

RESNOVA: AI-Powered Research Intelligence Platform

Prof. M. Santhosh^{*1}, Prof. R. Lalitha^{*2}, Mr. A. Benito Kenric Rowan^{*3}

^{*1}Assistant Professor, Software Systems & AIML Department, Sri Krishna Arts and Science College, Coimbatore, India.

^{*2}Assistant Professor, Software Systems & AIML Department, Sri Krishna Arts and Science College, Coimbatore, India.

^{*3}Student, Software Systems & AIML Department, Sri Krishna Arts and Science College, Coimbatore, India.

Abstract

ResNova is a browser-based application that helps anyone doing research, whether a student or a professional, manage a whole set of papers from a single place. Users can upload PDF documents, and the system processes each paper once, saving its textual content, segments, metadata, and reference list. All subsequent activities, such as searching for information, conducting a literature review, comparing papers, analyzing citations, identifying research gaps, writing proposals, drafting reports, tracking evidence, and planning the execution of research, are carried out using the stored data. This ensures that no data needs to be re-entered multiple times. Many existing research tools either store data or provide chat-based interaction, but they often do not clearly indicate the source of specific information. In ResNova, every result is linked to its source, along with the exact page and paragraph where it was found. The citation network is built entirely from parsed reference data and metadata, using standard algorithms to calculate PageRank, betweenness centrality, and connected components, ensuring the generated graph remains consistent. Google Gemini (and Groq in the Defense Lab) is used exclusively for tasks that are reviewed by researchers, and any issues with the provider are reported as errors rather than being concealed. The platform is built using React, Vite, and Tailwind CSS, with the server running on Node.js and Express.js. Data is stored in PostgreSQL, and user authentication is handled with JSON Web Tokens and bcryptjs. The goal of ResNova is to support researchers rather than replace their thinking by keeping all materials organized and easily traceable.

Keywords: Research Intelligence, Literature Review, Citation Network, PDF Ingestion, Research Gap Discovery, Retrieval-Augmented Generation, Node.js, PostgreSQL.

I. INTRODUCTION

A literature review often fails not due to a lack of available papers, but because the material becomes scattered across different folders and files. PDFs may be stored in one location, notes in another, references in a spreadsheet, and the research proposal in yet another document. When defending a claim, the user must search for the relevant paper and specific sentence again. Guidelines for systematic reviews highlight the significant effort required in searching, screening, and recording these materials [1]. While language models have simplified the generation of summaries, a summary without a traceable source holds little academic value [6], [7].

ResNova: AI-Powered Research Intelligence Platform aims to bring every step of the research workflow, from the first PDF upload through to a finished plan, into one web application. Work is organised around collections of papers instead of single files. Each paper is processed only once, and all future features rely on the stored results. From

the beginning, a rule was established: wherever a fixed procedure can compute an outcome, it should be done that way, and generative models are used only when the output will be reviewed and verified by a researcher.

A session starts with registration and login. The user creates a collection and uploads PDFs. The ingestion process extracts text, splits it into chunks linked to specific pages and paragraphs, reads metadata, and parses the references. Once this is complete, the collection can be searched, reviewed, and compared. The citation network shows how the uploaded papers are connected. Candidate research gaps are listed, and the user selects one, which then flows into a proposal and an execution plan without the need for re-entering data. All collected evidence retains its source details, and a Defense Lab feature allows users to test their positions against challenging questions.

The current version of the platform includes thirteen modules and runs in desktop browsers without requiring any installation. The paper continues in

the following order. Related work is surveyed in Section II, existing practice and the problem statement are covered in Section III, and the framework we propose is described in Section IV. Architecture, workflow, and implementation follow in Sections V to VII. Section VIII reports the testing, Section IX discusses the results, and Section X closes with conclusions and future work.

II. LITERATURE SURVEY

Our work is based on three main areas: literature review practices, citation and graph analysis, and retrieval-augmented generation along with secure web design.

