Indian Journal of Computer Science and Technology
https://www.doi.org/10.59256/indjcst.20260502096
Volume 5, Issue2 (May-August 2026), PP: 885-893.
www.indjcst.com
ISSN No: 2583-5300

An AI Based Approach for Research Paper Summarization Using Deep Learning

Santhosh Kumar C[1], Yogaraj S[2], Yogeshwaran G[3], Rishivanth D[4], D. Praveen Kumar[5]
[1,2,3,4] *Department of Artificial Intelligence and Data Science, Gnanamani College of Technology (Autonomous), Namakkal, Tamil Nadu, India.*
[5]*Assistant Professor, Department of Artificial Intelligence and Data Science, Gnanamani College of Technology (Autonomous), Namakkal, Tamil Nadu, India.*

To Cite this Article: *Santhosh Kumar C[1], Yogaraj S[2], Yogeshwaran G[3], Rishivanth D[4], D. Praveen Kumar[5], "An AI Based Approach for Research Paper Summarization Using Deep Learning", Indian Journal of Computer Science and Technology, Volume 05, Issue 02 (May-August 2026), PP: 885-893.*

Abstract: *The growing number of scientific publications requires tools that can automatically convert raw PDF research papers into structured, searchable knowledge. This paper introduces the Research AI Pipeline, a five-layer automated system for analysing academic PDF documents from start to finish. Layer 1 ingests PDFs through a FastAPI gateway that streams data in real time. Layer 2 extracts text using a combination of pdfplumber, PyMuPDF and pytesseract to create a normalised JSON schema that includes sections, tables, figures and numerical data. Layer 3 conducts a thorough quality audit with thirty-two checks across six validation groups, producing a severity score on a one-hundred-point scale. Layer 4 employs local large language model reasoning using DeepSeek-V3 via Ollama, with prompts tailored to maintain numerical accuracy in six types of summarised sections and a final synthesis. Layer 5 organises the output and supports an interactive QA interface that streams responses based on the extracted content. Tested on fifty research papers across various fields, the system achieves an F1-score of 0.89 for section boundary detection, 0.91 for figure caption matching and 88 percent accuracy in factual QA, all while operating locally without relying on cloud APIs. The average processing time is 87 seconds per paper.*

Keywords: *Research paper summarization; Deep learning; Large language model; PDF extraction; Audit validation; Question answering*

I. INTRODUCTION

The volume of scientific publications indexed globally now exceeds 200 million papers, with an estimated 4 million new articles appearing each year. In fields such as artificial intelligence this rate is even higher — arXiv alone receives upward of 300 machine learning papers per day. The structural consequence of this growth is a widening gap between knowledge production and knowledge consumption. Researchers spend 40 to 60 minutes reading and annotating a single paper, and synthesizing a literature survey of 50 papers can consume multiple weeks of focused work.

Existing tools address fragments of this problem in isolation. GROBID parses PDFs for bibliographic metadata but produces no downstream reasoning or QA capabilities. Commercial tools such as ChatPDF and Semantic Scholar AI provide conversational QA over documents but operate exclusively through cloud APIs, creating data privacy concerns for unpublished research and per-query cost dependencies that restrict high-volume use.

The Research AI Pipeline emerges from this gap. It is a five-layer, locally deployable, open-source system that handles the complete lifecycle of a research paper — from raw PDF bytes to interactive question answering — without routing any document content to external servers. The architecture was designed around three non-negotiable constraints: local execution for data privacy, structured audit output for trustworthy extraction, and a streaming user interface for responsive user experience.

1.1 Problem Statement

Given a collection of research papers in PDF format, this work addresses four coupled problems:

1. Design and implement an automated extraction pipeline that recovers structured academic content — sections, tables, figures and numerical data — from both digitally-born and scanned documents, handling the diverse layout conventions of IEEE, ACM, Elsevier, Springer and arXiv preprint formats.
2. Build a systematic audit framework that quantifies extraction quality through parameterized checks, providing severity-graded findings that enable informed decisions about whether extracted content is reliable enough for downstream reasoning.
3. Integrate a locally-deployed large language model that generates section-by-section academic summaries with strict numerical preservation, without hallucinating, rounding or omitting precise metric values, model names or dataset identifiers from the source text.
4. Expose the synthesized knowledge through a session-persistent interactive QA interface that streams responses grounded in actual paper content, without any external cloud API dependency.

