
Project Overview 📄

The AI Research Paper Analyzer is a full-stack web application designed to help users quickly understand complex scientific documents. For its Minimum Viable Product (MVP), the application will allow a user to upload a research paper in PDF format and receive a concise, AI-generated summary.

The project's primary goal is to serve as a portfolio piece that demonstrates modern, industry-standard software engineering practices. This includes not just the application's code but also its containerization, cloud deployment, and automation, bridging the gap between academic research and production-ready software.

MVP Functionality

The user experience for the MVP is designed to be simple and direct, focusing on the core task of summarization.

1. **Upload: The user navigates to the web interface, which presents a clean UI with a file selector. The user chooses a PDF file from their computer and clicks a "Summarize" button.**
2. **Processing: The frontend sends the PDF file to a backend API running in the cloud. This backend service extracts the complete text from the document.**
3. **Summarization: The extracted text is then sent to the Google Gemini Large Language Model (LLM) with a specific prompt instructing it to summarize the research paper.**
4. **Display: The LLM generates the summary and sends it back to the backend. The backend then relays this summary to the frontend, which displays the text clearly for the user to read.**

Complete Technology Stack

The tech stack is chosen to be modern, scalable, and highly relevant to the skills sought after in top tech internships.

Application & AI

- **Frontend: React.js. A popular JavaScript library for building dynamic, interactive user interfaces. It will be used to create the user-facing webpage.**
- **Backend: Python with FastAPI. A high-performance Python framework for building APIs. It will receive the file upload, manage the AI interaction, and send back the result.**
- **AI Service: Google Gemini API. The core AI component that will handle the complex task of reading and summarizing the research paper's text.**
- **Core Libraries:**
    - **PyMuPDF: A Python library used to efficiently extract raw text from PDF documents.**
    - **python-dotenv: A utility to manage environment variables, specifically for keeping the Gemini API key secure.**

Infrastructure & Deployment

- **Containerization: Docker. The backend FastAPI application will be packaged into a standardized, portable Docker container. This ensures it runs reliably and consistently across different environments.**
- **Backend Hosting: Google Cloud Run. A serverless platform that is perfect for deploying and scaling containerized applications. It will run the Docker container that serves the backend API.**
- **Image Storage: Google Artifact Registry. A private repository for storing and managing the Docker images before they are deployed to Cloud Run.**
- **Frontend Hosting: Vercel or Netlify. These platforms are specialized for deploying static frontend applications (like our React app) quickly and efficiently, providing features like a global CDN and continuous deployment.**

- **Version Control: Git and GitHub. Used for tracking changes to the code and hosting the project repository. This is also the foundation for setting up CI/CD.**

Core AI/ML Enhancements 🧠

These features upgrade the AI capabilities of your application, moving beyond a simple summary.

- **Implement "Chat with Your Document" (RAG):** This is the most impactful next step. Instead of just a summary, allow users to ask specific questions about the paper's content.
    - **What you'll learn:** This forces you to learn **Retrieval-Augmented Generation (RAG)**, a cutting-edge AI architecture. You'll work with **text embedding models** and implement a **vector database** (like Pinecone, Chroma, or pgvector for PostgreSQL) to store and query document chunks. This is a highly sought-after skill.
- **Structured Data Extraction:** Automatically pull out key information from the paper—like the **methodology**, **dataset used**, **key results**, and **cited works**—and display it in a structured format.
    - **What you'll learn:** Advanced **prompt engineering** and how to design LLM outputs using formats like JSON for reliable data extraction.
- **Cross-Document Analysis:** Allow users to upload multiple papers and ask comparative questions, such as "What are the main differences in the methodologies of these two papers?"
    - **What you'll learn:** More complex state management in your backend and how to engineer prompts that can synthesize information from multiple sources.

User Experience & Frontend Features ✨

These features make the application more useful and professional for the end-user.

- **User Accounts and Document History:** Allow users to sign up, log in, and see a dashboard of all the papers they have previously uploaded and summarized.
    - **What you'll learn:** You'll implement **authentication** (e.g., using a library like Passport.js or a service like Firebase Auth or Auth0), manage user sessions, and model user data in your **PostgreSQL** database.
- **Streaming Responses:** Instead of waiting for the full summary to generate, stream the response token-by-token from the LLM to the frontend, making the UI feel much faster and more responsive.
    - **What you'll learn:** Working with **asynchronous generators** in FastAPI and handling streaming data on the frontend with the `fetch` API.
- **Highlighting and Annotations:** As the summary is generated, highlight the corresponding sentences or sections in the original PDF text.
    - **What you'll learn:** This is a complex frontend challenge involving PDF rendering libraries (like `react-pdf`) and managing the positional data of text within the document.

Backend & Infrastructure Upgrades ⚙️

These features improve the robustness, scalability, and maintainability of your application.

- **Asynchronous Processing with a Task Queue:** When a user uploads a PDF, instead of processing it immediately in the API request, add it as a "job" to a task queue (like **Celery with Redis**). A separate "worker" service then picks up the job, processes the PDF, and notifies the user when it's done.
    - **What you'll learn:** This teaches you how to build more **scalable and resilient systems**. It prevents API timeouts for large documents and is the standard architecture for handling long-running tasks.
- **Automated CI/CD Pipeline:** Fully automate your testing and deployment. When you push code to GitHub, a **GitHub Actions** workflow should automatically run your tests, build your Docker images, push them to the registry, and deploy the new versions to Google Cloud Run.
    - **What you'll learn:** This is a critical **DevOps** skill that all top tech companies use. It demonstrates that you can build and maintain software efficiently and reliably.