Literature Review Practice

Guidelines for conducting systematic reviews outline a series of detailed manual steps, including searching for sources, screening them, extracting data, and documenting the rationale for each decision [1]. Each of these steps creates some form of intermediate knowledge, which is typically recorded in informal notes during regular practice. While tools that organize files or manage bibliographic information can help with organizing this information, they generally do not allow for querying the actual content of the papers themselves.

Citation and Graph Analysis

Citation indexes have been used for a long time to track how ideas are passed from one paper to another [2]. Today a set of papers, together with the references that connect them, can be modelled as a directed graph. PageRank scores each node according to how important its incoming neighbours are [3], betweenness centrality identifies papers that are frequently found on the shortest paths between other papers [4], and force-directed placement arranges such graphs so that connected nodes are closer together [5]. All three methods are consistent procedures that produce the same output when given the same input, which makes the results easy to justify.

Grounded AI Generation and Secure Web Design

In a retrieval-augmented setup, the model is given specific passages pulled from a document store, so its replies rest on the supplied text and not only on what it picked up during training [6]. However, even with this approach, generative models can sometimes present information that is not supported

by the source material [7], so the generated text should still be carefully reviewed. For the web layer, we follow the REST architectural style [8], use JSON Web Tokens to identify authenticated users [9], and store passwords using salted bcrypt-style hashing [10]. Our security practices are informed by the common web vulnerabilities listed by OWASP, which guided our implementation of access control and input handling checks [11].

Research Gaps

Reference management tools store bibliographic information but do not read the actual research papers. Chat-based assistants can help answer questions, but they usually do not maintain a long-term collection or provide specific page references. Among the tools we reviewed, none were able to process a researcher's own papers in a way that connects the same content throughout the research process—such as during search, review, comparison, citation analysis, gap identification, proposal writing, drafting, and planning—while clearly distinguishing between content that is automatically generated and content that is manually created. ResNova is our effort to address this gap.

III. EXISTING SYSTEMS AND PROBLEM STATEMENT

In the typical research process, a researcher downloads papers, reads them one by one, manually copies useful information into notes, and creates comparison tables by hand. References are checked individually, and searching usually stops at the file name or what can be found within a single PDF using a viewer's search function. This is manageable for a small number of papers. However, as the number of papers increases, the process becomes slower, and it becomes easy to lose track of how different papers are connected. While available tools can address some aspects of this challenge, most of them share several common limitations.

- **Scattered material.** Papers, notes, comparison tables, and drafts are stored in separate locations, resulting in no unified workspace and no links between them.
- **Lost provenance.** Notes often lose connection with the original page or paragraph they were taken from, making it difficult to verify claims without referring back to the full paper.
- **Content that cannot be queried.** Reference managers only store titles and authors, and PDF

viewers can only search within a single file at a time, so it is not possible to search across an entire collection of papers.

- **Hidden structure.** The references within a researcher's own work reveal the organization of the field, but they are rarely transformed into a format that can be examined or analyzed.
- **Broken continuity.** Insights gained during the review phase are not available when drafting a proposal, and chat-based tools can generate fluent text without a clear source.

Our goal was to determine whether a single web application could effectively manage a researcher's papers, evidence, and decisions in a connected manner from the initial upload to the final plan, while clearly showing the origin of every output and being transparent about its limitations. ResNova is our answer.

IV. PROPOSED WORK: RESNOVA FRAMEWORK

ResNova combines paper processing, search, analysis, and AI assistance into a single workspace. It is built on three key commitments. First, a paper is processed only once, and the results are used throughout the entire workflow. Second, every output maintains a link to its original source. Third, deterministic tasks are kept separate from generative tasks, ensuring that only the parts of the work requiring human review are sent to a language model. PDF ingestion is all-or-nothing: papers are processed in a single transaction, and scanned images without text are rejected with a clear message, rather than being stored as blank records. Search operates on the stored content in chunks and returns each result along with the paper it came from, and where possible, the specific page and paragraph. Literature reviews and comparisons are built based on the papers the user selects, ensuring that generated text stays linked to a known set of sources. The citation network is constructed separately, without the use of generative models. Reference lists are analyzed using pattern matching and simple rules, then compared with the title, author, and publication year of the uploaded papers. A reference is only linked if a DOI or arXiv identifier matches, or if the surname, year, and a significant portion of the title align; otherwise, it

remains unmatched. PageRank, betweenness, and connected components are then calculated, and a manually created force-directed layout is used to visualize the graph as an SVG.

