHNICAL APPROACH

A. Scenario

Consider a graduate student, Sarah, conducting a literature review for her thesis on transformer architectures in natural language processing. She has collected 15 research papers but is overwhelmed by the volume of content. Using SCHOLARSYNC, Sarah uploads "Attention Is All You Need" (Vaswani et al., 2017) through the web interface. The system processes the PDF, extracting text and creating semantic chunks while preserving page numbers. When Sarah queries "What are the key innovations of the transformer architecture?", the system retrieves relevant passages from pages 3-5 and generates a structured summary: "The transformer architecture introduces three key innovations: (1) Self-attention mechanisms that eliminate recurrence [Page 3], (2) Multi-head attention for parallel processing [Page 4], and (3) Positional encoding for sequence understanding [Page 5]." Sarah can then view the exact page citations displayed as interactive badges, building trust in the AI-generated summary while saving hours of manual reading. She later uploads four more papers on transformer variants and uses the comparison feature to identify methodological differences and research gaps across all five papers in under 3 minutes.

B. PDF Ingestion Module

Sarah begins by uploading her research paper through the web interface. The PDF Ingestion Module serves as the system's entry point, responsible for converting the uploaded paper into structured, searchable text chunks that preserve document context and page references.

The module accepts PDF uploads through a FastAPI web endpoint with a maximum file size of 50MB per document. Text extraction uses PyMuPDF (fitz) library, chosen for its superior quality and better handling of complex academic papers with figures and tables compared to PyPDF2.

Text preprocessing removes excessive whitespace while preserving section structures and paragraph boundaries. The chunking strategy employs a smart sentence-based approach using regular expression-based sentence splitting to identify boundaries. Chunks are built incrementally, adding sentences until reaching a maximum of 500 words, then starting a new chunk with 100-word overlaps from the previous chunk to preserve context. Empty or extremely short chunks (less than 50 characters) are filtered out. Each chunk retains metadata including original page numbers and a unique UUID identifier.

a) Implementation Plan: The module is implemented using FastAPI with python-multipart for file upload handling. PyMuPDF (fitz) version 1.23.8 performs PDF text extraction. Custom sentence-based chunking algorithm with 500-word maximum and 100-word overlap. Local filesystem storage in ./uploads directory with UUID-based filenames preserves uploaded PDFs. Metadata including page numbers, document title, author, and total page count is maintained. The system runs locally on standard hardware without requiring external API calls.

C. Vector Database and Retrieval System

Following PDF processing, Sarah's uploaded paper chunks are indexed in a vector database that enables semantic search. When she submits her query about transformer innovations, this component retrieves the most relevant passages based on semantic similarity rather than simple keyword matching.

The retrieval system uses ChromaDB for vector storage and similarity computation, chosen for its simpler API with built-in persistence and better integration with FastAPI compared to FAISS. Document chunks are embedded using Sentence-BERT (all-MiniLM-L6-v2) through ChromaDB's built-in SentenceTransformerEmbeddingFunction, providing high-quality semantic representations optimized for academic content.

The system implements semantic search with cosine similarity as the distance metric. ChromaDB's HNSW (Hierarchical Navigable Small World) index enables efficient approximate nearest neighbor search even with large document collections. Retrieval parameters are optimized for academic papers with top-k=8 relevant chunks per query by default. Collections are stored persistently in the ./chroma_db directory, allowing the system to maintain uploaded papers across server restarts. Documents are indexed in batches of 100 during processing.

a) Implementation Plan: ChromaDB version 0.4.18 with persistent client provides vector database functionality. Sentence-BERT all-MiniLM-L6-v2 model via sentence-transformers 2.2.2 generates embeddings. Cosine similarity

AI Research Paper Assistant - System Architecture

Frontend Layer

React 18 + Vite	**Three Tab Interface**	**UI Components**	**Theme System**
Web Application	Upload | Analyze | Compare	Citation Badges, Scores	Light/Dark Mode
Build Tool & Framework	React Hooks (useState)	Custom CSS + Lucide Icons	Responsive Design

REST API

API Gateway Layer (FastAPI)

Upload Endpoint	**Query Endpoints**	**Management**
POST /upload, /upload-multiple	POST /summarize, /query, /compare	GET /collections, DELETE