1.2 Objectives

The following objectives drive the design and implementation of this system:

5. Develop a hybrid text extraction engine combining pdfplumber, PyMuPDF and pytesseract, with a FigureSchema dataclass that captures bounding boxes, captions, axis labels, legend items and semantic insights per figure.
6. Design a normalised JSON output schema that serves as a contract between all five layers, ensuring predictable data structure regardless of source document format.
7. Implement a 32-check audit engine organised across six validation groups with a 0-100 quality score, where CRITICAL findings deduct 15 points each and WARNING findings deduct 5 points each.
8. Integrate deepseek-v3.1 via the Ollama OpenAI-compatible /v1/chat/completions endpoint using a six-chunk, strict-rules prompting strategy at temperature 0.05 to maximise numerical fidelity in academic summaries.
9. Build a FastAPI backend with asynchronous pipeline execution, SSE-based real-time log streaming and relational persistence for sessions, messages and job outputs.
10. Create a React-Vite frontend with a three-panel layout — sidebar, main content area and right panel — using a silver-dark CSS variable design system.
11. Validate the complete pipeline across 50 research papers and demonstrate measurable, quantified improvements over baseline extraction approaches.

1.3 Scope of the Work

In scope: English-language academic papers in PDF format up to 100 pages; IEEE, ACM, Elsevier, Springer and arXiv preprint layout conventions; both digitally-born PDFs and scanned or image-heavy documents; documents containing text, tables, figures and mathematical notation; local hardware deployment with Ollama-served LLM inference; and single-paper processing with session-based multi-turn QA.

Out of scope: multilingual document processing; cross-paper bibliographic graph construction; simultaneous batch processing of multiple papers; integration with external academic databases such as Semantic Scholar or PubMed; fine-tuned extraction models, since a heuristic approach is used in Layer 2; and cloud LLM API integration, which is excluded by design as the system is local-first.

II.LITERATURE REVIEW

2.1 PDF Text Extraction Approaches

Klampfl and Kern proposed an unsupervised machine learning approach to body text and table-of-contents extraction from scientific articles, achieving 78% precision on the SectLabel benchmark using geometric heuristics derived from font size, line spacing and column alignment patterns. The approach struggles with multi-column IEEE layouts, where column boundary detection errors cause text flow inversions. GROBID applies Conditional Random Fields trained on labelled academic data and achieves precision above 0.90 on reference string parsing and author name extraction; however, its table detection capability is limited to lattice-structured tables and performs poorly on borderless tables common in modern machine learning papers.

pdfplumber, built on pdfminer, provides character-level bounding box access that enables rule-based layout reconstruction. PyMuPDF approaches PDF parsing from the rendering engine side, enabling both text extraction and page rasterization to images. This dual capability is the reason the Research AI Pipeline uses both: pdfplumber for primary extraction, and PyMuPDF for the OCR fallback path and figure bounding box detection. Tesseract 5, powered by an LSTM recognition model, achieves character error rates below 1% on cleanly scanned documents, and OpenCV preprocessing improves Tesseract accuracy on low-contrast or skewed scans through adaptive thresholding and deskewing. LayoutLM and LayoutLMv3 incorporate spatial position embeddings for joint layout-and-text understanding and achieve state-of-the-art results on form understanding benchmarks; however, these models require substantial GPU resources disproportionate to the extraction subtask in a local-first pipeline, motivating the heuristic approach retained in Layer 2.

2.2 LLM-Based Research Paper Summarization

Brown et al. demonstrated that GPT-3 achieves strong few-shot performance on summarization tasks without fine-tuning. SciBERT showed that pre-training on scientific corpora — 1.14 million papers from Semantic Scholar — improves performance on downstream scientific NLP tasks by 3 to 5 F1 points over BERT-base.

The critical challenge in academic summarization is numerical faithfulness. Maynez et al. showed that 70% of errors in abstractive summarization involve unfaithful facts such as hallucinated numbers, wrong model names or incorrectly attributed results. This finding directly motivates the strict-rules prompt block used in Layer 4 of this work, which mandates exact reproduction of every numerical value, percentage and model name present in the source section text.

DeepSeek-V3 achieves competitive performance with GPT-4o on coding and reasoning benchmarks while remaining deployable on consumer-grade GPU hardware. The variant served via Ollama provides the reasoning capability needed for academic synthesis tasks while maintaining the local-deployment constraint fundamental to this system. The chunk-based processing strategy in Layer 4 — processing abstract, introduction, methods, results-text, results-data and conclusion as six independent calls before synthesis — addresses context window limits and enables section-type-specific prompt engineering that generic full-paper prompts cannot achieve.

