Assignment 4 - End-to-End-Reseach-Tool

Dharun Karthick Ramaraj

Nikhil Godalla

Linata Deshmukh

Introduction

The **Assignment4-End-to-End-Research-Tool** is an advanced, AI-powered research assistant designed to automate and streamline the research workflow, from document ingestion to report generation. Built with a combination of **FastAPI** for backend operations and **Next.js** for the frontend interface, it leverages cloud services like **AWS S3**, **Snowflake**, and **Pinecone** for storage, processing, and querying large datasets. The tool incorporates **LangChain** and **CopilotKit** to manage multiple agents, each specialized in tasks such as document selection, querying, and external research, providing an intuitive, powerful platform for researchers.

Technologies and Their Roles

- **FastAPI**: Chosen for its speed and asynchronous capabilities, FastAPI allows efficient backend processing and seamless integration with Pydantic for data validation.
- **Next.js**: Selected for its high-performance, interactive user interface, essential for real-time research interaction.
- **LangChain** and **CopilotKit**: These enable a modular, scalable multi-agent system with various agents specialized in querying, document retrieval, and interacting with language models.
- **AWS S3**: Used for cloud storage, facilitating secure, scalable access to research documents.
- **Snowflake**: Provides a robust data warehouse for storing and accessing publication metadata, enhancing retrieval speeds.
- **Pinecone**: Acts as a vector store, supporting similarity search and RAG (Retrieval-Augmented Generation) by enabling quick, relevant responses to user queries.
- **Google Docs API and Codelabs**: Supports the export of research notes to Google Docs in Codelabs format, offering a structured, shareable document style for research findings.

The main goal is to create a tool that allows users to efficiently parse, retrieve, and research document-based content, complete with export options for sharing results in professional formats (PDF, Codelabs).

Problem Statement

The research assistant tool solves the challenge of managing large sets of documents, performing efficient queries, and producing structured, exportable outputs. Traditional research workflows are often fragmented, requiring separate tools for each step. This tool consolidates research tasks, from document selection and query processing to output generation, all within a single interface.

Description of the Problem

The research process involves handling massive volumes of unstructured data from various sources, including academic publications, web articles, and documents. Researchers often face challenges such as:

1. **Inefficient Data Management**: Difficulty in organizing and storing research publications, metadata, and related materials for easy access.
2. **Time-Consuming Content Retrieval**: Extracting specific, relevant information from large documents is labor-intensive and prone to errors.
3. **Lack of Interactive Insights**: Limited tools to query content contextually and interactively, hindering deeper analysis.
4. **Fragmented Documentation Workflow**: Lack of seamless integration for note-taking, drafting, and exporting research in shareable formats.

Desired Outcome

The solution aims to provide an **AI-powered research assistant** that simplifies and accelerates the research workflow by offering:

1. **Automated Data Ingestion**: A system to scrape, store, and manage research documents and metadata.
2. **Efficient Retrieval and Insights**: Tools for querying document content using RAG (Retrieval-Augmented Generation) to deliver precise and context-aware responses.
3. **Streamlined Workflow**: An intuitive interface for document selection, note-taking, and drafting, integrated with robust export options like PDF and Google Codelabs.
4. **Scalability and Reliability**: A backend architecture capable of handling large-scale data with minimal latency, ensuring consistent performance.

Constraints and Limitations

1. **Data Source Constraints**:
    - Dependence on publicly accessible or API-supported sources like CFA publications and Arxiv.
    - Limited access to proprietary or paywalled content.
2. **System Requirements**:

    - Integration with Snowflake, AWS S3, Pinecone, and Google APIs requires proper setup of credentials and configurations.
    - Resource-intensive tasks such as document chunking and indexing demand optimized processing pipelines.
3. **Export Formatting**:
    - Ensuring consistent formatting for Codelabs and PDF exports may require additional validation and adjustments.
4. **Scalability Limitations**:
    - Handling increasing volumes of data and queries requires efficient database scaling and optimized API endpoints.

By addressing these challenges and working within the constraints, the project aims to provide a seamless, efficient, and user-friendly research assistant for academics, professionals, and students.

Proof of Concept

To solve the challenges outlined, this project leverages various technologies, each serving a specific purpose. The technologies were chosen based on their efficiency, flexibility, and ability to handle large data volumes:

Major Technologies

1. **FastAPI & Next.js**: Chosen for their speed and scalability, ensuring responsive interactions and real-time querying.
2. **Airflow**: Automates data pipeline workflows for data scraping, loading, and processing.
3. **AWS S3 & Snowflake**: S3 provides reliable cloud storage for documents, while Snowflake enables efficient retrieval of structured metadata.
4. **Pinecone**: Supports fast, context-aware document retrieval, essential for delivering relevant insights to user queries.
5. **CopilotKit**: Manages multi-agent processing for diverse tasks, such as document retrieval, RAG-based queries, and formatted exports.