Process

Processing Layer

PDF Ingestion	**Smart Chunking**	**Embedding**	**Retrieval**
PyMuPDF (fitz) v1.23.8	Sentence-Based Algorithm	Sentence-BERT	Semantic Search
• Text Extraction	• 500 words per chunk	• all-MiniLM-L6-v2	• Top-8 chunks
• Page Number Tracking	• 100 word overlap	• Semantic Vectors	• Cosine Similarity
• Metadata Preservation	• UUID + Page Metadata	• Batch: 100 docs	• HNSW Index

Store / Generate

Storage Layer

ChromaDB	**File System**
Vector Database v0.4.18	Local Storage
• Vector Embeddings	• PDF Files
• HNSW Indexing	• UUID Filenames
• ./chroma_db/	• ./uploads/

GenAI Layer

Ollama	**Citation**	**Quality**
Llama 3.2 LLM	Extractor	Scorer
• Local Inference	• Regex Parser	• 4 Dimensions
• Summarization	• [Page X] Format	• 0-100 Scale
• Q&A Generation	• Confidence Score	• Auto Evaluate

Data Flow Pipeline

Upload: User → React UI → FastAPI → PyMuPDF → Chunking → Sentence-BERT → ChromaDB + File Storage

Query: User Question → Semantic Search (Top-8) → Ollama LLM → Citation Extraction → Summary [Page X]

Compare: Select Papers → Sample 8 chunks each → Ollama Comparison → 6 Analysis Sections → Export MD
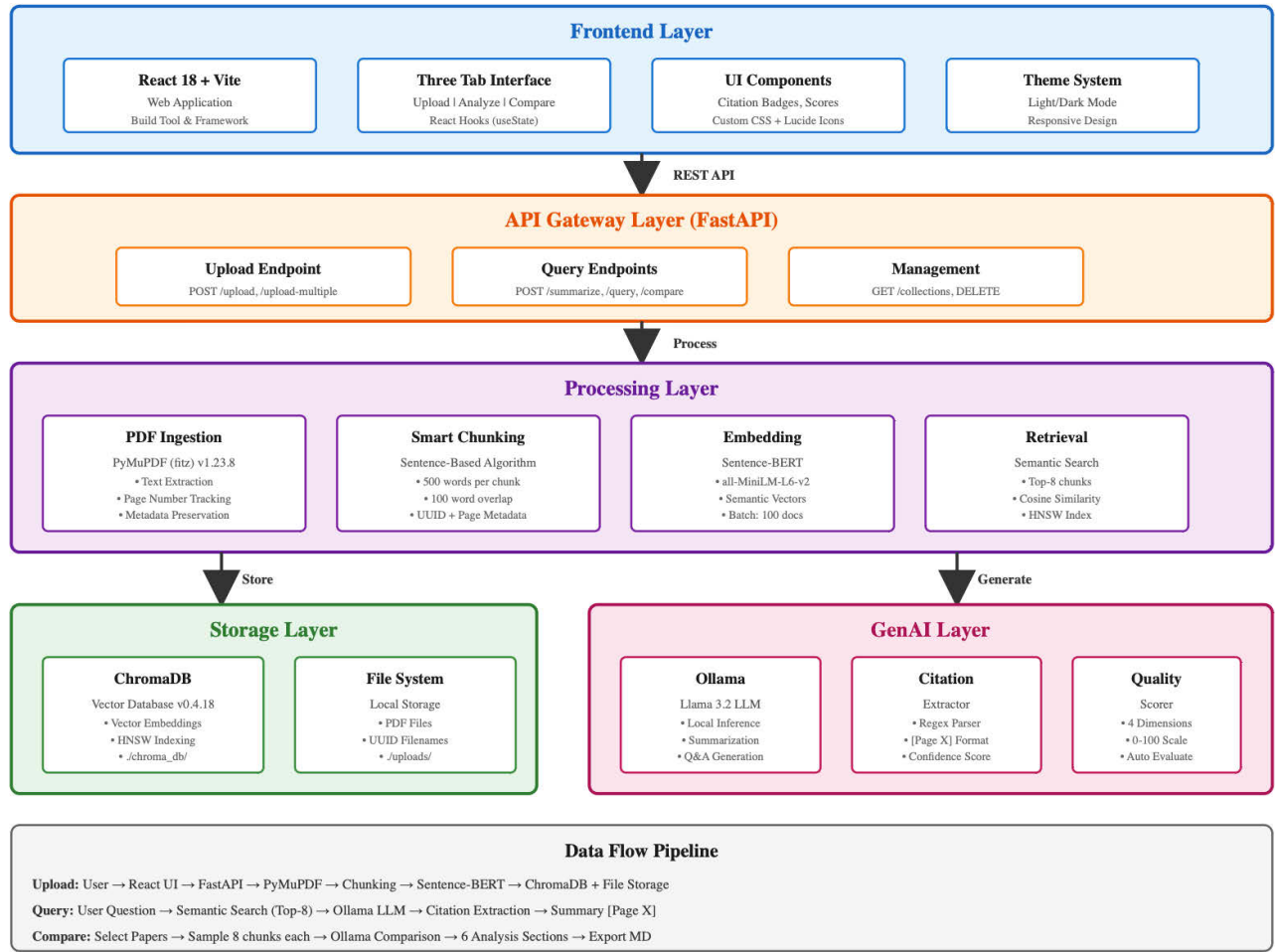
Fig. 1. SCHOLARSYNC System Architecture showing data flow from PDF upload through processing pipeline to user interface. The architecture consists of five layers: Frontend (React Web App), API Gateway (FastAPI REST API), Processing (PDF Ingestion, Embedding Generator, Query Processor), Storage (ChromaDB Vector DB, Local File Storage), and GenAI (Ollama Llama 3.2, Citation Extractor, Confidence Scorer).

