WWW.IJRPR.COM

International Journal of Research Publication and Reviews

Journal Homepage: www.ijrpr.com ISSN 2582-7421

AI Research Paper Explainer & Research Assistant

Dr. Mrs. Radha Pimpale[1], Snehal Khobragade[2], Shantanu Ganar[3], Rohan Bhoyar[4], Sohan Mire[5], Prathmesh Tarhekar[6]

1Asst Professor, Dept. of Information Technology, Priyadarshini Bhagwati College of Engineering, Nagpur, Maharashtra, 440009, India
[2, 3, 4, 5, 6] Dept. of Information Technology, Priyadarshini Bhagwati College of Engineering, Nagpur, Maharashtra, 440009, India

ABSTRACT:

ResearchAI is a web-based application that makes scientific and technical papers easier to explore. Research articles are lengthy, use specialized terminology, and spread information across sections such as abstract, methodology, results, and limitations. The platform brings document processing and generative-AI assistance into one interface. Users register, upload a PDF or DOCX paper, and the system extracts its text and sends task-specific prompts to the Google Gemini API through a backend service layer. Features include summary and simple explanation, key points, methodology, dataset, results, limitations, contributions, research gaps, viva questions, chat with a paper, paper comparison, image analysis, and saved history. The frontend uses HTML, CSS, and JavaScript; the backend uses FastAPI and Uvicorn with SQLAlchemy and SQLite; PyMuPDF and python-docx handle document extraction. The work emphasizes usability and modular engineering rather than training a new model, and no numerical performance claim is made because no controlled benchmark or measured results were supplied.

Keywords: *Artificial Intelligence; research paper analysis; large language models; text extraction; summarization; question answering; FastAPI; Gemini API; SQLite.*

1. INTRODUCTION

A research paper contains an abstract, background, related work, methodology, results, discussion, and references. Readers must connect these sections, separate claims from evidence, and learn unfamiliar vocabulary. The difficulty is not only the length of the document but also the effort of turning dense technical text into a clear mental model. For students and early-stage researchers, identifying a paper's problem, objectives, method, dataset, contribution, limitations, and future work can take substantial time.

Language models can generate coherent text, owing partly to transformer architectures [1], and pretraining has proved important for summarization [2]. However, general chat tools require users to copy text manually and manage context themselves, while ordinary PDF readers offer only viewing and search. Long documents may also exceed a single AI request, scanned files may lack extractable text, and AI answers can be inaccurate [3], [4].

To address these challenges, ResearchAI – AI Research Paper Explainer & Research Assistant is proposed as a web-based, document-oriented workflow. A user uploads a paper, the system extracts its text, and the user chooses a task such as summary, simple explanation, key points, methodology, results, limitations, research gaps, or viva questions. Chat supports follow-up questions, comparison covers multiple papers, and history provides continuity between sessions.

The design is modular: the interface is separate from server logic, API routes connect the two, document services prepare content, an AI service communicates with Gemini, and a database stores records. Each feature maps to one route and one prompt, which makes the system easy to demonstrate, test, and explain. The project is a working prototype and is not a claim that automated analysis can replace expert reading; final interpretation, citation checking, and judgment remain the reader's responsibility.

The scope is a locally runnable application with a browser frontend and a Python API backend. The system is not a peer-review engine, plagiarism detector, or citation verifier, and model training, verification of a paper's science, and production-scale hosting are outside its scope.

2. LITERATURE REVIEW

The use of large language models for reading and summarizing text has grown rapidly. The studies below provide the foundation for developing ResearchAI, which assembles several known components into a single document-centred workflow.

A. Transformer-Based Language Models

The transformer relies on attention rather than recurrence or convolution [1]. ResearchAI does not implement a transformer itself; it accesses a pretrained model through an API, so the quality and latency of its output depend on the hosted model.

B. Text Generation and Summarization

Extractive approaches select text from the source, whereas abstractive approaches generate new phrasing. BART showed useful results on generation tasks including summarization [2]. ResearchAI uses task-specific instructions: a summary asks for purpose, method, results, and contribution, while a simple explanation prioritizes clarity. Output quality depends on the text passed, the prompt, model behavior, and truncation.