2.3 Automated Audit and Validation Systems

Data quality validation in document processing pipelines has parallels in data engineering. The Great Expectations framework provides programmatic assertions over tabular data such as column types, value ranges and null counts, but lacks semantic understanding of document structure. SHACL provides RDF-based schema validation but requires knowledge graph representations

not produced by PDF parsers. Mariani et al. proposed consistency checking for biomedical entity extraction, cross-validating numerical claims in text against table values and achieving 0.84 precision on a biomedical corpus; this work is the closest antecedent to the audit engine presented here. The Research AI Pipeline extends this principle to a six-group, 32-check audit framework with a quantified 0-100 scoring model — a design absent from existing open-source PDF extraction tools.

2.4 Research Question Answering Systems

BioASQ established benchmarks for biomedical QA with factoid, list, summary and yes/no question types. PubMedQA targets research question answering with yes/no/maybe responses backed by abstract justification. The COVID-19 CORD-19 challenges accelerated development of retrieval-augmented generation approaches for open-domain scientific QA. This work implements single-paper QA with full-context construction rather than retrieval, capping the QA context at 14,000 characters to remain within the effective window of the model.

2.5 Gaps in Existing Work

Three gaps in the existing literature motivate this work. First, no open-source system combines PDF extraction, quality auditing, local LLM reasoning and interactive QA in a single deployable pipeline. Second, no existing extraction system provides quantified quality scores with severity-graded findings; outputs are delivered without confidence metadata, silently propagating extraction errors downstream. Third, no published system uses locally-deployed open-source LLMs for research paper synthesis without cloud API dependencies, addressing the privacy and cost concerns of academic and industrial research environments.

III.MATERIAL AND METHODS

3.1 Existing System

Existing systems for research paper analysis typically focus on isolated components of the overall problem. Tools such as GROBID provide reliable metadata extraction but lack capabilities for summarization or interactive querying, while commercial platforms offer advanced features but rely heavily on cloud-based infrastructure. These systems exhibit incomplete coverage of the analysis pipeline, a lack of transparency in processing, dependence on external APIs, and limited support for numerical validation. Table no 1 compares the capabilities, limitations and cost of representative tools.

Tool	Capabilities	Limitations	Cost
GROBID	PDF parsing, bibliographic field extraction	No LLM reasoning, no QA interface, limited to digital PDFs	Free / Self-hosted
Semantic Scholar AI	Paper QA, citation graph recommendations	Cloud-only, no audit metrics, no local deployment	Free tier limited
ChatPDF	PDF upload, question answering	No structured extraction, no audit, no local	Paid SaaS
Tool	Capabilities	Limitations	Cost
		deployment, privacy concerns	
Elicit.ai	Research paper summarization	Cloud-only, limited to abstract summaries, no full paper analysis	Paid SaaS
SciSpace / Typeset	Paper explanation, citation tools	Cloud-only, no offline capability, limited audit	Freemium
Manual Analysis	Complete human understanding	40-60 minutes per paper, subject to bias and inconsistency	Researcher time

Table no 1: Comparison of existing research paper analysis tools.

3.2 Proposed System

The proposed system, the Research AI Pipeline, introduces a comprehensive multi-layer architecture that addresses the limitations of existing approaches. The system consists of five interconnected layers: an API gateway and user interface; a text and structure extraction layer; an audit and validation engine; a local LLM reasoning layer; and an output formatting and QA interface. Each layer performs a specific function, ensuring modularity, scalability and ease of maintenance.

3.3 Feasibility Analysis

Technical feasibility. The system leverages open-source technologies such as FastAPI, React, pdfplumber and Ollama, ensuring that all components are implementable with available tools.

Economic feasibility. The use of local deployment eliminates recurring API costs, making the system cost-effective for academic institutions.

Operational feasibility. The user interface is designed to be intuitive, enabling users with minimal technical expertise to interact with the system.

3.4 System Requirements

The system was developed and evaluated on the configuration specified in Table no 2. The Python dependencies and their pinned versions are listed in Table no 3.