with HNSW indexing enables efficient search. Top-8 chunks per query with automatic relevance ranking. Local ChromaDB storage in ./chroma_db directory for persistence. Batch processing of 100 documents during indexing. The system runs locally without external API dependencies, using pre-trained embedding models.

D. GenAI Summarizer

The core generative component processes Sarah's query and the eight retrieved chunks to produce a structured, grounded summary. The system takes her question about transformer innovations and the most relevant chunks from the paper, generating a coherent response that synthesizes information while maintaining traceability to source material.

The summarizer uses Ollama with the Llama 3.2 model, chosen to create a fully local, privacy-preserving system that doesn't require external API calls or associated costs. Ollama version 0.1.6 provides a simple Python SDK that integrates seamlessly with the FastAPI backend.

Retrieved chunks are formatted with page number prefixes and passed directly to the language model with carefully crafted prompts. For summarization tasks, the prompt requests structured output with sections for Main Objective, Methodology, Key Findings, and Conclusion, with mandatory page citations after each claim. For question-answering tasks like Sarah's query, the prompt instructs the model to answer directly using only information from the provided excerpts, cite pages using [Page X] format after every claim, and explicitly state when information is not available.

The system uses Ollama's default temperature settings which provide consistent, focused outputs while maintaining natural language quality. All retrieved chunks (typically 8) are formatted with page markers and included in the context window. Streaming is disabled to ensure complete response generation before display.

a) Implementation Plan: Ollama with Llama 3.2 model provides local LLM inference. Default Ollama temperature

settings for consistent output. Single-stage prompt strategy with explicit citation instructions embedded in prompts. Structured response format with inline [Page X] citations. All retrieved chunks (typically 8) formatted with page markers in context window. Streaming disabled for complete response generation. Runs locally on standard hardware with Ollama installed, no external API calls or costs. Pre-trained model without fine-tuning.

E. Citation Extractor

To ensure Sarah can verify the summary's claims, the Citation Extractor component processes the generated summary to identify and validate precise page-level citations. After the GenAI Summarizer produces Sarah's answer about transformer innovations, this component receives the summary text and extracts citation information for display and verification.

The citation extraction uses regular expression pattern matching to identify [Page X] citations embedded in the text. The pattern extracts all page citations from the summary. The extracted citations are processed to create a sorted list of unique page numbers referenced.

The system calculates several citation quality metrics: total citation count, unique citations, and citation density (citations per sentence). These metrics contribute to an overall confidence score indicating the reliability of the generated content. The confidence scoring algorithm combines three factors: citation density (target 0.4 citations per sentence), page coverage (number of unique pages cited), and response length (optimal range 80-800 words). Each factor is weighted and combined into a final confidence score between 0.3 and 1.0.