Gap discovery identifies potential research gaps, but none are selected automatically. The user chooses one. That choice is saved as a research idea, and both the proposal and the roadmap point to it instead of holding their own copy of the wording, so they stay tied to the original gap. Evidence records hold the fields that were actually extracted: paper, author, year, page, and paragraph. Any field that is absent is labelled as unavailable and is never guessed.

Objectives of ResNova

The project defines nine objectives, which fall into four groups.

- **Organisation and ingestion.** Collections owned by the user, and conversion of each PDF into text, chunks, metadata, and references tied to page and paragraph positions.
- **Retrieval and analysis.** Search across processed content, literature reviews, side-by-side paper comparison, and a deterministic citation network.
- **Research continuity.** Gap identification, proposal generation, drafting, and an execution roadmap that build on the chosen gap, supported by evidence that can be traced to its source.
- **Trust and security.** Plain reporting of AI provider failures, hashed passwords, JWT-based login, and an ownership check on every route.

V. SYSTEM ARCHITECTURE

ResNova consists of a client and a server. The React frontend handles screen rendering, shared state, and drawing of the citation graph. The Express backend exposes REST endpoints for authentication, collections, search, citation analysis, literature review, comparison, gap identification, proposals, drafting, roadmaps, evidence tracking, activity logging, and the Defense Lab, with a separate route module for each. All browser requests pass through one API module, which attaches the token and turns non-JSON failures into readable messages. The architecture is shown in Figure 1, and each layer is described in Table 1.