Component	Specification
CPU	Intel Core i7 12th Gen or AMD Ryzen 7 5000 Series or above (8+ cores recommended)
RAM	Minimum 32 GB DDR4 (64 GB recommended for large paper corpora)
GPU	NVIDIA GPU with 24 GB VRAM (RTX 4090 or dual RTX 3090) for DeepSeek local inference
Storage	500 GB SSD (NVMe preferred for model loading performance)
OS	Ubuntu 22.04 LTS or Windows 11 with WSL2 (Linux preferred for Ollama stability)
Python	Python 3.10 or above
Node.js	Node.js 18 LTS or above (for Vite React frontend build)
Ollama	Ollama 0.3.0 or above with DeepSeek model pulled
Browser	Chrome 120+, Firefox 120+ or Edge 120+ for SSE streaming support

Table no 2: Hardware and software requirements specification.

Library	Version	Purpose
fastapi	0.115.0	REST API framework and SSE streaming
uvicorn[standard]	0.30.6	ASGI server for FastAPI
pdfplumber	0.11.4	Primary PDF text and table extraction
pymupdf	1.24.10	Page rasterization and fallback extraction
pytesseract	0.3.13	OCR for scanned or image-heavy PDFs
pandas	2.2.2	Tabular data handling
opencv-python	4.10.0.84	Image preprocessing for OCR pipeline
Pillow	10.4.0	Image format handling
python-dotenv	1.0.1	Environment variable management
Library	Version	Purpose
aiofiles	24.1.0	Async file operations

Table no 3: Python library dependencies with versions.

3.5 System Architecture

The Research AI Pipeline is designed as a strictly layered architecture in which each stage performs a well-defined transformation on the data before passing it forward. This design enforces separation of concerns, improves maintainability and allows independent testing of each module. The pipeline follows a linear progression: PDF input, extraction, audit, reasoning, and finally output with QA. Each layer refines the representation of the document: the raw PDF becomes semi-structured text; semi-structured text becomes validated structured data; structured data becomes interpreted knowledge; and knowledge becomes an interactive queryable system. Unlike monolithic systems, this architecture ensures that failure in one layer does not propagate silently — intermediate outputs can be inspected, validated and corrected. Table no 4 summarises the responsibilities and technology stack of each layer.

Layer	Name	Function	Technology
L1	API Gateway & UI	PDF upload, session management, SSE progress streaming	FastAPI, React, Vite
L2	Text Extraction	Hybrid PDF parsing; section, table, figure and number extraction; normalised JSON	pdfplumber, PyMuPDF, pytesseract, pandas
L3	Audit & Validation	32 quality checks, severity scoring, grade assignment, audit report generation	Python stdlib, statistics
L4	LLM Reasoning	Chunk summarization and synthesis using local DeepSeek model via Ollama	Ollama, DeepSeek-V3, requests
L5	Output & QA	Final JSON formatting, QA context construction, chat API session streaming	FastAPI, MySQL, requests

Table no 4: Five-layer architecture summary.

3.6 Layer 1 — API Gateway and User Interface

Layer 1 acts as the system entry point and is responsible for handling user interaction and orchestrating backend processing. Its core responsibilities are to accept PDF uploads, generate unique job identifiers, trigger asynchronous processing, stream progress updates to the frontend and maintain session persistence. The backend is implemented using FastAPI, chosen for its asynchronous capabilities and high performance. When a user uploads a PDF the system creates a background task, ensuring that the API remains responsive. Progress tracking is achieved using Server-Sent Events, where the frontend continuously receives updates without polling overhead. The frontend, built using React, provides an upload interface, real-time progress visualisation and chat-based QA

interaction. Instead of blocking execution until processing completes, the system adopts a non-blocking streaming model, improving scalability and user experience.

3.7 Layer 2 — Text and Structure Extraction

Layer 2 transforms raw PDF content into a structured representation. No single tool can reliably extract all PDF formats, and therefore a hybrid approach is adopted. pdfplumber acts as the primary extraction engine, providing character-level bounding box data that enables reconstruction of reading order. PyMuPDF is used for rendering pages as images and supports high-resolution rasterization. pytesseract provides OCR and is activated when text extraction fails, handling scanned or image-based PDFs.

Each page is analysed for text density, and if text is insufficient the OCR fallback is triggered. Sections are identified using font size variation, bold text detection and pattern matching over numbered headings. All extracted data is normalised into a JSON schema comprising sections, paragraphs, tables, figures and numerical values. The hybrid model ensures robustness across both digital and scanned documents — something most single-method systems fail to achieve.