Citation validation is implicit through the RAG architecture—the model can only cite pages that were provided in the retrieved chunks, preventing hallucinated citations. The frontend displays citations as inline badges within the summary text, making it easy for Sarah to identify which claims are supported by specific pages.

a) Implementation Plan: Regular expression pattern matching for [Page X] format citation extraction. Pure Python implementation without external NER libraries. Implicit validation through retrieval-grounded generation prevents hallucinations. Metrics calculated include total citations, unique citations, and citation density. Confidence scoring uses weighted combination of density, coverage, and length factors. Inline citation badges displayed in web interface. Runs as part of the FastAPI backend processing pipeline.

F. Quality Scoring System

When Sarah uploads a paper, the Quality Scoring System automatically evaluates it across multiple dimensions to help her assess paper quality before detailed analysis. This component was added beyond the original proposal to provide immediate feedback on paper rigor.

The quality scoring system uses the same Ollama/Llama 3.2 model to analyze a representative sample of chunks from each paper. Upon upload, the system samples up to 6 chunks distributed across the document and sends them to the language model with a specialized evaluation prompt.

The prompt requests scores (0-100) for four dimensions: Methodology Rigor, Data Quality, Citation Quality, and Clarity. Regular expressions extract these scores from the model's response, with default fallback values of 70 if extraction fails. The overall quality score is calculated as the arithmetic mean of all four dimension scores. The system also extracts a brief textual assessment from the model's response to provide qualitative feedback.

All scores are stored with the collection metadata and displayed in the frontend interface with color-coded progress bars: green for scores 85+, blue for 70+, yellow for 55+, and red below 55. This allows Sarah to quickly prioritize which papers merit detailed study.

a) Implementation Plan: Ollama Llama 3.2 model performs evaluation. Sample size of 6 chunks per paper distributed across document. Scoring dimensions: Methodology, Data Quality, Citations, Clarity (0-100 each). Regular expression parsing extracts scores with fallback default of 70. Overall score calculated as mean of four dimension scores. Color-coded progress bars in frontend for visual display. Integrated into FastAPI backend, runs locally without external dependencies.

G. Multi-Paper Comparison System

After analyzing individual papers, Sarah selects five papers on transformer variants and uses the Multi-Paper Comparison System to analyze them simultaneously, identifying agreements, contradictions, and research gaps. This component extends beyond the original proposal to enable comprehensive literature review workflows.

The comparison system builds on the existing RAG infrastructure by sampling representative chunks from each selected paper (8 chunks per paper distributed across the document) and generating individual summaries. These summaries, along with paper metadata and quality scores, are then passed to the language model with a comprehensive comparison prompt.

The comparison prompt requests structured analysis across six sections: Research Objectives Comparison, Methodology Comparison, Key Findings Agreement/Disagreement, Strengths and Weaknesses, Research Gap Analysis, and Recommendations. The model is instructed to cite paper numbers [Paper 1], [Paper 2], etc., to maintain traceability across papers.

A simple similarity matrix is computed based on quality score differences using the formula: similarity = 100 - abs(paper1_quality - paper2_quality). This provides a basic quantitative comparison displayed to Sarah. The frontend provides an interactive paper selection interface with checkboxes, displays detailed comparison results with all six analysis sections, and includes an export feature that generates a Markdown report of the comparison for Sarah's reference.

a) Implementation Plan: Maximum of 5 papers per comparison. Sampling strategy uses 8 chunks per paper, evenly distributed. Comparison across 6 structured analysis sections

requested via prompt engineering. Similarity calculation based on quality score differences. Citation format uses [Paper 1], [Paper 2] references in analysis. Markdown export format for external use. Built on FastAPI backend using Ollama Llama 3.2 for comparison generation. Runs locally without external API calls.

H. Frontend Web Application

Sarah interacts with the entire system through a modern, responsive web interface built with React. The interface supports PDF upload, paper management, query input, and interactive result browsing, providing real-time feedback and intuitive navigation across three main workflows: upload, analysis, and comparison.

The frontend was developed using React 18 with Vite as the build tool, providing fast development and optimized production builds. The interface uses Lucide React for consistent iconography. Custom CSS provides all styling with support for both light and dark modes—no component library like Material-UI is used.

