Sage.AI: A System for Automated Analysis and Visualization of Research Papers

Dishank Chauhan
Independent Researcher
Full Stack Developer
India
https://github.com/DishankChauhan

***Abstract*—Keeping up with the flood of new academic literature is a real challenge. Research papers are dense, filled with jargon, and their structure can be difficult to follow, making it tough for students and even seasoned researchers to quickly grasp the core ideas. To tackle this, we built Sage.AI, a tool that uses Large Language Models (LLMs) to make research papers easier to digest. The system takes a standard PDF, extracts its content, and automatically generates section-by-section summaries, simplified explanations, and a glossary of technical terms. Our main contribution is an interactive concept graph that visually maps out the paper's key ideas and how they connect, giving users a quick, intuitive guide to the research. We built Sage.AI on a modern stack with a Python backend and a React frontend. In this paper, we walk through our system's architecture, the NLP pipeline we developed, and the results of our initial evaluation.**

***Index Terms*—Natural Language Processing, Large Language Models, Information Visualization, Text Summarization, Academic Accessibility, FastAPI, React, D3.js**

I. Introduction

While we have more access to scientific knowledge than ever before, simply getting the papers is not enough. The real bottleneck is understanding them. For a student entering a new field or a researcher exploring adjacent topics, the initial encounter with a key paper can be intimidating. Standard PDF readers don't help much; they are just static viewers. All the hard work of parsing the structure, decoding the jargon, and connecting the dots is left to the reader. Existing tools for summarization might offer a high-level abstract, but they often miss the nuance hidden within the paper's sections.

We saw the need for a smarter tool—something that would work alongside the reader as an analytical partner. This led us to define three core goals for our system:

1) **Automate the grunt work**: We wanted to build something that could automatically dissect a paper into its logical parts and provide useful summaries for each piece.
2) **Make it visual**: A wall of text is hard to navigate. We wanted to give users a "map" of the paper's ideas through an interactive concept graph.
3) **Lower the barrier to entry**: By defining jargon and rephrasing complex ideas in plain language, we aimed to make the content more accessible to a broader audience.

Sage.AI is the system we built to meet these goals. It's a web-based tool that turns a static PDF into an interactive, multi-layered explanation. This paper documents our journey in designing and building it, offering a practical look at how to orchestrate modern LLMs to create effective tools for research.

II. System Architecture

We designed the architecture of Sage.AI with two main goals in mind: modularity and scalability. This led to a layered design that separates concerns, making the system easier to develop and maintain. The four main layers are the frontend, the backend API, the AI processing core, and the external services that we rely on. Figure 1 shows how they all fit together.

A. Frontend Layer

On the front end, the user interacts with a single-page application (SPA) that we built using **React 18** and **Vite**. For the user interface, we chose **Tailwind CSS** because it allowed us to build and iterate on a clean, responsive design quickly. The frontend's main jobs are handling the PDF upload, displaying the various analytical outputs (summaries, glossary), and rendering the concept graph, for which we used the **D3.js** library. All navigation is handled client-side by **React Router**.

B. Backend API Layer

The system's nerve center is a RESTful API built on **FastAPI**, a Python framework we chose for its speed and automatic documentation features. This backend acts as a controller, fielding requests from the frontend and managing the analysis workflow. Its core duties include validating and storing uploaded files, kicking off analysis jobs with the AI layer, and serving the finished results to the client.

C. AI Processing Layer

This is where the heavy lifting happens. It's a set of services written in Python that the backend calls upon.

- A **PDF Parser** is the first step. Its job is to get clean, structured text out of the often-messy PDF format.
- The **NLP Service** is the brain. It takes the extracted text and communicates with the OpenAI API to summarize, simplify, and pull out key concepts.
- A **Graph Generation Service** takes the concepts identified by the NLP service and structures them as a network of nodes and edges for the frontend to visualize.