Initial Setup and Testing

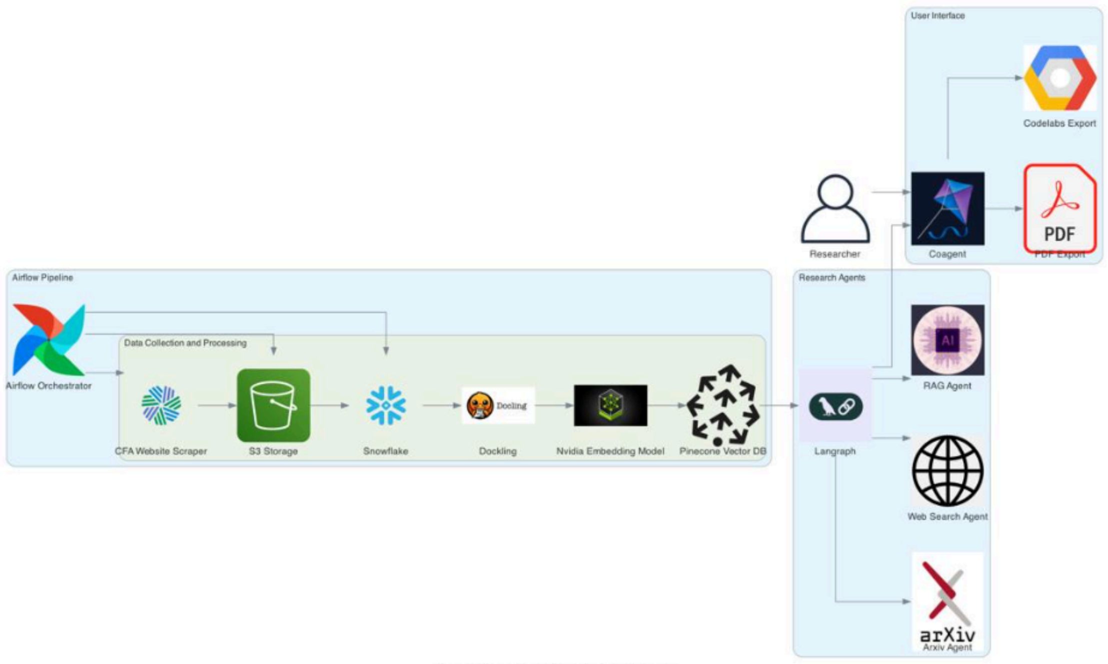
The initial setup included:

- Configuring API endpoints in FastAPI for document management.
- Setting up Airflow to automate pipeline tasks for data ingestion and loading.
- Testing Pinecone and Snowflake integration for smooth data retrieval.

Anticipated Challenges and Solutions

1. **Multi-Agent Coordination**: Managing consistent state across multiple agents. *Solution*: Use CopilotKit for efficient state management.
2. **Handling Large Documents**: Processing large PDFs and enabling efficient retrieval. *Solution*: Use Docling to chunk documents for faster search and retrieval.
3. **Codelabs Export Formatting**: Ensuring research drafts export correctly to Codelabs format. *Solution*: GPT-4 structures and formats drafts, and Google Docs API uploads them for easy sharing.

Architecture Diagram



Simplified End-to-End Research Tool Architecture

1. Frontend (UI)

- **Technology**: Built with **Next.js** and styled using **TailwindCSS**.
- **Role**: Provides an intuitive interface for researchers to interact with documents, ask questions, manage resources, and generate reports.
- **Key Features**:
  - **Query Interface**: Allows users to input research questions or specific queries.
  - **Document Selection**: Enables users to add or remove resources for analysis.
  - **Research Drafting and Export Options**: Provides options to export research drafts as PDFs or Codelabs, integrating with backend APIs for seamless document generation.
- **Interaction**:

    - Communicates with the backend for querying, retrieving resources, and exporting results.
    - Triggers document processing actions based on user input.

2. Backend (API Layer)

- **Technology**: **FastAPI** for creating RESTful API endpoints.
- **Role**: Serves as the central logic layer, connecting the frontend to the data pipelines, database, and AI services.
- **Key Features**:
    - **Document Management**: Handles routes for querying documents, generating summaries, and managing document metadata.
    - **Export Functionality**: Generates structured outputs for PDF and Codelabs formats.
    - **Processing Orchestration**: Interacts with the pipeline orchestrator (Airflow) for automated data collection and processing tasks.
- **Code Reference**:
    - document_selection.py: Manages document retrieval and metadata handling from Snowflake.
    - rag_router.py: Manages RAG (Retrieval-Augmented Generation) queries, leveraging embeddings in Pinecone.
    - export_router.py: Facilitates export requests for generating PDFs or Codelabs.
- **Interaction**:
    - Communicates with Airflow for data ingestion and processing.
    - Retrieves structured metadata from Snowflake and document embeddings from Pinecone.
    - Connects with AI services for summarization and query processing.

3. Pipeline Orchestration (Airflow)

- **Technology**: **Apache Airflow**.
- **Role**: Automates and schedules data ingestion and processing tasks.
- **Key Pipelines**:
    - **Scraping Pipeline**: Scrapes publications from the CFA website and uploads metadata to **AWS S3**.
    - **Processing Pipeline**: Processes PDF content using **Docling**, splitting it into chunks suitable for embedding.
    - **Snowflake Load Pipeline**: Loads structured data into **Snowflake** for querying.
- **Interaction**:
    - Runs periodically or on-demand to keep document data up-to-date in S3 and Snowflake.
    - Triggers processing tasks that store document chunks in Snowflake and embeddings in Pinecone.

4. Database (Snowflake)

- **Technology**: **Snowflake** Cloud Data Platform.
- **Role**: Stores metadata and document-related information, enabling fast and efficient querying.
- **Interaction**:
    - Metadata is loaded by Airflow and queried by the backend.
    - Backend fetches document metadata based on user queries.

5. Cloud Storage (AWS S3)

- **Technology**: **Amazon S3**.
- **Role**: Stores raw PDF files and processed data, including publication metadata.
- **Interaction**:
    - Airflow pipelines upload documents and metadata to S3.
    - Backend generates presigned URLs for secure access to stored files.