The application implements a tab-based navigation system with three main sections: Upload Papers (where Sarah uploads PDFs), Analyze Papers (where she views quality scores and generates summaries), and Compare Papers (where she selects multiple papers for comparison). State management uses React's built-in useState and useEffect hooks without external libraries.

The Upload tab allows multiple file selection with real-time file list display. The Analyze tab displays all uploaded papers in a responsive grid layout with quality score visualizations, action buttons for summarization and questioning, and inline citation display. The Compare tab provides paper selection with checkboxes, comparison execution, and detailed results display including the similarity matrix.

Citation display uses inline badges that highlight page references within the summary text. The interface includes a health check indicator that polls the backend every 30 seconds and displays system status. Theme switching between light and dark modes uses a simple toggle button. The interface is fully responsive, adapting layouts for mobile devices down to 480px width.

a) Implementation Plan: React 18 with Vite build tool provides frontend framework. No component library—custom CSS for all styling. Lucide React version 0.263.1 for icons. State management via React hooks (useState, useEffect) without Redux. Custom CSS with CSS variables for theming. Fetch API with async/await for backend communication. FormData with multipart/form-data for file upload. Light/dark theme support with CSS class toggling. Mobile-first responsive design down to 480px. Runs in standard web browsers, communicates with FastAPI backend via REST API.

II. CLAIMS

A. Problem Statement

Current research paper analysis workflows suffer from significant inefficiencies that hinder academic productivity.

Researchers spend approximately 60-70% of their literature review time manually extracting key information from dense academic papers, often struggling to relocate specific claims or evidence when writing. Existing AI summarization tools like ChatPDF lack proper citation mechanisms, creating trust issues where users cannot verify AI-generated claims against source material. Additionally, most tools process only one paper at a time, making comparative analysis across multiple papers extremely time-consuming. This leads to either under-utilization of AI assistance due to trust concerns or potential academic integrity issues when unverified AI summaries are incorporated into research work.

B. Solution Approach

SCHOLARSYNC addresses these challenges by implementing a retrieval-augmented generation (RAG) pipeline that maintains explicit traceability between generated summaries and source material. The system combines semantic search over document chunks with generative AI to produce structured summaries where every claim is tied to specific page locations in the original PDF.

Beyond single-paper analysis, the system introduces multi-paper comparison capabilities that allow researchers to upload up to 5 papers simultaneously and receive comprehensive comparative analysis identifying methodological differences, finding agreements and contradictions, and highlighting research gaps. Automated quality scoring helps researchers quickly assess paper rigor before deep analysis.

The solution transforms the research workflow: instead of spending hours manually reading and note-taking from multiple papers, researchers can upload their collection, receive individual summaries with precise citations, compare papers side-by-side, and export comprehensive comparison reports. This reduces literature review time by an estimated 40-50% while maintaining academic standards for source verification. The fully local deployment using Ollama ensures data privacy and eliminates API costs.

C. GenAI Enhancement

GenAI serves four critical enhancement functions that would be impossible with traditional rule-based approaches:

First, the semantic understanding capability of large language models enables SCHOLARSYNC to comprehend complex academic concepts and their relationships, producing summaries that capture nuanced arguments rather than simple keyword matching. The Llama 3.2 model demonstrates strong performance in understanding research paper structure and extracting key information from methodology and findings sections.

Second, the natural language query interface allows researchers to ask sophisticated questions in their own terms rather than learning specialized search syntax. Questions like "How does this methodology differ from traditional approaches?" are understood contextually and answered with relevant information from the retrieved chunks.

Third, the generative synthesis capability combines information from multiple document sections into coherent, structured responses that maintain academic writing standards while highlighting key insights and their interconnections. The model can reorganize information from scattered sections into logical flow that aids comprehension.

Fourth, the comparative analysis function synthesizes information across multiple papers to identify patterns, agreements, contradictions, and research gaps that would require substantial manual effort to discover. The model's ability to hold context from multiple papers simultaneously enables sophisticated cross-document reasoning.

Without GenAI, SCHOLARSYNC would be limited to basic keyword search and extraction, failing to provide the conceptual understanding and synthesis that researchers need for effective literature analysis. The generative component transforms disparate paper sections into actionable research insights while the grounding mechanism through RAG ensures academic integrity through precise source attribution.