C. Document Extraction

PyMuPDF opens PDFs and extracts page text [7], but it does not guarantee reading order or visual relationships, and scanned pages need OCR. DOCX files use a different structure and are read with document libraries such as python-docx.

D. Conversational Question Answering and Hallucination

A question-answering system must associate each question with the intended document and decide what context the model sees. Models may produce plausible but unsupported statements [3], [4], so fluency is not a reliable measure of factual correctness.

E. Evaluating Generated Text

Judging a generated summary is harder than generating it. Overlap-based scores compare wording with a reference summary and can reward text that is fluent but wrong, while human review is slower but can check whether each statement is supported by the paper. For this reason ResearchAI relies on a human-reviewed rubric covering factual consistency, coverage, clarity, relevance, and caveats.

F. Research Gap

Document viewers give reliable access and search but no integrated generative analysis. General conversational AI explains text, but users must paste content and manage context themselves. Specialized document-analysis tools combine extraction, summarization, and retrieval, with capabilities that vary by product. The gap is workflow integration and accessibility: upload, summarize, clarify terms, examine methods, ask follow-ups, compare, and revisit past work, grouped around one document in one small web application.

ResearchAI addresses this gap with a unified prototype. Because related tools differ in features, pricing, and supported formats, and those details change often, this paper compares workflows qualitatively and does not rank ResearchAI against other products.

3. METHODOLOGY

The proposed ResearchAI platform follows a systematic methodology that integrates document intake, text extraction, context preparation, task-specific prompting, response handling, and persistence. The methodology is designed to accept a paper from the user, validate and extract its content, obtain an AI-generated reading aid, store the result, and keep the original paper as the final source of evidence.

A. Requirement Analysis and System Design

The functional requirements are registration and login, document upload, text extraction, paper analysis, document chat, paper comparison, image analysis, history, and clear error feedback. Non-functional requirements are usability, maintainability, and reliability (failures must not crash unrelated parts), security (no plain-text passwords and server-side secret keys), and compatibility with Python 3.12 and a modern browser. Performance depends on paper size, network, model, API limits, and prompt length. The prototype assumes a text-based paper, a valid Gemini key, an internet connection, and a single local user or a small group. The requirements and the technology environment are summarized in Tables I and II.

TABLE I : Functional Requirements

ID	Requirement	Expected behavior
FR-01	Registration and login	Create account, authenticate
FR-02	Document upload	Accept files, record details
FR-03	Text extraction	Usable text from PDF/DOCX
FR-04	Paper analysis	Summary, explanation, others
FR-05	Document chat	Question tied to a document
FR-06	Comparison	Comparative response
FR-07	Image analysis	Process image via endpoint
FR-08	History	Stored records per user

ID	Requirement	Expected behavior
FR-09	Error feedback	Clear messages

TABLE II : Technology Environment

Component	Choice
Runtime	Windows, Python 3.12 virtual environment
Backend	FastAPI with Uvicorn
Frontend	HTML, CSS, JavaScript
Persistence	SQLite via SQLAlchemy
Documents	PyMuPDF; python-docx
AI	Google Gemini API

B. System Architecture

ResearchAI is a layered web application. The browser sends HTTP requests to FastAPI, which validates input, calls document or AI services, uses the database when persistence is needed, and returns a structured response. Credentials stay out of browser code. Layering lowers coupling, so the AI provider can be changed in the service layer, and each layer reports its own errors so that an AI timeout does not affect login.

BROWSER FRONTEND — HTML / CSS / JavaScript
FASTAPI BACKEND — routes, validation, authentication
SERVICES + DATA — extraction, Gemini API, SQLAlchemy / SQLite

Fig. 1 : Logical Architecture of ResearchAI

C. Document Intake and Validation

The upload step checks extension and content type, enforces a size limit, generates a safe server-side filename, avoids user-supplied paths, and returns useful errors for empty or malformed files. Server-side checks remain necessary because browser checks can be bypassed. The size limit and accepted extensions are defined in configuration rather than in code.