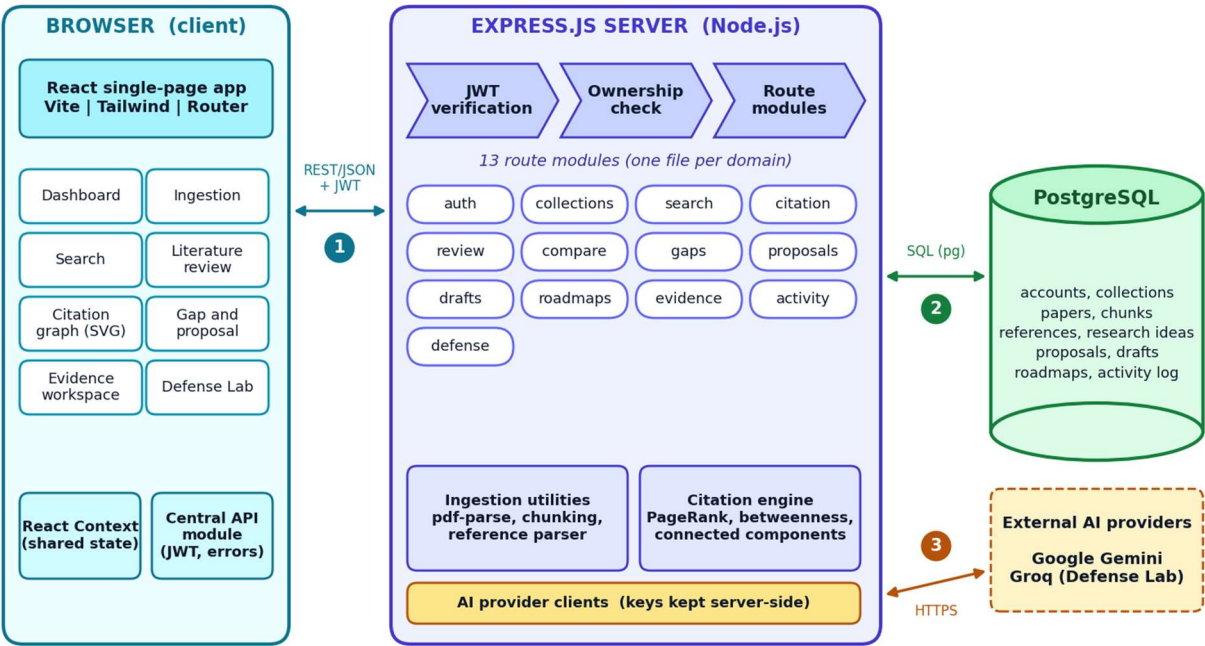


Figure 1: System architecture of ResNova

Table 1: Overview of the architecture

Layer	Component	Function
1. Presentation	React, Vite, Tailwind CSS, React Router DOM	Displays the login, dashboard, ingestion, search, review, citation network, gap, proposal, evidence, roadmap, and Defense Lab screens.
2. Client state and API	React Context, centralized API module	Keeps the active collection and evidence in shared state, adds the JWT to every request, and turns failures into readable messages.
3. Service	Node.js, Express.js, jsonwebtoken, pdf-parse	Validates the token, enforces ownership, performs ingestion, search, and citation analysis, and calls Gemini and Groq from the server side.
4. Data	PostgreSQL (pg driver, raw SQL)	Holds accounts, collections, papers, chunks, references, research ideas, proposals, drafts, roadmaps, and the activity log.

Two design choices safeguard the user's data. Authentication and ownership are verified in separate steps: the JWT middleware establishes who is calling, while each route confirms that the requested collection or paper belongs to that person. API keys live only in server environment variables, so frontend code cannot read them and they never reach the browser bundle.

VI. WORKFLOW OF THE RESNOVA SYSTEM

ResNova's process runs through eight stages, shown in Figure 2.

1. Authentication: The user registers or signs in and receives a signed JWT. Every failed login returns the same generic message.
2. Collection setup: The user creates a collection, which anchors every later module.
3. Ingestion: Each PDF is validated and processed a single time into metadata, chunks, and parsed references, all saved in one transaction.
4. Search and review: Chunk-level search returns results with their sources, and Gemini builds a literature review from those results.
5. Comparison and citation analysis: Studies are placed side by side, and a citation graph is constructed from the matched references.

6. Gap discovery: Candidate gaps are listed; the user picks one, which is stored as a research idea.
7. Proposal and roadmap: Both are produced from that same research idea.
8. Evidence, drafting and defense: Evidence keeps its source details, drafts are edited in place, and the Defense Lab raises counter-arguments.

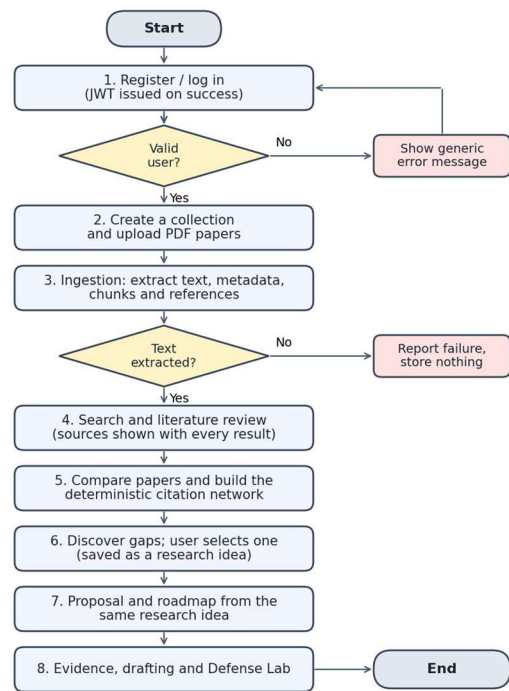


Figure 2: System flow of ResNova

VII. IMPLEMENTATION

ResNova was built in phases. Project setup and authentication came first, then ingestion, then the analysis modules. The citation system was

completed before any AI-assisted feature so that the deterministic components could be tested on their own. Interface polish and validation followed. Table 2 lists the technologies used.