6. Vector Database (Pinecone)

- **Technology**: **Pinecone Vector Database**.
- **Role**: Stores embeddings of document chunks, enabling fast and scalable similarity-based retrieval during RAG queries.
- **Interaction**:
    - Document embeddings are generated by NVIDIA AI endpoints and indexed in Pinecone.
    - Backend retrieves relevant chunks based on similarity matching for user queries.

7. AI Services

- **Technologies**: **OpenAI** (for GPT-4), **NVIDIA AI endpoints**, **LangChain**.
- **Role**: Powers intelligent querying, summarization, and context-aware responses.
- **Interaction**:
    - Backend uses OpenAI for generating responses to user questions.
    - NVIDIA models create embeddings for document chunks, which are stored in Pinecone for RAG.
    - LangChain facilitates multi-modal RAG querying across text and document content.

8. Export Integration

- **Technologies**: **Google Drive and Docs APIs** for Codelabs export, **PDFKit** for PDF generation.
- **Role**: Provides export functionalities to save research drafts in user-friendly formats.
- **Interaction**:
    - Backend routes process export requests, converting research notes into structured Google Docs (Codelabs format) or downloadable PDFs.

Data Flow

Data Ingestion

1. Airflow scrapes CFA publications, storing metadata and PDFs in S3.
2. Metadata is loaded into Snowflake for structured querying.

Processing and Indexing

1. PDFs are processed using Docling, with chunks embedded using NVIDIA AI endpoints.
2. Document embeddings are indexed in Pinecone for efficient retrieval.

Frontend Interaction

1. Users query documents via the Next.js frontend.
2. Backend retrieves data from Snowflake or Pinecone, processes it with AI models, and generates responses.
3. Users can export results to Google Docs (Codelabs) or PDFs.

Backend Communication

- API endpoints facilitate interaction between the frontend, pipelines, and cloud services, ensuring seamless integration across the system.

Component Interaction Summary

- **Frontend** ↔ **Backend**: API endpoints handle document selection, querying, and export functionalities.
- **Backend** ↔ **Pipelines**: Backend triggers or retrieves results from Airflow pipelines for data ingestion and processing.
- **Backend** ↔ **Databases**: Fetches metadata from Snowflake and embeddings from Pinecone.
- **Backend** ↔ **AI Services**: Utilizes AI models for summarization, querying, and insight generation.
- **Backend** ↔ **Export Tools**: Generates outputs for PDFs and Google Docs.

Walkthrough of the Application

Application Overview

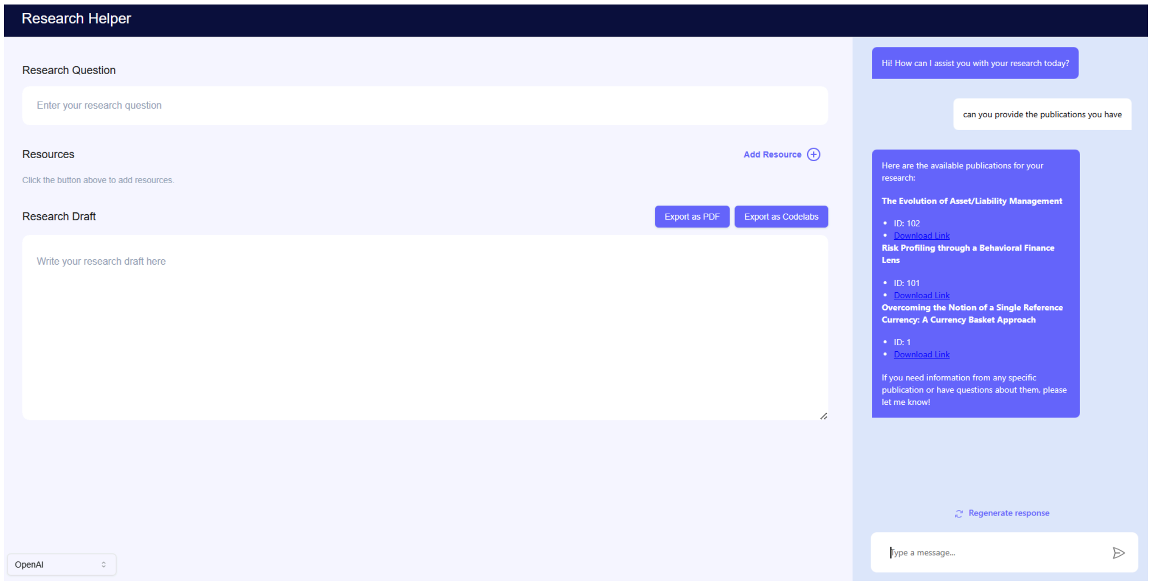
The **Research Helper** application provides a streamlined interface for conducting research and managing resources effectively. The main page is divided into three functional sections:

1. **Chat Interface**
    - On the right side of the page, there is a conversational chat interface where users can interact with the AI assistant. Users can:
    - Ask questions related to specific research publications. The AI will first search within the provided publications to answer the question. If the relevant information isn't found in the existing resources, the AI will broaden its search to the web.
    - View publications or resources available for research. Each publication is displayed with a title, ID, and a Download Link for easy access.
2. **Resources Management**:
    - In this area, users can add, edit, or delete resources relevant to their research. The resources that are retrieved in response to queries are automatically populated here, allowing users to manage and curate information conveniently.
3. **Research Draft**:
    - This is where users can draft and compile their research findings. Users can manually add content or import information directly from the responses generated by the AI.
    - Once the draft is complete, users have the option to **Export as PDF** or **Export as Codelabs**, making it easy to share, save, or continue working on the document in other formats.