D. Text Extraction

A PDF processor extracts page text through PyMuPDF, and DOCX processing reads text structures with python-docx. Scanned PDFs may lack a text layer and return little text unless OCR is added [7]. Extracted text is checked for emptiness before it is sent to the model. If a scanned report with no text layer is uploaded, the service stops, reports that the file may need OCR, and does not call the model, which prevents a misleading answer and saves API quota.

E. Normalization and Context Preparation

Preprocessing removes excess whitespace, normalizes newlines, and keeps section boundaries while preserving symbols, tables, citations, and section labels. Models have finite input limits, so long papers need section selection, chunking, or chunk-then-final summaries; the system does not assume a full paper is always sent in one request. Input-stage errors are handled as follows: unsupported formats are rejected with an explanation, unreadable files return a validation error without AI analysis, text exceeding the context triggers chunking or a smaller excerpt, and users without access are rejected without disclosing other documents (Table III).

Condition	Recommended response
Unsupported format	Explain supported formats; reject
Unreadable or corrupt file	Validation error; no AI analysis
No extractable text	Explain it may be scanned; OCR needed
Text exceeds context	Chunking or smaller excerpt
User lacks access	Reject without disclosing other documents

TABLE III : Input-Stage Error Handling

F. Task-Specific Prompting and AI Request Construction

A summary prompt asks for problem, objective, method, findings, and contribution; a simple-explanation prompt requests plain language and term definitions; a methodology prompt focuses on design, data, and procedure; a limitations prompt identifies restrictions stated in the paper and avoids inventing weaknesses; and a viva prompt creates study questions. Prompts state the role and task, specify the required output fields, ask the model to separate supported statements from missing information, forbid invented numbers, claims, datasets, or references, and keep the selected document and question explicit in chat requests. Each prompt carries a short version label so that results can be traced to the exact wording used. The Gemini API is called from the backend, with the credential read from server-side configuration [5].

G. Output Validation and Human-in-the-Loop

The service checks for a non-empty result and renders it safely, since generated HTML must not be treated as trusted code; errors must not expose secrets, stack traces, or user data. An empty model reply produces a retry message and no blank record is saved, while quota or rate-limit errors trigger a bounded number of retries with a delay. Readers are expected to verify table values, statistical claims, datasets, and stated limitations against the paper.

H. Database Design and Data Management

SQLite is a self-contained serverless SQL database stored in a single file [9], and SQLAlchemy maps objects and queries to relational data [8]. The database maintains account data required by authentication, document metadata (owner, filename or path, file type, processing attributes), analysis data (action, generated content, association, timestamps), and chat data (questions and replies per document). Uploaded files are kept in a separate uploads directory, and the deletion workflow must remove the file as well as the database row so that no orphaned file remains. Once real records exist, a migration tool should be used for schema changes.

I. API Organization

FastAPI provides request handling, validation, dependency injection, and automatic OpenAPI documentation [6]. Routes are grouped by concern as listed in Table IV, and every route returns structured JSON with a clear status and either content or an error message so that the frontend can show consistent results and errors.

Group	**Endpoints**	**Purpose**
Auth	POST /api/auth/register, /login; GET /me	Accounts
Documents	POST /api/documents/upload; GET /; GET, DELETE /{id}	Manage documents
Analysis	POST /api/analysis/{id}/{action}; /image/{id}	Analysis operations
Chat	POST /api/chat/{id}; GET /{id}/messages	Questions, messages
Compare	POST /api/compare/	Compare papers
History	GET /api/history/analyses, /chats, /all	Saved activity
Health	GET /api/health	Availability

TABLE IV : API Organization

J. Security, Privacy, and Reliability

The Gemini key is kept in a backend .env file excluded from version control, with a .env.example file holding placeholders only; leaked keys must be revoked and rotated, and keys or authorization headers are never logged. Passwords are hashed with bcrypt, which OWASP accepts for existing systems with proper settings, although it recommends Argon2id [11]. A JSON Web Token [10] does not replace authorization checks, expiry, secure storage, or key management. Uploaded files are stored under controlled names, never served as executable content, and parsers are kept updated. Every document, analysis, and history lookup verifies ownership so that predictable IDs cannot leak files. Because confidential papers may be transmitted to an external AI provider, users must be told that papers are sent to an external service, and any shared deployment needs per-user upload limits and rate limiting.