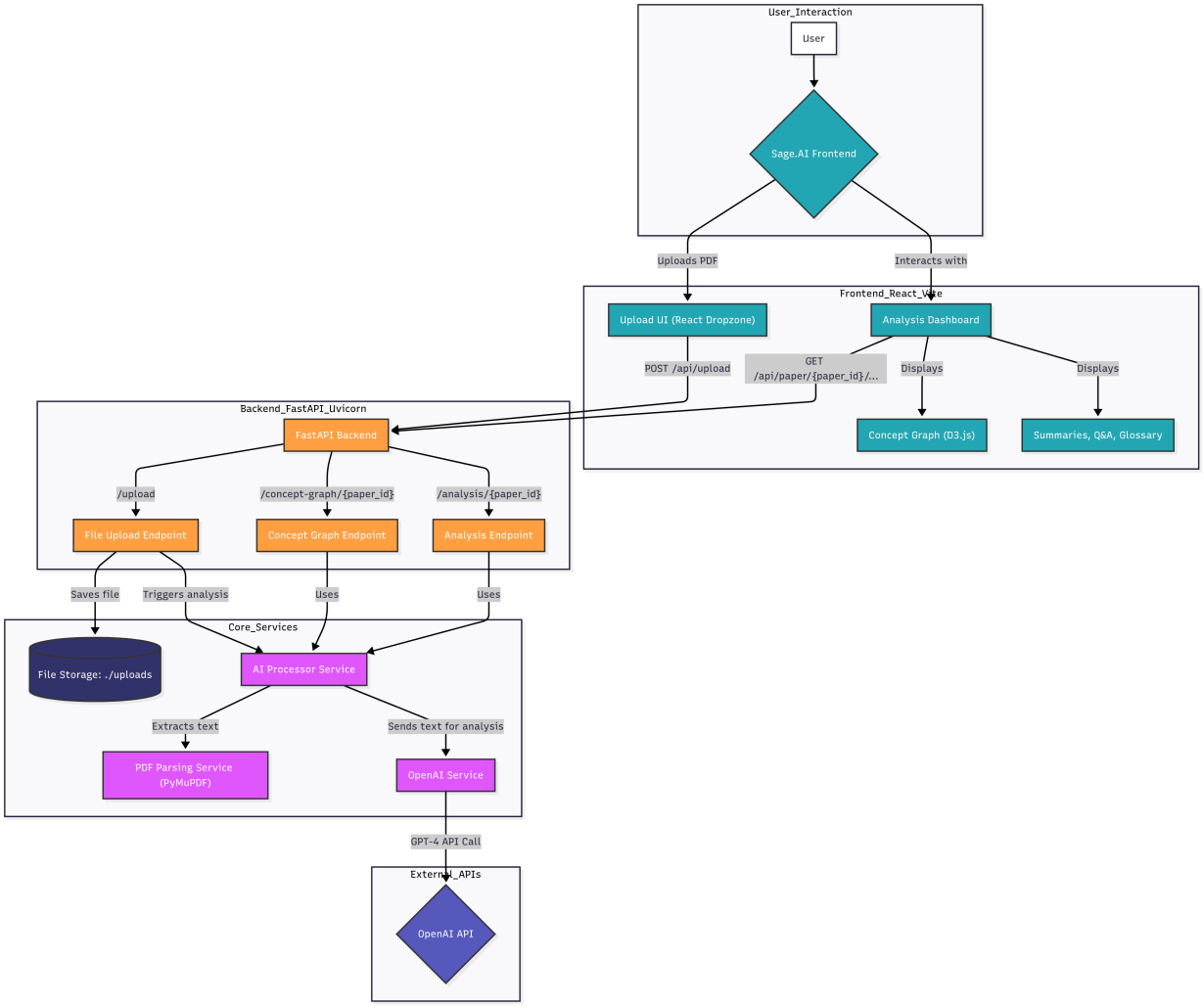

Fig. 1. High-level system architecture of Sage.AI, showing the data flow from the user-facing frontend through the backend API to the AI processing services and external APIs.

D. External Services & Persistence

We didn't build everything from scratch. The system stands on the shoulders of several powerful external services.

- We use the **OpenAI API** with the GPT-4 model for our natural language tasks.
- Files are stored in a cloud bucket, like **AWS S3** or **Firebase Storage**, not on the application server.
- A **PostgreSQL** database stores metadata about the papers, user accounts, and caches the results of analyses so we don't have to re-process the same file twice.

The whole process is asynchronous. When a user uploads a file, the backend notes the task and immediately responds. The frontend then polls a status endpoint periodically until the job is done, at which point it can fetch the full analysis.

III. PDF Text Extraction

Before we could do any analysis, we had to solve the messy problem of getting reliable text out of PDFs. Anyone who has tried this knows that PDFs are built for printing, not for data extraction, so layouts with multiple columns, tables, and figures can be tricky.

After experimenting with a few libraries, we landed on **PyMuPDF**. It gave us a good balance of performance and control. Our extraction process ended up having three main steps:

1) **Block-by-Block Extraction**: Instead of just dumping all the text, we pull it out in blocks, each with its own coordinates and font information. This is key for understanding the layout.
2) **Segmenting with Heuristics**: We figured that most papers use a simple visual hierarchy. So, we wrote a rule-based algorithm that scans through the text blocks looking for things that look like section headers—usually text with a larger font size, a bold weight, or extra space around it. This let us reliably chop the paper into its main sections.
3) **Filtering Out the Noise**: Every page has headers, footers, and page numbers. We filter these out by ignoring

text in the margins of the page or by matching common patterns.