This conversational approach makes it easy to ask follow-up questions, request summaries, and gather insights.

Image1: Asking a Question About Publications

In this image, the user initiates a query by typing in the right-side chat interface, asking for available publications. The AI assistant responds by listing relevant publications, complete with titles, IDs, and download links. This setup allows users to quickly access and download documents for their research. The left side of the screen remains empty initially, with placeholders indicating where research question input, resources, and draft notes will be displayed as the session progresses.



Hi! How can I assist you with your research today?

can you provide the publications you have

Here are the available publications for your research:

The Evolution of Asset/Liability Management

- ID: 102
- Download Link

Risk Profiling through a Behavioral Finance Lens

- ID: 101
- Download Link

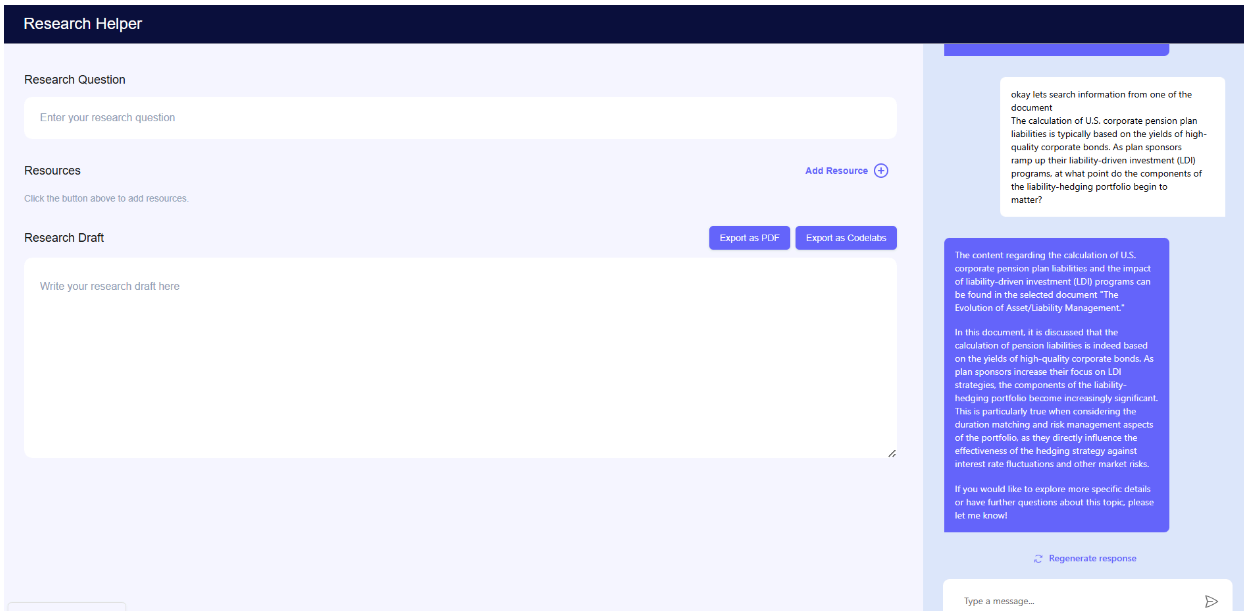
Overcoming the Notion of a Single Reference Currency: A Currency Basket Approach

- ID: 1
- Download Link

If you need information from any specific publication or have questions about them, please let me know!

Image 2: Adding a Research Question

In this image, the user inputs a specific research question in the **Research Question** field on the left side. This step allows the AI assistant to better understand the user's focus, guiding the interaction with relevant information and resources. The question also serves as a reference point for future interactions, helping structure the research process and responses from the assistant.



okay lets search information from one of the document
The calculation of U.S. corporate pension plan liabilities is typically based on the yields of high-quality corporate bonds. As plan sponsors ramp up their liability-driven investment (LDI) programs, at what point do the components of the liability-hedging portfolio begin to matter?

The content regarding the calculation of U.S. corporate pension plan liabilities and the impact of liability-driven investment (LDI) programs can be found in the selected document "The Evolution of Asset/Liability Management."

In this document, it is discussed that the calculation of pension liabilities is indeed based on the yields of high-quality corporate bonds. As plan sponsors increase their focus on LDI strategies, the components of the liability-hedging portfolio become increasingly significant. This is particularly true when considering the duration matching and risk management aspects of the portfolio, as they directly influence the effectiveness of the hedging strategy against interest rate fluctuations and other market risks.