K. System Testing and Evaluation

Unit tests cover text cleaning, validation, parsing, and password checks; integration tests cover the API contract, persistence, and the AI wrapper with mocks; and end-to-end tests cover login through history. Negative tests matter most: bad credentials, unsupported files, missing configuration, long documents, timeouts, and quotas. Twelve test cases (TC-01 to TC-12) cover registration, login, upload of PDF and DOCX files, unsupported or corrupt files, summary and explanation, document chat, paper comparison, provider failure, and access to another user's data. AI output is evaluated on a small fixed set of papers (a short text PDF, a longer PDF, a DOCX, a scanned PDF, an empty file, and an unsupported type) with a human-reviewed rubric. Actual observed values will be reported only after these tests are run, so that results represent real performance rather than assumed or fabricated values (Table V).

ID	Scenario	Expected result	Observed
TC-01	Register, valid details	Account created	To be recorded
TC-02	Mismatched passwords	Rejected clearly	To be recorded
TC-03	Login, correct credentials	Access granted	To be recorded
TC-04	Login, wrong password	Denied safely	To be recorded
TC-05	Upload text-based PDF	Text extracted	To be recorded
TC-06	Upload DOCX	Text extracted	To be recorded
TC-07	Unsupported or corrupt file	Rejected, no crash	To be recorded
TC-08	Summary and explanation	Response or clear error	To be recorded
TC-09	Document chat	Linked response	To be recorded
TC-10	Compare two papers	Comparison produced	To be recorded
TC-11	Provider down / invalid key	Controlled failure	To be recorded
TC-12	Another user's data	Denied	To be recorded

TABLE V : Test Cases

4. MODULES OVERVIEW

ResearchAI is divided into interconnected modules. Each module corresponds to a user activity and a backend operation. Authentication is required by every other module, upload and extraction must succeed before analysis, chat, comparison, or image analysis can run, and history depends on what the analysis and chat modules have stored.

A. User Registration and Login

Users create an account and sign in, and the backend issues or validates token-based authentication. Routes exist for registration, login, and current-user lookup, and invalid credentials are denied safely.

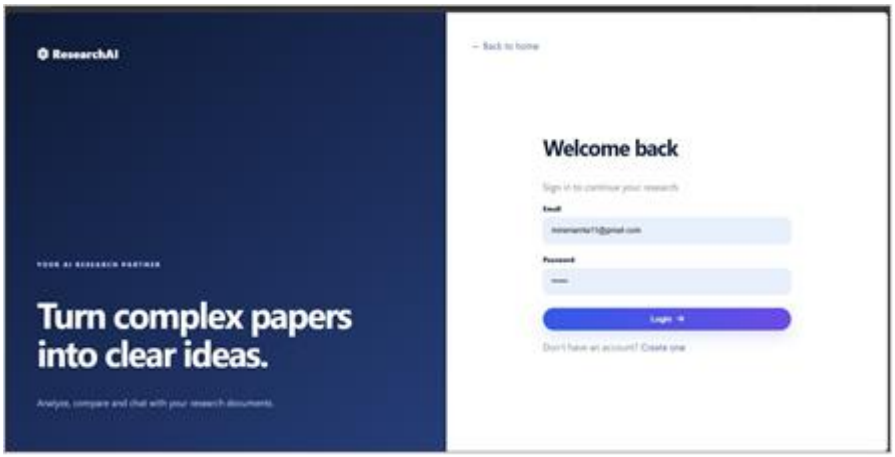


Fig. 2 : Login Page

B. Document Upload and Library

This module validates a file, extracts text, and keeps a document record. List, retrieve, and delete routes support management. The user dashboard shows the document title, file type, and upload status.