3.8 Layer 3 — Audit and Validation Engine

This layer introduces a capability often missing in similar systems: quantified reliability. Its purpose is to evaluate the correctness and completeness of extracted data before it is passed to the LLM. The system performs 32 validation checks across six categories: structural completeness, section integrity, numerical consistency, cross-reference validation, table correctness and figure-caption alignment. Findings are graded as CRITICAL for a major failure, WARNING for a partial issue, INFO for a minor observation and PASS where no issue is detected.

The scoring model begins at 100, deducting 15 points per CRITICAL finding and 5 points per WARNING finding. The final output is a score in the range 0-100, a grade from A to F and a detailed audit report. Without validation, LLMs may amplify extraction errors; this layer ensures that only trustworthy input reaches the reasoning stage. The severity weights and grade thresholds are given in Table no 5.

Severity	Score Penalty	Grade Range	Status
CRITICAL	-15 per finding	—	Do Not Proceed
WARNING	-5 per finding	—	Review Recommended
INFO	0	—	Informational
PASS	0	—	Verified
—	90-100	A	EXCELLENT
—	75-89	B	GOOD
—	60-74	C	FAIR
—	40-59	D	POOR
—	0-39	F	CRITICAL — DO NOT PROCEED

Table no 5: Severity levels, scoring weights and grade thresholds.

3.9 Layer 4 — Local LLM Reasoning

This layer performs semantic interpretation and summarization using a locally hosted large language model, DeepSeek-V3 served through the Ollama local runtime. Instead of feeding the entire document at once, the system uses chunk-based reasoning over six sections: abstract, introduction, methodology, results (text), results (data) and conclusion. Each chunk is processed independently, followed by a final synthesis stage.

The prompt design enforces numerical fidelity, avoids hallucination, preserves original meaning and maintains an academic tone; a representative constraint is the instruction not to alter numerical values and to retain the original value when uncertain. Chunking reduces context overload and significantly improves factual accuracy compared with naive full-document summarization. The request parameters used for the reasoning layer are listed in Table no 6.

Parameter	Value / Description
Ollama Endpoint	/v1/chat/completions (OpenAI-compatible)
Model	deepseek-v3.1:671b-cloud (configurable via OLLAMA_MODEL env)
Temperature	0.05 (near-deterministic for factual accuracy)
Top-p	0.9
Chunk Token Limit	900 tokens per section summary
Synthesis Token Limit	3072 tokens for final report
Request Timeout	600 seconds (large model inference)
Stream	False (collect full response before returning)

Table no 6: Ollama API request parameters for the reasoning layer

3.10 Layer 5 — Output Formatter and QA Interface

This layer converts processed data into user-consumable formats, producing structured JSON, section-wise summaries and

a final synthesized report. The QA system builds context from the extracted and validated data, supports multi-turn conversations and streams responses in real time. The QA system is context-grounded, meaning it answers only on the basis of document content rather than external knowledge, which reduces hallucination.

3.11 Database Design

The system uses a MySQL relational database for persistence, accessed through PyMySQL. The core tables are chat_sessions, messages and job_outputs, with one session mapping to multiple messages and one job mapping to one processed document. The schema is summarised in Table no 7.

Table	Column	Description
chat_sessions	id (PK)	UUID session identifier
chat_sessions	name	User-provided session name
chat_sessions	filename	Original PDF filename
chat_sessions	created_at	Timestamp of session creation
chat_sessions	job_id (FK)	Reference to job_outputs.job_id
messages	id (PK)	UUID message identifier
messages	session_id (FK)	Reference to chat_sessions.id
messages	role	user or assistant
messages	content	Message text content
messages	created_at	Timestamp of message
job_outputs	job_id (PK)	UUID pipeline job identifier
job_outputs	output_json	Serialized Layer 5 output JSON
job_outputs	output_path	Filesystem path to output JSON file
job_outputs	report_path	Filesystem path to audit report markdown
job_outputs	stem	Base filename stem for output files

Table no 7: Database schema — sessions, messages and job outputs.

3.12 Implementation

The system is developed using a modular full-stack approach. The backend uses Python 3.10 or above with FastAPI served by Uvicorn; the frontend uses React with Vite; the extraction stack uses pdfplumber, PyMuPDF, pytesseract and OpenCV; and the LLM runtime is Ollama. The backend is structured into API routes, an extraction engine, an audit engine, a reasoning engine and an output formatter, each operating independently and communicating via structured data.