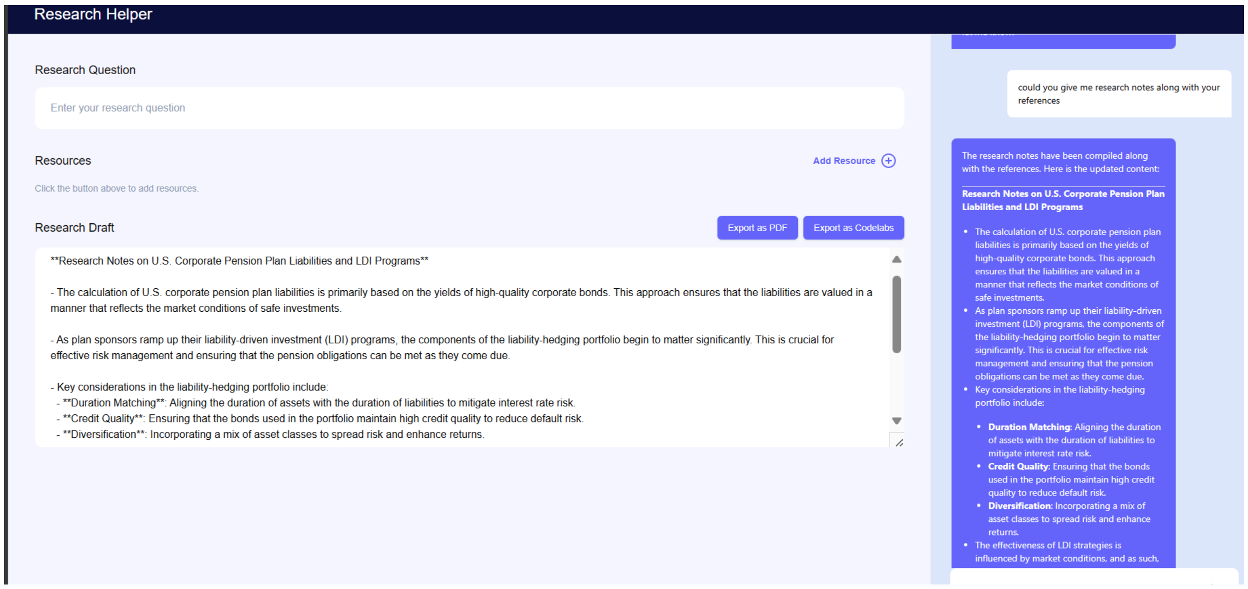
If you would like to explore more specific details or have further questions about this topic, please let me know!

Image 3: Adding Research Notes to the Draft

In this image, the user has asked the AI assistant on the right to generate research notes along with relevant references. The assistant responds with a well-organized summary, listing key insights and recommendations related to U.S. corporate pension plan liabilities and liability-driven investment (LDI) programs. This content is then transferred to the **Research Draft** section on the left, where it is displayed in a structured format.

The **Research Draft** area now includes detailed notes with bolded headings and bullet points for easy reading. These notes provide a concise overview of the requested topic, complete with sections on **Duration Matching**, **Credit Quality**, and **Diversification**, which are crucial considerations in liability-hedging portfolios.

At this stage, the user has the option to review the notes, make edits if needed, and proceed to export the document either as a PDF or in Codelabs format using the buttons provided. This seamless integration of research notes into the draft allows users to compile information quickly and efficiently within the app.



```
**Research Notes on U.S. Corporate Pension Plan Liabilities and LDI Programs**

- The calculation of U.S. corporate pension plan liabilities is primarily based on the yields of high-quality corporate bonds. This approach ensures that the liabilities are valued in a manner that reflects the market conditions of safe investments.

- As plan sponsors ramp up their liability-driven investment (LDI) programs, the components of the liability-hedging portfolio begin to matter significantly. This is crucial for effective risk management and ensuring that the pension obligations can be met as they come due.

- Key considerations in the liability-hedging portfolio include:
  - **Duration Matching**: Aligning the duration of assets with the duration of liabilities to mitigate interest rate risk.
  - **Credit Quality**: Ensuring that the bonds used in the portfolio maintain high credit quality to reduce default risk.
  - **Diversification**: Incorporating a mix of asset classes to spread risk and enhance returns.
```

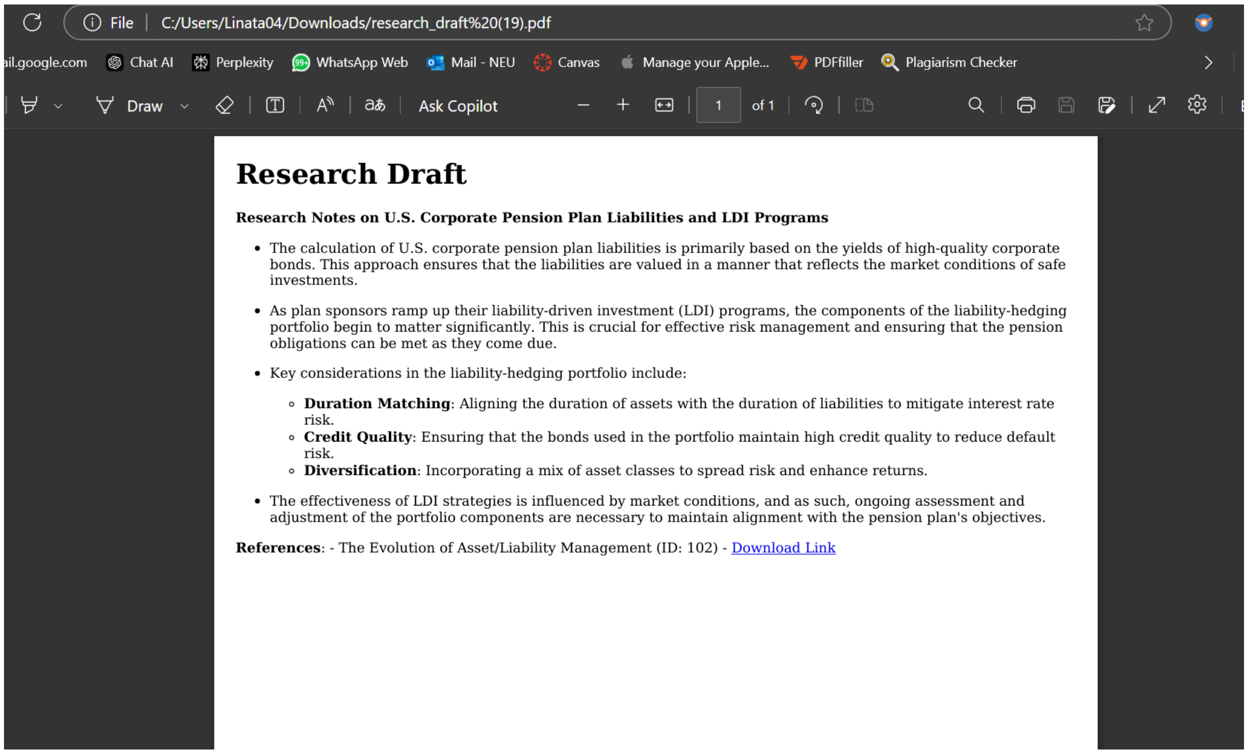
could you give me research notes along with your references