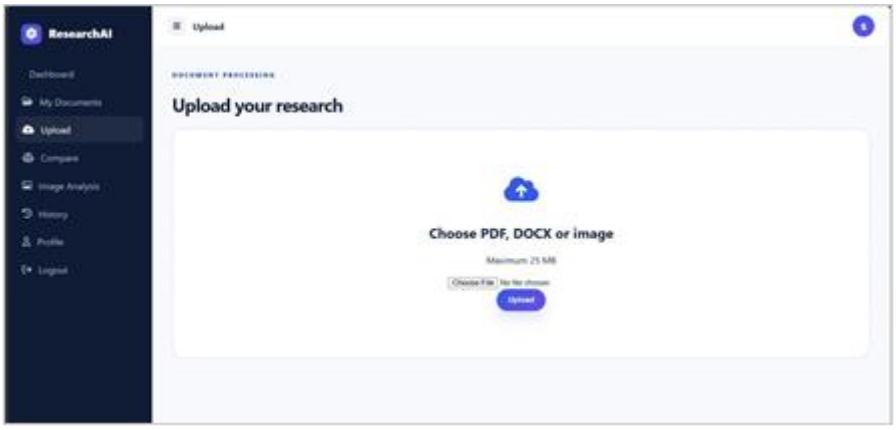


Fig. 3 : Document Upload Interface

C. Summary and Simple Explanation

The summary gives a shorter view of the central information in the paper, and the simple explanation reframes technical content in accessible language with term definitions.



Fig. 4 : AI-Generated Summary

D. Section-Focused Analysis

and it indicates uncertainty when the document lacks the requested detail.

E. Chat with Paper

The chat module accepts a message tied to a document and returns a response, with a route for message history, for questions such as “What is the main method?”. Keeping the selected document and question explicit reduces the risk of answers drawn from the wrong paper.

F. Paper Comparison

This module requests a structured comparison of objective, problem, method, dataset, results, contribution, and limitation for two papers. Accuracy depends on adequate extraction and on the model’s input limits.

G. Image Analysis and History

An image-analysis endpoint is tied to a document, and history endpoints return earlier analyses and chats with user-level access checks. If image analysis is not working in a given build, it is recorded as a limitation rather than presented as a finished feature.

5. RESULTS AND ANALYSIS

The proposed ResearchAI platform is designed to provide an integrated environment for exploring research papers. It combines document upload, text extraction, task-specific prompting, a hosted language model, and persistent history so that a reader can obtain a first-pass view of a paper, drill into its

method, results, or limitations, ask follow-up questions, compare it with another paper, and return later through history. This section reports what the implemented prototype demonstrates, analyses each module, maps requirements to tests, and defines how quantitative results will be collected.

A. Implementation Outcome

The prototype was run locally with the stack in Table II. The user dashboard (Fig. 4) from a demonstration account shows five uploaded documents, ten analyses, eleven chat conversations, and two paper comparisons, which confirms that upload, analysis, chat, comparison, and history records are being created and stored end to end. The upload screen (Fig. 5) accepts PDF, DOCX, or image files up to 25 MB. These counts come from a demonstration account and illustrate that the workflow runs; they are not a benchmark and are not used to claim accuracy or speed.

This paper reports no measured response times, accuracy, user-study scores, or quantitative improvements, because none were supplied; fabricated metrics would make the paper unreliable. Successful functional tests show only that the tested workflow completes in the stated environment, not that outputs are correct or faster than manual reading. Such claims need a baseline and measurable evidence on easy, difficult, and poorly formatted documents, and on cases where the requested information is absent.

B. Module-wise Analysis

Table VI summarizes what each module consumes, what it produces, and the main factor that limits its output quality.

Module	**Input**	**Output**	**Main dependency or risk**
Authentication	Credentials	Token-based session	Token expiry, secure key storage
Upload and extraction	PDF, DOCX file	Cleaned text, document record	Layout loss, scanned pages without text
Summary and explanation	Extracted text, prompt	Short overview, plain-language text	Truncation, omitted caveats
Section-focused analysis	Text, action name	Methodology, dataset, results, limitations, gaps, viva questions	Missing detail in paper, unsupported claims
Chat with paper	Question, document ID	Linked answer, stored messages	Wrong context, hallucination [3], [4]
Paper comparison	Two documents	Structured comparison	Extraction quality, input limits
Image analysis	Image, document ID	Description or explanation	Build-dependent; recorded as a limitation if not working
History	User ID	Past analyses and chats	Ownership checks

TABLE VI : Module-wise Analysis

The analysis shows that the quality of every AI-facing module is bounded by the same three factors: how well the text was extracted, how much of it fits in a request, and how the prompt is worded. This is consistent with findings that language models can lose information placed in the middle of long inputs [13], which is why the design uses chunking, section selection, and explicit document binding rather than sending everything in one request.