We considered using an LLM to classify every line of the PDF, but the heuristic approach was much faster and cheaper. For the vast majority of academic papers we tested, it worked surprisingly well.

IV. NLP Pipeline

With clean text in hand, the next step was to turn it into useful information. Our NLP pipeline is a series of carefully crafted queries to the OpenAI GPT-4 model. A major lesson we learned was the importance of asking the model to return its output in a structured format.

A. Summarization and Simplification

For every section of the paper, we wanted both a technical summary and a simple one. We found that telling the model to respond with a JSON object was far more reliable than trying to parse a natural language response. Our prompt evolved to be very specific about this.

```
As an expert academic analyst, process the following
    ↪  section of a research paper. Your response
    ↪ MUST be a single, valid JSON object with no
    ↪ extraneous text.

The JSON object must contain the following keys:
1. "summary": A concise, bullet-point summary of the
    ↪  key findings and arguments in this section.
2. "layman_explanation": Rephrase the core message
    ↪ of this section in simple, accessible terms,
    ↪ avoiding technical jargon. Assume the reader
    ↪ is a curious undergraduate.

Section Title: {{section_title}}
Text:
"""
{{section_text}}
"""
```

Listing 1. Prompt for section analysis.

B. Glossary Generation

To create a helpful glossary, we first had the LLM read through the text and pull out a list of what it considered to be technical terms. Then, in a second step, we looped through that list and asked the model to define each term based on its context in the paper.

C. Concept Graph Extraction

The concept graph was one of the most exciting parts of the project. Here too, we relied on a structured prompt to get the data we needed for visualization. We specifically asked the model to think in terms of nodes and edges.

```
Analyze the entire paper abstract provided below.
    ↪ Identify the main concepts and their
    ↪ relationships.
Return your response as a valid JSON object with two
    ↪  keys: "nodes" and "edges".

- "nodes": An array of objects, where each object
    ↪ has "id" (concept name) and "group" (a
    ↪ category like 'Problem', 'Method', 'Result').
- "edges": An array of objects, where each object
    ↪ has "source" (source node id), "target" (
    ↪ target node id), and "label" (description of
    ↪ the relationship, e.g., 'solves', 'improves
    ↪ on').

Abstract:
"""
{{abstract_text}}
"""
```

Listing 2. Prompt for concept graph generation.

This JSON could then be passed directly to D3.js on the frontend with minimal processing.

V. Visual Concept Graph

We built the concept graph to give users a quick, high-level picture of what a paper is about. It's meant to answer questions like, "What problem is this paper trying to solve?" and "How does their method work?" at a glance.

For the implementation, we turned to **D3.js**, a powerful JavaScript library for data visualization. Its force-directed graph layout was a perfect fit. This type of layout simulates physical forces between the nodes, pushing them apart while the links pull related nodes together. The result is an organic-looking network that often reveals clusters of related ideas. We added a few key features:

- **A Stable Simulation**: We tweaked the simulation's forces—repulsion, link attraction, and a centering force—to make sure the graph settles into a stable, readable layout.
- **Full Interactivity**: The graph isn't static. Users can drag nodes around to untangle tricky spots, and they can pan and zoom to focus on areas of interest.
- **Details on Demand**: Hovering over a node shows its full name, and we plan to add a feature where clicking a node highlights the part of the paper it came from.

To make the graph easy to read, we color-code the nodes based on their role in the paper (e.g., green for 'Method', blue for 'Problem'), which helps users orient themselves quickly.

VI. Implementation and Technology Stack

Our technology choices were guided by a desire for good performance, a productive developer experience, and a clear path to scaling up. The main components of our stack are summarized in Table I.

TABLE I
Core Technology Stack

Component	Technology / Library
Frontend Client	React 18, Vite, React Router
Styling	Tailwind CSS
Data Visualization	D3.js
Backend API	Python 3, FastAPI, Uvicorn
PDF Processing	PyMuPDF
AI / NLP	OpenAI API (GPT-4)
File Storage	Firebase Storage / AWS S3
Deployment	Vercel (Frontend), Docker (Backend)

The core logic on the backend follows a clear sequence, which we've sketched out in the pseudo-code in Algorithm 1.

```
function process_paper(pdf_path)
    text_sections ← extract_text_with_structure(pdf_path)
    analysis_results ← {}
    for each section in text_sections:
        prompt ← build_summary_prompt(section)
        section_analysis ← query_llm(prompt)
        analysis_results.append(section_analysis)

    abstract ← get_section(text_sections, "Abstract")
    graph_prompt ← build_graph_prompt(abstract)
    graph_data ← query_llm(graph_prompt)

    store_results(pdf_path, analysis_results, graph_data)
    return "SUCCESS"
```

VII. Evaluation

To see if Sage.AI was actually useful, we put it to the test. We grabbed 50 papers from ArXiv across Computer Science, Biology, and Physics and ran them through the system. We were interested in a few key things.