The research notes have been compiled along with the references. Here is the updated content:

Research Notes on U.S. Corporate Pension Plan Liabilities and LDI Programs

- The calculation of U.S. corporate pension plan liabilities is primarily based on the yields of high-quality corporate bonds. This approach ensures that the liabilities are valued in a manner that reflects the market conditions of safe investments.
- As plan sponsors ramp up their liability-driven investment (LDI) programs, the components of the liability-hedging portfolio begin to matter significantly. This is crucial for effective risk management and ensuring that the pension obligations can be met as they come due.
- Key considerations in the liability-hedging portfolio include:
  - **Duration Matching**: Aligning the duration of assets with the duration of liabilities to mitigate interest rate risk.
  - **Credit Quality**: Ensuring that the bonds used in the portfolio maintain high credit quality to reduce default risk.
  - **Diversification**: Incorporating a mix of asset classes to spread risk and enhance returns.
- The effectiveness of LDI strategies is influenced by market conditions, and as such,

After exporting to PDF



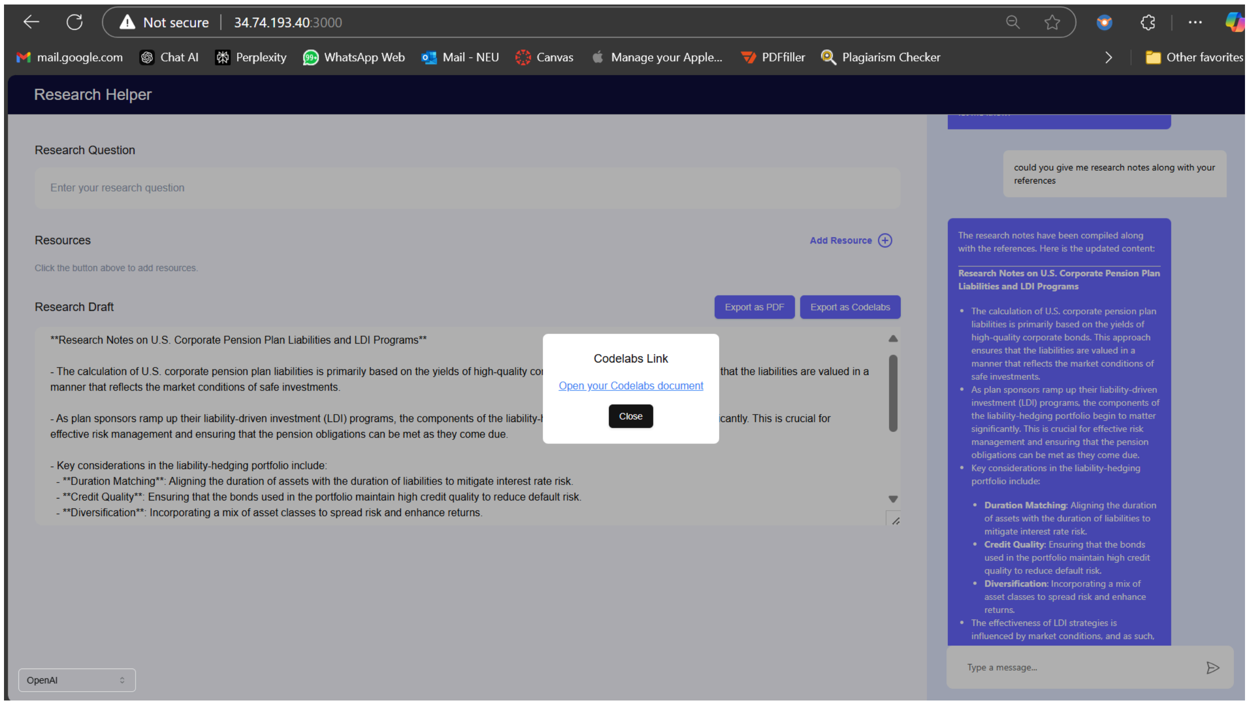
Research Draft

Research Notes on U.S. Corporate Pension Plan Liabilities and LDI Programs

- The calculation of U.S. corporate pension plan liabilities is primarily based on the yields of high-quality corporate bonds. This approach ensures that the liabilities are valued in a manner that reflects the market conditions of safe investments.
- As plan sponsors ramp up their liability-driven investment (LDI) programs, the components of the liability-hedging portfolio begin to matter significantly. This is crucial for effective risk management and ensuring that the pension obligations can be met as they come due.
- Key considerations in the liability-hedging portfolio include:
  - **Duration Matching**: Aligning the duration of assets with the duration of liabilities to mitigate interest rate risk.
  - **Credit Quality**: Ensuring that the bonds used in the portfolio maintain high credit quality to reduce default risk.
  - **Diversification**: Incorporating a mix of asset classes to spread risk and enhance returns.
- The effectiveness of LDI strategies is influenced by market conditions, and as such, ongoing assessment and adjustment of the portfolio components are necessary to maintain alignment with the pension plan's objectives.