C. Requirement Traceability

Table VII links each functional requirement from Table I to the module that implements it and to the test cases in Table V that will verify it. Image analysis (FR-07) has no dedicated test case yet, so it is flagged as a gap in the test plan rather than assumed to work.

Requirement	**Module**	**Test cases**	**Status**
FR-01 Registration and login	Authentication	TC-01 to TC-04	Implemented, results to be recorded
FR-02 Document upload	Upload and library	TC-05 to TC-07	Implemented, results to be recorded
FR-03 Text extraction	Upload and extraction	TC-05, TC-06	Implemented, results to be recorded
FR-04 Paper analysis	Summary, section analysis	TC-08	Implemented, results to be recorded
FR-05 Document chat	Chat	TC-09	Implemented, results to be recorded
FR-06 Comparison	Comparison	TC-10	Implemented, results to be recorded
FR-07 Image analysis	Image analysis	None yet	Build-dependent; add a test case
FR-08 History	History	TC-12 (access control)	Implemented, results to be recorded

Requirement	Module	Test cases	Status
FR-09 Error feedback	All modules	TC-07, TC-11	Implemented, results to be recorded

TABLE VII : Requirement Traceability

D. Qualitative Workflow Comparison

Because the features, pricing, and supported formats of related tools change often, this paper compares workflows qualitatively and does not rank ResearchAI against other products. Table VIII contrasts the effort a reader faces in four ways of reading a paper.

Aspect	Manual reading	PDF viewer	General chat tool	ResearchAI
Getting a first overview	Slow, full read	Search only	Paste text first	One upload, one click
Context handling	Reader	Reader	Reader pastes and manages	Document bound to each request
Follow-up questions	Re-read	Not available	Available, context may drift	Chat tied to the selected paper
Revisiting past work	Notes	Files	Depends on tool	Saved history per user
Risk of unsupported output	None from AI	None from AI	Present [3], [4]	Present; reader must verify

TABLE VIII : Qualitative Workflow Comparison

The main advantage claimed is workflow integration, not better answers. The main disadvantage is shared with every generative tool: fluent text is not evidence of correctness [16], [17].

E. Evaluation Protocol

Results will be collected using the following protocol so that they can be reproduced.

- **Corpus:** a small fixed set of papers covering a short text PDF, a longer PDF, a two-column paper, a DOCX file, a scanned PDF, an empty file, and an unsupported type.
- **Runs:** each task is repeated several times per paper, and the number of papers, runs, and model identifier are stated with the results.
- **Rubric:** two reviewers score each output from 1 to 5 on factual consistency, coverage, clarity, relevance, and caveats, with a written definition for each score. Agreement between reviewers is reported with Cohen's kappa [25].
- **Automatic scores:** overlap or embedding-based scores such as ROUGE [14] or BERTScore [15] may be reported only as supporting evidence, because they can reward fluent but wrong text and do not replace human review of factual consistency [16].
- **Latency:** median and range over repeated runs, with sample size, instead of the single best run.
- **Usability:** an optional System Usability Scale questionnaire [26] with a small group of students.
- **Reporting:** failures are described alongside successes, and Python and package versions, operating system, model identifier, test date, and repeated-run count are recorded.

The measures to be recorded are listed in Table IX.