Table 2: Technologies used

Category	Technology	Purpose
Frontend	React 18, Vite, Tailwind CSS, React Router DOM, Context API	Screens, routing, shared state, and the SVG citation graph
Backend	Node.js, Express.js	REST endpoints, route guarding, and application logic
Database	PostgreSQL, pg driver (raw SQL)	Relational tables for papers, chunks, references, and research outputs
Security	jsonwebtoken, bcryptjs	Token-based access control and password hashing
PDF and AI	pdf-parse, custom utilities, Google Gemini, Groq	Extracting text, chunking, parsing references, and running generative tasks
Tools	Visual Studio Code, npm	Coding environment and package management

Frontend. On the client side we built a single-page JavaScript app using React, bundled with Vite [12], [13]. React Router handles navigation, the Context API holds shared state such as the active collection and evidence list, and Tailwind CSS provides the styling. The citation network is

drawn as an SVG so that nodes can respond to hover and selection. Because all requests pass through one API module, a cross-cutting change touches only a single file.

Backend. The service layer is written for Node.js using the Express.js framework [14]. A token-

checking guard verifies the JWT [9] before any protected route executes, and passwords are stored as bcryptjs hashes [10]. Uploaded PDFs are handled by pdf-parse together with custom utilities that extract text and metadata, divide the content into chunks that keep page and paragraph information, and parse the reference list [17]. Google Gemini [18] powers literature reviews, comparisons, gap suggestions, proposal and roadmap generation, and drafting, while the Defense Lab runs on a separate Groq configuration [19]. When a service is down or a key is absent, the response is an error that names the provider. Retrieval is implemented in custom code in the

retrieval-augmented style [6], with no orchestration framework.

Database. PostgreSQL is the primary database, accessed through the pg driver using parameterized SQL and no ORM [15], [16]. Its tables hold accounts, collections, papers, chunks, references, research ideas, proposals, drafts, roadmaps, and activity events. Proposals and roadmaps point to the research idea by foreign key rather than storing the gap text themselves. Figure 3 traces how data moves between the user, the processes, and the stores.

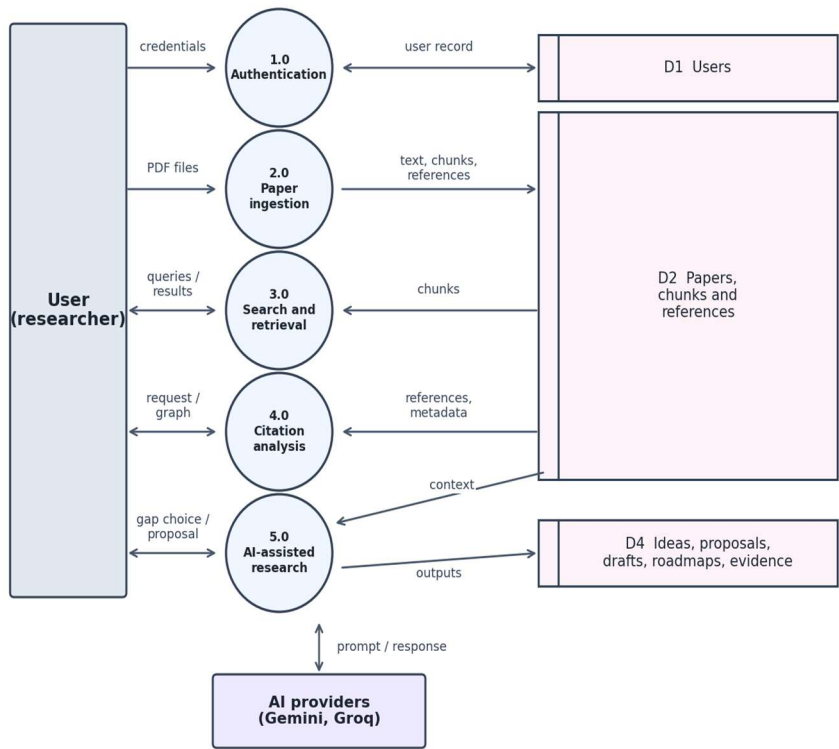


Figure 3: Data flow at Level 1 for ResNova

Reliability and error handling. Deterministic modules have no dependence on external models. If Gemini or Groq is unreachable or rate-limited, only the AI-assisted features are affected; search, the citation graph, evidence, and every other database-backed function carry on as usual. During development, the repository provides an in-memory JSON store that takes over when PostgreSQL is unavailable. It keeps the interface working but is not equivalent to the real database.