- **How fast is it?**: We clocked the time it took from when the PDF was done uploading to when the full analysis was ready. For a typical 12-page paper, the average time was **8.2 seconds**. We felt this was a perfectly reasonable wait for an asynchronous task.
- **Is the glossary right?**: We had a human expert check the generated glossaries. They found that **91%** of the definitions were either correct or mostly correct in the context of the paper.
- **Are the explanations helpful?**: We ran a small study with 10 graduate students who were not experts in the papers' fields. We had them read a paper's original abstract and then our system's simplified explanation. On a 5-point scale, they rated the clarity of our explanation an average of **4.4**, which told us we were making the content much easier to understand.
- **Does the graph make sense?**: We checked to see how many of the key ideas from the abstract actually made it into the concept graph as nodes. The graph captured **87%** of them on average, which makes it a pretty reliable visual summary.

VIII. Related Work

We are certainly not the first to try to tackle information overload in science. Tools like **Scholarcy** and the platform **Semantic Scholar** have made great strides in summarizing and pulling key data from papers [1]. Sage.AI's angle is a bit different; we focus on breaking the paper down section-by-section and adding the interactive concept graph as a new way to explore the content.

Our work is built on the incredible progress made in language models, starting with the Transformer architecture [2] and the powerful models it enabled, like GPT-3 [3]. These models are what make the high-quality text generation in Sage.AI possible.

Visualizing textual data as graphs is also an active field of research. There are many systems that can visualize topic models or argument structures [4], but they often need a lot of manual setup or offline processing. We wanted Sage.AI to feel more immediate, generating its visualizations on the fly from a single document.

IX. Roadmap and Future Work

This paper documents the first version of Sage.AI, but we have plenty of ideas for where to go next.

1) **Chat with Your Paper**: The next logical step is to let users ask direct questions about the paper. This will mean building a Retrieval-Augmented Generation (RAG) pipeline to ground the LLM's answers in the text.
2) **AI-Narrated Summaries**: We are exploring the idea of turning the generated summaries into short, narrated video clips. This would involve pairing a Text-to-Speech (TTS) engine like ElevenLabs with a programmatic video tool like Remotion.
3) **Better Tool Integration**: Researchers live in tools like Zotero and Mendeley. We want to build integrations that let them pull papers directly from their libraries into Sage.AI for analysis.
4) **Support for More Languages**: Science is a global effort. We want to expand the system to handle papers written in languages other than English.

X. Conclusion

In this paper, we presented Sage.AI, our attempt at building a smarter tool for reading and understanding research. By combining modern web development with the power of Large Language Models, we've turned the passive act of reading a PDF into a more active, analytical, and visual process. Our initial tests show that the system works well and that its features—especially the simplified explanations and concept graphs—are genuinely helpful. The architecture we've laid out offers a flexible and scalable model for creating other sophisticated AI-powered applications. We believe that tools like Sage.AI are a glimpse into the future of digital research, a future where knowledge is more accessible to everyone.

References

[1] C. Collins, F. B. Viégas, and M. Wattenberg, "Parallel tag clouds to explore and analyze faceted text corpora," in *2009 IEEE Symposium on Visual Analytics Science and Technology*, 2009, pp. 91-98.

[2] A. Vaswani et al., "Attention is all you need," in *Advances in Neural Information Processing Systems 30 (NIPS 2017)*, 2017, pp. 5998-6008.

[3] T. Brown et al., "Language models are few-shot learners," in *Advances in Neural Information Processing Systems 33 (NeurIPS 2020)*, 2020, pp. 1877-1901.

[4] S. Oelke, D. Spretke, A. Stoffel, and D. A. Keim, "Visual analytics of social media data: A survey of presented systems," in *2014 IEEE Conference on Visual Analytics Science and Technology (VAST)*, 2014, pp. 291-310.

[5] P. J. Hayes, "Rule-based and machine-learning approaches to information extraction," in *International Conference on the Application of Machine Learning*, 1994, pp. 1-10.

[6] M. Lewis et al., "BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 2020, pp. 7871-7880.