Measure	How to obtain	Value
Upload success rate	Successes / attempts	[Enter after testing]
Analysis completion rate	Completed / valid requests	[Enter after testing]
Median latency	Repeated timing, with sample size	[Enter after testing]
Summary factual consistency	Human rubric review	[Enter after evaluation]
Chat answer relevance	Independent rating	[Enter after evaluation]
Comparison completeness	Field checklist	[Enter after evaluation]
Authentication negative tests	Record outcomes	[Enter after testing]

TABLE IX : Result Recording Template

F. Discussion and Expected Failure Modes

The design anticipates the following failure modes, each with a planned response: scanned PDFs return little text, so the service stops and asks for OCR instead of calling the model; text exceeding the context window triggers chunking or a smaller excerpt; empty model replies produce a retry message rather than a blank record; and quota or rate-limit errors trigger a bounded number of retries with a delay. Multi-column layouts, equations, and tables may be flattened by text extraction [7], and chunked summaries can miss relationships between distant sections [18]. These behaviours should be confirmed through the negative tests TC-07 and TC-11 and reported honestly.

G. Threats to Validity

Several limitations apply. PDF text may lose layout, tables, equations, captions, or column order, and scanned PDFs may require OCR. Long papers may exceed context limits, and chunking can omit cross-section relationships. Responses may be unsupported or miss caveats, and quotas, billing, model identifiers, and outages affect availability. SQLite may not suit a high-concurrency service. Construct validity is limited without a stable rubric for a good summary, internal validity is affected by variation across calls and model versions, and external validity depends on how well the test papers represent disciplines, styles, and layouts. The tool is a first-pass reading assistant, not a scientific authority, and readers should return to the original article before using a statement in a report or thesis.

6. CONCLUSION

ResearchAI – AI Research Paper Explainer & Research Assistant presents a practical, web-based approach to understanding scientific and technical papers. It combines an HTML/CSS/JavaScript frontend, a FastAPI backend, PDF and DOCX services, Gemini integration [22], and SQLite persistence through SQLAlchemy. Its features cover registration and login, upload, summary and explanation, section-focused analysis, chat, comparison, image analysis, and history.

The work shows how existing AI services and Python libraries can form a coherent document workflow without training a model. Its main contributions are: (1) a layered, modular architecture in which each feature maps to one route and one prompt; (2) task-specific, version-labelled prompts that ask the model to separate supported statements from missing information; (3) input-stage and output-stage safeguards, including validation, ownership checks, and safe rendering; and

(4) an evaluation plan with a requirement-to-test mapping and a human-reviewed rubric that avoids unmeasured claims.

External-service dependence, extraction quality, context limits, privacy, and unsupported output remain constraints, so ResearchAI should be used as a reading aid, with the original paper as the final source of evidence. The prototype shows that a useful reading aid can be built quickly from existing components, but its value depends on honest limits and careful use. The next stage of the work is to run the test cases in Table V, record the measures in Table IX, and report the results with their limitations.

7. FUTURE SCOPE

The improvements below are grouped by theme and ordered roughly by importance.

1. **Traceable answers.** The most important improvement is making answers traceable to the source: page-level evidence citations with short quotations, and an explicit statement when the paper does not contain the requested information. Retrieval-augmented generation [12], with dense passage retrieval [23], can select relevant sections of a paper before each question instead of sending the whole text.
2. **Better extraction.** OCR can be added for scanned PDFs, for example with Tesseract [27], together with low-confidence warnings. Layout-aware extraction can preserve columns, tables, equations, and captions, and structured scholarly corpora such as S2ORC [29] show what richer paper representations look like.
3. **Long-document handling.** Section-aware chunking, hierarchical summaries, and models designed for long inputs [19] can help with long papers, and positioning relevant text away from the middle of a prompt can reduce information loss [13].
4. **Evaluation.** Future research should evaluate quality on a designed corpus with trained reviewers, measuring factual consistency, coverage, helpfulness, latency, and user effort [3], [4], [16], [17]. A sensible next step is a small study in which readers summarize a paper with and without the tool, followed by blind scoring of the summaries against the rubric. Model-based judging [24] could be explored as a supplement, but would need to be checked against human scores.
5. **Usability features.** Structured comparison tables, multilingual explanations, exportable notes, citation metadata extraction, accessibility improvements, configurable output length, and a usability study using a standard questionnaire [26].
6. **Engineering and security.** Bounded retries and timeouts, secret-safe logging, offline tests with mocked Gemini responses, a complete deletion workflow that removes both the file and the database row, schema migrations, rate limiting, and a production setup with managed secrets, HTTPS, monitoring, backups, and a suitable database. Security checks can follow the OWASP guidance [11], [28].