The production deployment configuration has also not been tested or verified in the current repository.

VIII. TESTING

Testing covered six levels: unit, functional, system, security, usability, and acceptance. A total of 38 documented test cases were executed across seven areas, and Table 3 summarises what each area covered and how it ended.

Table 3: Summary of testing

Test area	Focus	Cases	Outcome
Authentication and authorization	Sign-up, sign-in, token validation, and ownership of collections and papers	6	All passed; 2 fixed after first run
Collections and ingestion	Creating collections, uploading PDFs, extracting text, chunks, and references, rejecting scanned PDFs	6	All passed; 2 fixed after first run
Search and retrieval	Chunk-level search displaying paper, page, and paragraph	4	All passed; 1 fixed after first run
Citation network and graph analysis	Reference matching, PageRank, betweenness, components, and a repeatable layout	5	All passed; 1 fixed after first run
AI-assisted modules	Review, comparison, gaps, proposal, draft, roadmap, and Defense Lab	8	All passed; 2 fixed after first run
Evidence and research outputs	Provenance fields, display of unavailable fields, and proposal and roadmap tied to the gap	6	All passed; 1 fixed after first run
Error and provider handling	Provider errors, absent keys, failed ingestion, and readable error messages	7	All passed; 1 fixed after first run
Total		38	All passed; 10 fixed after first run

The security checks covered requests with a missing or expired token, attempts to reach another user's collection, a login with a wrong password and another with an unregistered email, an inspection of the database to confirm that passwords are stored only as bcrypt hashes, verification that no provider key appears in the production client bundle, and SQL syntax entered into the search box [11]. Roughly one case in four

failed on its first run, and the ten defects responsible were corrected before the final run.

IX. RESULTS AND DISCUSSION

Besides running the test cases above, we guided a new user through the whole workflow, from an empty account to a completed roadmap. Table 4 reports the outcome for each major feature, and the notes in Table 5 are observations made during use rather than measured benchmarks.

Table 4: Results of the main features

Feature	Expected behaviour	Result obtained
User authentication	Safe sign-up and sign-in using a JWT, without revealing which accounts exist	Passwords hashed, access by token, and a single generic failure message
Collections and ingestion	Every paper processed once into text, chunks, metadata, and references	Records saved together; scanned PDFs refused with a clear error
Search	Chunk-level results that carry their source	Each result returned with its paper, page, and paragraph
Literature review	Synthesis based on the papers selected	Synthesis lists the papers it drew on; provider errors are displayed
Paper comparison	Consistent layout across the chosen studies	Grid of dimensions against papers generated

Feature	Expected behaviour	Result obtained
Citation network	Repeatable graph built from actual references	Identical graph and metrics on each run; coverage summary displayed
Gap, proposal and roadmap	The single chosen gap carried onward	Proposal and roadmap draw on the same research idea
Evidence workspace	Provenance retained for every item	Source fields preserved; absent ones marked unavailable
AI provider failure	Clear error and no fabricated content	Error identifies the provider that failed

Table 5: Performance observations

Aspect	Observation	Assessment
Login, collections, search	Fast responses at the tested scale	Good
PDF ingestion	Slowest step; duration rises with document length	Acceptable
Citation graph	Calculated on demand with no external call; same result on repeated runs	Good
AI-assisted modules	Duration depends on the provider; a progress indicator is shown	Variable
Graph readability	Small collections yield sparse graphs; a coverage summary gives the reason	Acceptable

Discussion

On the whole, the main objectives of the project were achieved. A researcher can begin with an empty collection and finish with a roadmap on one platform, never retyping material, and every output can be linked back to a paper. Three design decisions had the greatest effect. A deterministic citation network is reproducible and costs nothing to recompute. Storing a pointer to the research idea, rather than a copy, keeps the proposal and roadmap in step with the gap. And reporting provider failures openly means an outage cannot be mistaken for an empty result. Testing also proved worthwhile: the first run exposed ten defects, among them a critical flaw that let any valid token open any collection, and a related fault in which a shared surname alone produced a false citation link. Both have been corrected.