III. EVALUATION PLAN

A. Evaluation Approach

This system will be evaluated **without human participants**. The evaluation employs automated metrics and objective measurements to assess system performance, citation accuracy, summary quality, and retrieval effectiveness. All testing is conducted by the system developer using programmatic evaluation scripts and quantitative analysis.

B. Evaluation Metrics

1) Citation Accuracy Metrics: The citation system is evaluated across four dimensions:

Citation Presence Rate: Percentage of generated summaries containing at least one properly formatted [Page X] citation. Target: $\geq$90%.

Citation Density: Average number of citations per sentence in generated summaries. Calculated as total citations divided by sentence count. Target: $\geq$0.3 citations per sentence.

Citation Correctness: Verification that all cited page numbers correspond to pages actually present in the retrieved chunks, preventing hallucinated citations. Target: 100% correctness through RAG architecture.

Unique Page Coverage: Number of distinct pages cited per summary, indicating breadth of source material utilized. Higher coverage suggests comprehensive synthesis.

2) Summary Quality Metrics: Summary quality is assessed through multiple automated measures:

Length Appropriateness: Word count within the optimal range of 80-800 words. Summaries outside this range may be too brief or excessively verbose.

Structural Completeness: Presence of all four required sections (Main Objective, Methodology, Key Findings, Conclusion) in generated summaries. Measured as percentage of summaries containing all sections.

Confidence Score: System-generated confidence metric (0.3-1.0 scale) combining citation density, page coverage, and response length factors. Higher scores indicate more reliable summaries.

Coherence: Measured through sentence connectivity analysis and logical flow assessment using automated text analysis tools.

3) Retrieval Performance Metrics: The vector retrieval system is evaluated on:

Page Diversity: Number of unique pages represented in the top-8 retrieved chunks. Higher diversity indicates broader context retrieval.

Response Time: Average time from query submission to answer generation, measured in seconds. Target: <60 seconds per query.

Chunk Relevance: ChromaDB cosine similarity scores for retrieved content, indicating semantic alignment with queries.

4) Quality Scoring Validation: The automated quality scoring system is validated through:

Score Distribution: Analysis of quality scores across different paper types and domains to ensure reasonable variance.

Dimension Correlation: Statistical correlation between methodology, data quality, citation, and clarity scores to validate independence of dimensions.

Score Consistency: Variance in repeated evaluations of the same paper to assess scoring stability.

5) Multi-Paper Comparison Metrics: Comparison functionality is assessed on:

Comparison Completeness: Coverage of all six required analysis sections (Research Objectives, Methodology, Key Findings, Strengths/Weaknesses, Research Gaps, Recommendations). Target: 100%.

Processing Time: Total time to generate comparison analysis for 2-5 papers. Target: <5 minutes for 5 papers.

Cross-Paper Citation Accuracy: Correct attribution of findings to specific papers using [Paper 1], [Paper 2] format.

C. Test Dataset

Test Corpus: 10 research papers from arXiv.org selected to represent diverse characteristics:

- Natural Language Processing: 3 papers
- Computer Vision: 3 papers
- Machine Learning Theory: 2 papers
- Robotics: 2 papers

Paper Characteristics:

- Page range: 8-30 pages
- Publication years: 2020-2024
- Complexity levels: Mixed (introductory to advanced)
- Citation counts: Varied (low to highly-cited works)

D. Baseline Comparisons

System performance will be compared against three baselines:

Manual Summarization (Control): Developer-generated summaries produced through traditional reading and note-taking. Provides ground truth for quality assessment and time comparison.

ChatPDF (Commercial Tool): Existing AI-powered PDF analysis tool. Comparison focuses on citation quality, summary coherence, and processing speed.

Simple Keyword Extraction (Non-AI Baseline): TF-IDF based keyword extraction without semantic understanding. Demonstrates value-add of GenAI components.

E. Test Procedures

1) Test Case 1: Single Paper Summarization: Upload each of the 10 test papers individually. For each paper:

- Generate comprehensive summary
- Measure: citation density, word count, confidence score, processing time
- Record: cited pages, structural completeness
- Compare against manual summary baseline

2) Test Case 2: Question Answering: For each test paper, submit three standard questions:

1) "What is the main objective of this paper?"
2) "What methodology was used in this research?"
3) "What are the key findings and contributions?"