The extraction layer performs page-by-page processing with conditional OCR fallback, section detection and JSON schema generation, with special care taken to maintain reading order and structural hierarchy. The audit engine implements rule-based validation functions, a scoring algorithm and a report generator, each check being a modular function that enables easy extension. The reasoning layer integrates with the Ollama API through prompt templates, a chunk processing pipeline and final synthesis logic, with temperature kept low to ensure deterministic outputs. The frontend comprises React components, API integration, SSE event handling and a chat interface, with focus placed on responsiveness and real-time updates.

3.13 Testing Methodology

Testing is structured into unit testing, integration testing, performance testing and user acceptance testing. Unit testing validates individual modules independently. The extraction module was tested using digitally generated academic papers, multi-column IEEE formatted papers and scanned documents with low text density, verifying text extraction accuracy, detection of section headings, correct identification of tables and figures, and OCR fallback activation. The audit engine was tested by injecting controlled inconsistencies — missing section headings, incorrect numerical values, mismatched figure-caption pairs and incomplete table structures — and correctly identified anomalies with appropriate severity levels. API endpoints were tested for PDF upload, job status retrieval and QA response generation, with error handling returning appropriate status codes for invalid inputs. Representative unit test cases for the audit engine are given in Table no 8.

Test ID	Test Case	Expected Result	Status
UT-A01	Document with missing metadata.total_pages field	CRITICAL A2	PASS
UT-A02	Document with all metadata fields populated correctly	PASS A2	PASS
UT-B01	Section with page_start > page_end (e.g. 10 > 5)	CRITICAL B2	PASS
UT-B02	Section with text shorter than 20 characters	WARNING B3	PASS
UT-C01	Number entry missing value field	WARNING C1	PASS
UT-C02	Value 999.99 repeated 8 times — suspicious duplicate	INFO C2	PASS
UT-D01	Table section_ref pointing to nonexistent section_id	WARNING D1	PASS

UT-E01	Table with 3 headers but rows of length 2	WARNING E3	PASS
Test ID	Test Case	Expected Result	Status
UT-F01	Figure with missing caption string	WARNING F1	PASS
UT-F02	Figure semantic_insight of 5 chars (below 10 threshold)	WARNING F2	PASS
UT-SC01	Score with 0 CRITICAL, 0 WARNING findings	Score=100, Grade=A	PASS
UT-SC02	Score with 2 CRITICAL, 3 WARNING findings	Score=55, Grade=D	PASS

Table no 8: Unit test cases for audit engine checks.

Integration testing evaluated the complete pipeline using real-world research papers across the full workflow: upload, extraction, audit validation, summary generation and QA interaction. Data was correctly passed between layers without loss, audit results influenced reasoning quality, and QA responses were grounded in extracted content, with no major failures observed in cross-layer communication. Consistency between extracted data and generated summaries was verified for matching numerical values, preservation of section hierarchy and alignment between extracted figures and descriptions.

IV.RESULT

The system was evaluated using a dataset of approximately fifty research papers collected from multiple domains including artificial intelligence, computer science and engineering. The dataset comprises multi-column IEEE formatted documents, mixed content including text, tables and figures, variation in formatting styles across publishers, and both digitally generated and scanned PDFs. This diversity ensures that the evaluation reflects real-world variability rather than controlled experimental conditions. Table no 9 consolidates the principal quantitative results.

Evaluation Metric	Result	Interpretation
Section boundary detection (F1-score)	0.89	High accuracy across diverse layouts
Figure-caption matching (F1-score)	0.91	Strong contextual association
Factual question answering (accuracy)	88%	Reliable grounded responses
Average audit score	Above 80 (Grade A/B) for the majority of documents	Extraction quality verified
Average processing time per paper	87 seconds	Acceptable for practical use
Evaluation dataset size	50 research papers	Multi-domain, multi-publisher

Table no 9: Consolidated evaluation results of the Research AI Pipeline.

4.1 Extraction Performance

The system achieves an F1-score of 0.89 for section boundary detection, demonstrating strong performance in identifying section boundaries across diverse document layouts. The use of font-based heuristics combined with structural pattern recognition enables accurate segmentation of content. The minor inaccuracies observed are primarily due to unconventional formatting styles or inconsistent heading structures in certain documents; however, the performance remains significantly higher than basic rule-based extraction methods.

Figure-caption matching achieves an F1-score of 0.91, and the system effectively associates figures with their corresponding captions, which is critical for maintaining contextual understanding. The hybrid extraction approach plays a key role in achieving this performance, although challenges arise in documents where captions are embedded within paragraphs or span multiple lines without clear separation.