Some limitations remain. Search relies on custom retrieval over stored chunks without a verified vector index, and its quality has not been measured. No OCR stage exists, so scanned papers are rejected. The researcher must still verify whatever Gemini and Groq produce. Collections cannot yet be shared, the interface targets desktop widths, and the production deployment setup is

unverified. These points are revisited in the future scope.

X. CONCLUSION AND FUTURE SCOPE

ResNova was built to demonstrate that a research workspace can keep a researcher's papers, evidence, and decisions linked from the first upload to the final plan. Every paper is processed once, and search, review, comparison, citation analysis, gap discovery, proposals, drafts, evidence, and roadmaps all operate on the same stored records. A React and Vite frontend communicates with an Express backend on PostgreSQL, while JWT login, per-route ownership checks, and server-held keys safeguard the user's work. The deterministic citation network yields repeatable results, and provider failures are reported instead of being masked by invented content. Across 38 test cases the main features behaved as intended, and the limitations are openly stated.

The project also deepened our understanding of PDF processing, reference parsing, graph algorithms, relational design, API security, and testing. Future work includes a verified vector index such as pgvector, together with labelled queries and references so that retrieval and

matching quality can be measured. Further plans are an OCR stage for scanned papers, DOI-based lookup of unmatched references, shared collections for supervised projects, BibTeX and structured evidence export, and a documented production deployment with key rotation and backups.

XI. REFERENCES

- [1] Kitchenham, B. (2004). Procedures for performing systematic reviews. Technical Report TR/SE-0401, Keele University.
- [2] Garfield, E. (1955). Citation indexes for science: A new dimension in documentation through association of ideas. *Science*, 122(3159), 108-111.
- [3] Brin, S., & Page, L. (1998). The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, 30(1-7), 107-117.
- [4] Brandes, U. (2001). A faster algorithm for betweenness centrality. *Journal of Mathematical Sociology*, 25(2), 163-177.
- [5] Fruchterman, T. M. J., & Reingold, E. M. (1991). Graph drawing by force-directed placement. *Software: Practice and Experience*, 21(11), 1129-1164.
- [6] Lewis, P., Perez, E., Piktus, A., et al. (2020). Retrieval-augmented generation for knowledge-intensive NLP tasks. *Advances in Neural Information Processing Systems*, 33, 9459-9474.
- [7] Ji, Z., Lee, N., Frieske, R., et al. (2023). Survey of hallucination in natural language generation. *ACM Computing Surveys*, 55(12), 1-38.
- [8] Fielding, R. T. (2000). Architectural Styles and the Design of Network-based Software Architectures. Doctoral dissertation, University of California, Irvine.
- [9] Jones, M., Bradley, J., & Sakimura, N. (2015). JSON Web Token (JWT). RFC 7519, Internet Engineering Task Force.
- [10] Provos, N., & Mazieres, D. (1999). A future-adaptable password scheme. *Proceedings of the USENIX Annual Technical Conference, FREENIX Track*, 81-91.
- [11] OWASP Foundation. (2021). OWASP Top Ten Web Application Security Risks. <https://owasp.org/www-project-top-ten/>
- [12] React Team. (2025). React Documentation. <https://react.dev>
- [13] Vite. (2025). Vite Documentation. <https://vitejs.dev>
- [14] OpenJS Foundation. (2025). Express: Node.js Web Application Framework. <https://expressjs.com>
- [15] PostgreSQL Global Development Group. (2025). PostgreSQL Documentation. <https://www.postgresql.org/docs>
- [16] node-postgres. (2025). node-postgres Documentation. <https://node-postgres.com>
- [17] pdf-parse. (2025). pdf-parse Package Documentation. <https://www.npmjs.com/package/pdf-parse>
- [18] Google. (2025). Gemini API Documentation. <https://ai.google.dev/gemini-api/docs>
- [19] Groq. (2025). Groq API Documentation. <https://console.groq.com/docs>