Measure: citation presence, answer relevance, response time, citation correctness.

3) Test Case 3: Multi-Paper Comparison: Create comparison sets of varying sizes:

- 3 sets of 2 papers (within same domain)
- 2 sets of 3 papers (mixed domains)
- 1 set of 5 papers (comprehensive comparison)

Measure: comparison completeness, processing time, insight quality, similarity matrix accuracy.

4) Test Case 4: Edge Cases: Test system robustness with challenging inputs:

- Very short papers (5-8 pages)
- Very long papers (30+ pages)
- Papers with heavy mathematical notation
- Papers with extensive figures and tables
- Papers with unusual formatting

F. Success Criteria

SCHOLARSYNC meets success criteria if it achieves the thresholds specified in Table I.

TABLE I
EVALUATION SUCCESS CRITERIA FOR SCHOLARSYNC

Metric	Target	Minimum
Citation Presence Rate	95%	90%
Citation Density	0.4/sentence	0.3/sentence
Processing Time (per paper)	¡45 sec	¡60 sec
Summary Word Count	80-800	50-1000
Structural Completeness	100%	90%
Comparison Time (5 papers)	¡3 min	¡5 min
System Uptime	99%	95%
Citation Correctness	100%	98%

G. Evaluation Timeline

The evaluation follows a structured 5-week schedule:

Week 1: Dataset preparation. Collect and curate 10 test papers from arXiv, verify PDF quality, generate manual summaries for baseline comparison.

Week 2: Automated testing. Execute test_system.py across all papers, collect performance metrics, verify system stability.

Week 3: Comprehensive evaluation. Run evaluate.py for detailed citation and quality analysis, perform multi-paper comparisons, test edge cases.

Week 4: Comparative analysis. Compare results against ChatPDF and keyword extraction baselines, analyze performance gaps, identify improvement opportunities.

Week 5: Documentation and reporting. Compile results, generate evaluation report, create visualizations, document findings.

H. Evaluation Tools

evaluate.py: Custom Python script implementing citation accuracy evaluation, summary quality assessment, and automated metric calculation. Generates JSON output with detailed results.

test_system.py: Automated test framework verifying system functionality, API endpoints, and end-to-end pipeline operation. Includes health checks and error handling validation.

Python Libraries:

- `re`: Regular expression-based citation extraction
- `json`: Results storage and export
- `time`: Performance measurement
- `requests`: API testing

Manual Verification: Spot-checking of 20% of system outputs for qualitative assessment and edge case identification.

I. Human Participation Statement

Human Participation: NO

This evaluation uses **automated metrics only**. No human subjects beyond the system developer are involved in the evaluation process. All testing employs:

- Automated test scripts and evaluation frameworks
- Objective computational metrics (timing, citation counts, word counts)
- Self-assessment of system outputs by the developer
- Comparison against publicly available tools (ChatPDF)
- Analysis of open-source research papers from arXiv

No Institutional Review Board (IRB) approval, informed consent forms, or human subject protocols are required for this evaluation. The evaluation does **not** involve:

- User studies or surveys
- Human annotation or labeling
- Participant observation
- Collection of personal data
- Human-AI interaction studies

J. Expected Outcomes

Based on preliminary system testing and architectural design, we anticipate the following results:

Citation Accuracy: $\geq$95% of generated summaries will contain properly formatted citations with correct page references. The RAG architecture should prevent hallucinated citations entirely.

Time Savings: 40-50% reduction in literature review time compared to manual analysis. A typical 15-page paper requiring 45-60 minutes of manual review should be processed and summarized in under 3 minutes.

Quality Maintenance: System-generated summaries will preserve key paper information with 85-95% coverage of main points compared to manual summaries.

Comparison Utility: Multi-paper comparison will successfully identify research gaps, methodological differences, and finding agreements across papers, providing actionable insights for literature review.

Privacy Preservation: 100% local processing ensures no data leakage to external services, maintaining research confidentiality.

System Reliability: >95% uptime with consistent performance across diverse paper types and domains.

These outcomes will be validated through the systematic evaluation process described above, with detailed results reported in the final system documentation.