4.2 Audit Engine Evaluation

The audit engine produces scores on a scale of 0 to 100 representing extraction quality. The majority of documents scored above 80, corresponding to Grade A or B, while lower scores were associated with scanned or poorly formatted PDFs. A strong correlation was observed between audit scores and downstream reasoning quality: documents with higher audit scores resulted in more accurate summaries, better QA responses and reduced hallucination in LLM outputs. This validates the importance of the audit layer as a quality control mechanism within the pipeline.

4.3 LLM Reasoning Performance

The system generates section-wise summaries followed by a final synthesis. Summaries are structured and coherent, key contributions of papers are preserved, and numerical values are retained accurately. A major evaluation criterion was the preservation of numerical data, and the results show high consistency between source and generated summaries with minimaldistortion of experimental results. The use of constrained prompt engineering significantly reduces the common issue of numerical hallucination in LLMs.

4.4 Question Answering Performance

The question answering interface achieves 88% accuracy, measured on the correctness of responses to factual queries derived from the document. Responses are context-aware and relevant, generation of unsupported claims is minimal, and the system handles

multi-turn queries. The QA system benefits from structured context construction and grounded responses based on extracted data, which ensures that the system behaves more like a document assistant than a generic chatbot.

4.5 Performance Efficiency

The average processing time is approximately 87 seconds per document. Within this budget, extraction consumes a moderate share, the audit is fast, and reasoning carries the highest computational cost. The use of asynchronous processing and streaming ensures that users receive real-time feedback during execution. CPU usage remained stable, memory usage increased during LLM processing, and GPU acceleration significantly reduced inference time where available. Sequential processing of multiple documents was handled efficiently, with no memory leaks observed and minimal performance degradation. Although processing time is non-trivial, it remains acceptable given the depth of analysis performed, and optimisation opportunities exist in the reasoning layer through model tuning or hardware acceleration.

4.6 User Acceptance Testing

User acceptance testing was conducted with a group of students and researchers, evaluating ease of use, clarity of interface, accuracy of summaries and relevance of QA responses. Users found the system intuitive and responsive, considered the summaries informative and structured, and reported that QA responses were relevant and context-aware.

V.DISCUSSION

The results demonstrate that the Research AI Pipeline successfully achieves its objective of automating research paper analysis while maintaining high levels of accuracy and reliability. Compared conceptually with existing approaches, the system offers end-to-end pipeline integration, local deployment with the attendant privacy and cost advantages, audit-based validation as a unique contribution, and a structured QA system. Its limitations relative to some systems are a higher processing time than lightweight extraction tools and a requirement for moderate computational resources.

The system demonstrates several notable strengths: robust handling of diverse document formats, high accuracy in structural extraction, effective audit-based validation, reliable and context-grounded QA responses, and strong numerical consistency in summaries. The integration of hybrid extraction, audit validation and controlled LLM reasoning creates a system that is not only functional but also trustworthy — an essential requirement for academic applications. The system strikes a balance between automation and reliability, ensuring that users can depend on its outputs for meaningful insights without sacrificing data integrity.
Certain limitations were identified during evaluation. Accuracy is reduced for low-quality scanned documents, where OCR performance degrades. Highly complex tables may not be extracted perfectly. Processing time increases with document length. Domain adaptation remains limited without prompt tuning, and LLM reasoning may still require refinement for domain-specific jargon.

Several directions for future work follow from these limitations. Advanced document understanding models such as LayoutLM or transformer-based architectures that incorporate spatial awareness could improve extraction of complex layouts, especially in multi-column and highly formatted documents. OCR accuracy for low-quality scanned documents can be enhanced by incorporating image denoising, adaptive thresholding and deep learning-based OCR models. The reasoning layer can be improved by fine-tuning models or designing domain-specific prompt templates for specialised fields such as medicine, law or finance. Processing time can be reduced through quantization, caching strategies, GPU acceleration and parallel processing of independent tasks. A significant extension would be multi-document analysis enabling automated literature surveys, trend analysis and identification of research gaps across a collection of documents. Finally, the user interface can be enhanced with source-text highlighting for QA responses, visual summaries and dashboards, and export options, alongside integration with reference management tools and academic databases.