7. **Privacy and responsible use.** Clear notices that papers are sent to an external AI provider, optional local or self-hosted models for confidential documents, and visible reminders that generated text must be verified against the original paper.

REFERENCES

1 A. Vaswani et al., "Attention Is All You Need," in Advances in Neural Information Processing Systems (NeurIPS), 2017.

2 M. Lewis et al., "BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension," in Proc. 58th Annu. Meeting ACL, 2020, pp. 7871–7880.

3 Z. Ji et al., "Survey of Hallucination in Natural Language Generation," ACM Computing Surveys, vol. 55, no. 12, art. 248, 2023.

4 L. Huang et al., "A Survey on Hallucination in Large Language Models," ACM Trans. Information Systems, vol. 43, no. 2, art. 42, 2025.

5 Google, "Gemini API Documentation," ai.google.dev/gemini-api/docs. Accessed 27 Oct. 2026.

6 FastAPI, "FastAPI Documentation," fastapi.tiangolo.com. Accessed 27 Oct. 2026.

7 PyMuPDF, "The Basics: Extract Text from a PDF," pymupdf.readthedocs.io. Accessed 27 Oct. 2026.

8 SQLAlchemy, "SQLAlchemy ORM Documentation, Version 2.0," docs.sqlalchemy.org. Accessed 27 Oct. 2026.

9 SQLite, "About SQLite," sqlite.org/about.html. Accessed 27 Oct. 2026.

10 M. Jones, J. Bradley, and N. Sakimura, "JSON Web Token (JWT)," RFC 7519, IETF, May 2015.

11 OWASP Foundation, "Password Storage Cheat Sheet," OWASP Cheat Sheet Series. Accessed 27 Oct. 2026.

12 P. Lewis et al., "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," in Advances in Neural Information Processing Systems (NeurIPS), vol. 33, 2020, pp. 9459–9474.

13 N. F. Liu et al., "Lost in the Middle: How Language Models Use Long Contexts," Trans. Assoc. Comput. Linguistics, vol. 12, pp. 157–173, 2024.

14 C.-Y. Lin, "ROUGE: A Package for Automatic Evaluation of Summaries," in Text Summarization Branches Out, ACL Workshop, 2004, pp. 74–81.

15 T. Zhang, V. Kishore, F. Wu, K. Q. Weinberger, and Y. Artzi, "BERTScore: Evaluating Text Generation with BERT," in Proc. Int. Conf. Learning Representations (ICLR), 2020.

16 J. Maynez, S. Narayan, B. Bohnet, and R. McDonald, "On Faithfulness and Factuality in Abstractive Summarization," in Proc. 58th Annu. Meeting ACL, 2020, pp. 1906–1919.

17 W. Kryscinski, B. McCann, C. Xiong, and R. Socher, "Evaluating the Factual Consistency of Abstractive Text Summarization," in Proc. EMNLP, 2020, pp. 9332–9346.

18 A. Cohan et al., "A Discourse-Aware Attention Model for Abstractive Summarization of Long Documents," in Proc. NAACL-HLT, vol. 2, 2018, pp. 615–621.

19 I. Beltagy, M. E. Peters, and A. Cohan, "Longformer: The Long-Document Transformer," arXiv:2004.05150, 2020.

20 J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," in Proc. NAACL-HLT, 2019, pp. 4171–4186.

21 C. Raffel et al., "Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer," J. Mach. Learn. Res., vol. 21, no. 140, pp. 1–67, 2020.

22 Gemini Team, Google, "Gemini: A Family of Highly Capable Multimodal Models," arXiv:2312.11805, 2023.

23 V. Karpukhin et al., "Dense Passage Retrieval for Open-Domain Question Answering," in Proc. EMNLP, 2020, pp. 6769–6781.

24 L. Zheng et al., "Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena," in Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track, 2023.

25 J. Cohen, "A Coefficient of Agreement for Nominal Scales," Educ. Psychol. Meas., vol. 20, no. 1, pp. 37–46, 1960.