VI.CONCLUSION

The Research AI Pipeline presents a comprehensive and systematic solution to the growing challenge of research paper analysis in an era of rapidly expanding scientific literature. The system successfully integrates document ingestion, hybrid text extraction, audit-based validation, local large language model reasoning and interactive question answering into a unified and coherent pipeline. Its layered architecture ensures modularity, scalability and robustness, enabling clear separation ofresponsibilities and facilitating easier debugging, testing and future enhancement, while the hybrid extraction approach effectively handles both digitally generated and scanned documents.

The introduction of an audit and validation layer represents a significant contribution, as it provides a quantitative measure of extraction quality. This not only improves system reliability but also establishes a foundation for trust in downstream processes: by ensuring that only validated data is passed to the reasoning layer, the system minimises the propagation of errors and enhances the accuracy of generated summaries. The use of locally deployed large language models through Ollama ensures that the system operates without dependency on external cloud services, addressing critical concerns related to data privacy, cost and accessibility, and the chunk-based reasoning strategy further improves summary quality by preserving contextual integrity and reducing hallucination.

Experimental results across fifty research papers demonstrate strong performance, with an F1-score of 0.89 for section extraction, 0.91 for figure-caption alignment, 88% accuracy in factual question answering and an average processing time of 87 seconds per paper. The Research AI Pipeline therefore fulfils its objective of providing an intelligent, locally deployable and reliable system for automated research paper analysis, and represents a meaningful step toward enhancing research productivity by reducing the cognitive burden associated with literature review.

REFERENCES

1. C. L. Giles, K. D. Bollacker, and S. Lawrence, "CiteSeer: An automatic citation indexing system," in Proceedings of the 3rd ACM Conference on Digital Libraries, 1998, pp. 89–98.
2. P. Lopez, "GROBID: Combining automatic bibliographic data recognition and term extraction for scholarship publications," in International Conference on Theory and Practice of Digital Libraries, 2009.
3. S. Klampfl and R. Kern, "An unsupervised machine learning approach to body text and table of contents extraction from digital scientific articles," in International Conference on Theory and Practice of Digital Libraries, 2013, pp. 144–155.
4. J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in NAACL, 2019.
5. Y. Xu, M. Li, L. Cui, S. Huang, F. Wei, and M. Zhou, "LayoutLM: Pre-training of text and layout for document image understanding," in KDD, 2020, pp. 1192–1200.
6. Z. Xu, R. Jain, and M. Shah, "DocFormer: End-to-end transformer for document understanding," in ICCV, 2021.
7. T. Brown et al., "Language models are few-shot learners," in NeurIPS, 2020.
8. I. Beltagy, K. Lo, and A. Cohan, "SciBERT: A pretrained language model for scientific text," in EMNLP, 2019.
9. J. Maynez, S. Narayan, B. Bohnet, and R. McDonald, "On faithfulness and factuality in abstractive summarization," in ACL, 2020, pp. 1906–1919.
10. P. Lewis et al., "Retrieval-augmented generation for knowledge-intensive NLP tasks," in NeurIPS, 2020.
11. G. Mariani, F. Scheidegger, R. Istrate, C. Bekas, and C. Malossi, "Data validation in machine learning pipelines," IEEE Transactions on Knowledge and Data Engineering, 2019.
12. G. Tsatsaronis et al., "An overview of the BioASQ large-scale biomedical semantic indexing and question answering competition," BMC Bioinformatics, vol. 16, no. 138, 2015.
13. Q. Jin, B. Dhingra, Z. Liu, W. Cohen, and X. Lu, "PubMedQA: A dataset for biomedical research question answering," in EMNLP-IJCNLP, 2019, pp. 2567–2577.
14. DeepSeek-AI, "DeepSeek-V3 technical report," arXiv preprint arXiv: 2412.19437, 2024.
15. T. Cao, N. Raman, D. Dervovic, and C. Tan, "Characterizing multimodal long-form summarization: A case study on financial reports," arXiv preprint arXiv: 2404.06162, 2024.
16. S. Manic K., A. Al-Balushi, A. Al-Bemani, S. Al Araimi, B. G., U. Suresh, and A. Najeeb, "AI-driven multi-modal information synthesis: Integrating PDF querying, speech summarization, and cross-language text summarization," Procedia Computer Science, 2024.
17. S. Kumar, G. S. Kohli, T. Ghosal, and A. Ekbal, "Longform multimodal lay summarization of scientific papers: Towards automatically generating science blogs from research articles," in Proc. LREC-COLING, 2024, pp. 10790–10801.