References: - The Evolution of Asset/Liability Management (ID: 102) - Download Link

After clicking the **Export as Codelabs** button, a popup appears in the center of the screen, providing a link to view the exported document in Codelabs format. This popup contains a clickable link labeled "Open your Codelabs document," which redirects the user to the Codelabs platform, where the document is now formatted and accessible for further editing or sharing. The "Close" button on the popup allows the user to return to the main application interface once they have noted or accessed the Codelabs link.



Research Notes on U.S. Corporate Pension Plan Liabilities and LDI Programs

- The calculation of U.S. corporate pension plan liabilities is primarily based on the yields of high-quality co that the liabilities are valued in a manner that reflects the market conditions of safe investments.

- As plan sponsors ramp up their liability-driven investment (LDI) programs, the components of the liability-l icantly. This is crucial for effective risk management and ensuring that the pension obligations can be met as they come due.

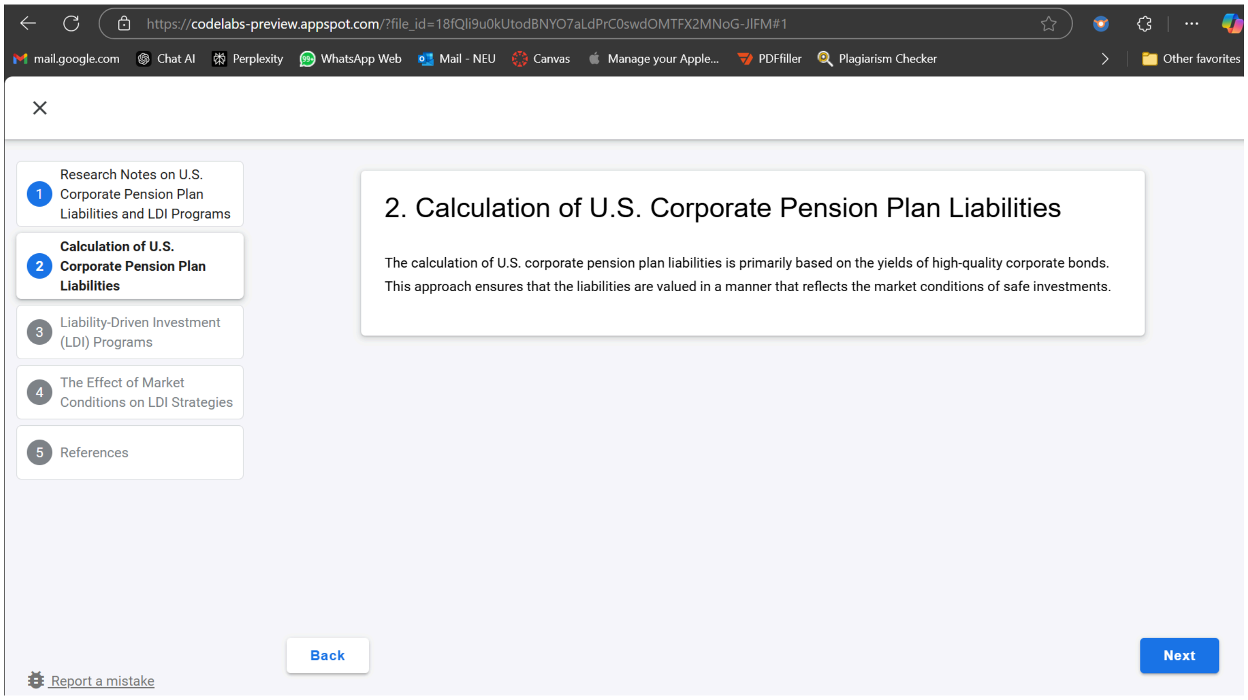
- Key considerations in the liability-hedging portfolio include:
 - **Duration Matching**: Aligning the duration of assets with the duration of liabilities to mitigate interest rate risk.
 - **Credit Quality**: Ensuring that the bonds used in the portfolio maintain high credit quality to reduce default risk.
 - **Diversification**: Incorporating a mix of asset classes to spread risk and enhance returns.

could you give me research notes along with your references

The research notes have been compiled along with the references. Here is the updated content:

Research Notes on U.S. Corporate Pension Plan Liabilities and LDI Programs

- The calculation of U.S. corporate pension plan liabilities is primarily based on the yields of high-quality corporate bonds. This approach ensures that the liabilities are valued in a manner that reflects the market conditions of safe investments.
- As plan sponsors ramp up their liability-driven investment (LDI) programs, the components of the liability-hedging portfolio begin to matter significantly. This is crucial for effective risk management and ensuring that the pension obligations can be met as they come due.
- Key considerations in the liability-hedging portfolio include:
  - **Duration Matching**: Aligning the duration of assets with the duration of liabilities to mitigate interest rate risk.
  - **Credit Quality**: Ensuring that the bonds used in the portfolio maintain high credit quality to reduce default risk.
  - **Diversification**: Incorporating a mix of asset classes to spread risk and enhance returns.
- The effectiveness of LDI strategies is influenced by market conditions, and as such,


https://codelabs-preview.appspot.com/?file_id=18fQli9u0kUtodBNYO7aLdPrC0swdOMTFX2MNoG-JlFM#1
mail.google.com
Chat AI
Perplexity
WhatsApp Web
Mail - NEU
Canvas
Manage your Apple...
PDFfiller
Plagiarism Checker
Other favorites
1 Research Notes on U.S. Corporate Pension Plan Liabilities and LDI Programs
2 Calculation of U.S. Corporate Pension Plan Liabilities
3 Liability-Driven Investment (LDI) Programs
4 The Effect of Market Conditions on LDI Strategies
5 References
Back
Next
Report a mistake

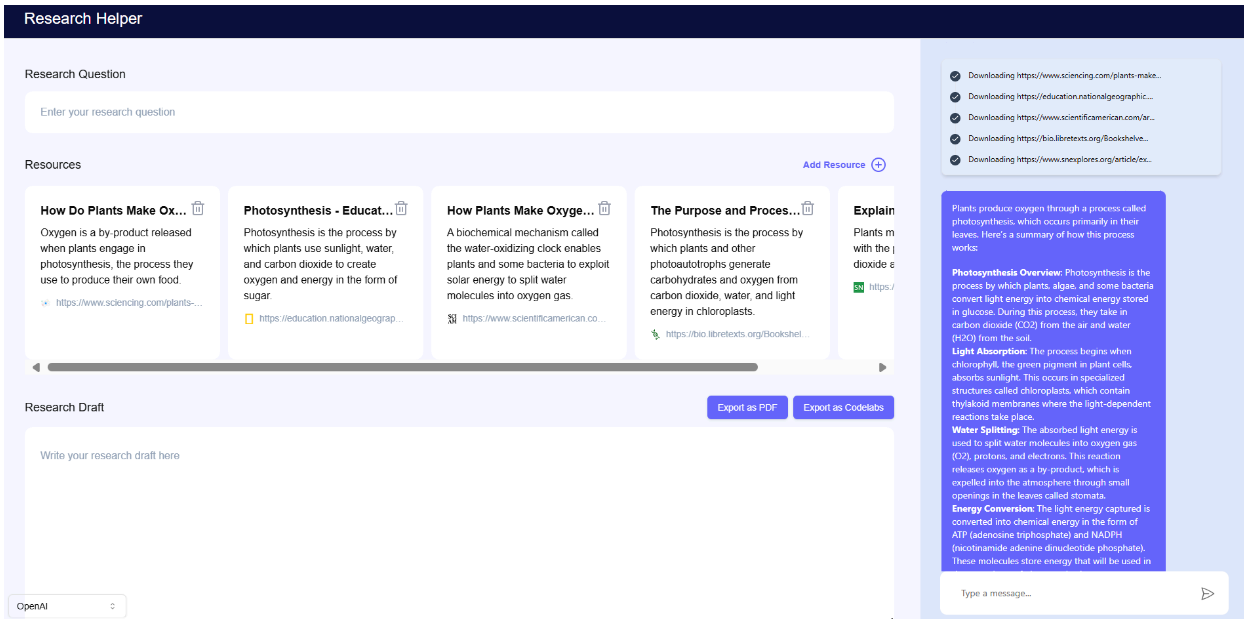
2. Calculation of U.S. Corporate Pension Plan Liabilities

The calculation of U.S. corporate pension plan liabilities is primarily based on the yields of high-quality corporate bonds. This approach ensures that the liabilities are valued in a manner that reflects the market conditions of safe investments.

Image 4: Asking a Question Outside of Existing Publications

In this image, the user has asked a question that goes beyond the information available in the existing publications. As a result, the system initiates an external search to gather relevant resources. The "Resources" section now displays various external articles related to the query, each summarized and with clickable links for deeper exploration.

On the right side, the assistant has provided a comprehensive response based on the aggregated information from the retrieved articles. This functionality demonstrates the tool's capability to extend beyond internal resources and gather real-time information, enhancing the research process with broader insights and context. This external search feature helps users get a comprehensive overview when in-house publications don't have the required information.


Research Helper

Research Question

Enter your research question

Resources | Add Resource

How Do Plants Make Ox…
Oxygen is a by-product released when plants engage in photosynthesis, the process they use to produce their own food.
https://www.sciencing.com/plants-…

Photosynthesis - Educat…
Photosynthesis is the process by which plants use sunlight, water, and carbon dioxide to create oxygen and energy in the form of sugar.
https://education.nationalgeograp…

How Plants Make Oxyge…
A biochemical mechanism called the water-oxidizing clock enables plants and some bacteria to exploit solar energy to split water molecules into oxygen gas.
https://www.scientificamerican.co…

The Purpose and Proces…
Photosynthesis is the process by which plants and other photoautotrophs generate carbohydrates and oxygen from carbon dioxide, water, and light energy in chloroplasts.
https://bio.libretexts.org/Bookshel…

Explain
Plants m
with the
dioxide a
https:/

Research Draft | Export as PDF | Export as Codelabs

Write your research draft here

OpenAI

Downloading https://www.sciencing.com/plants-make…
Downloading https://education.nationalgeographic….
Downloading https://www.scientificamerican.com/ar…
Downloading https://bio.libretexts.org/Bookshelve…
Downloading https://www.snexplores.org/article/ex…

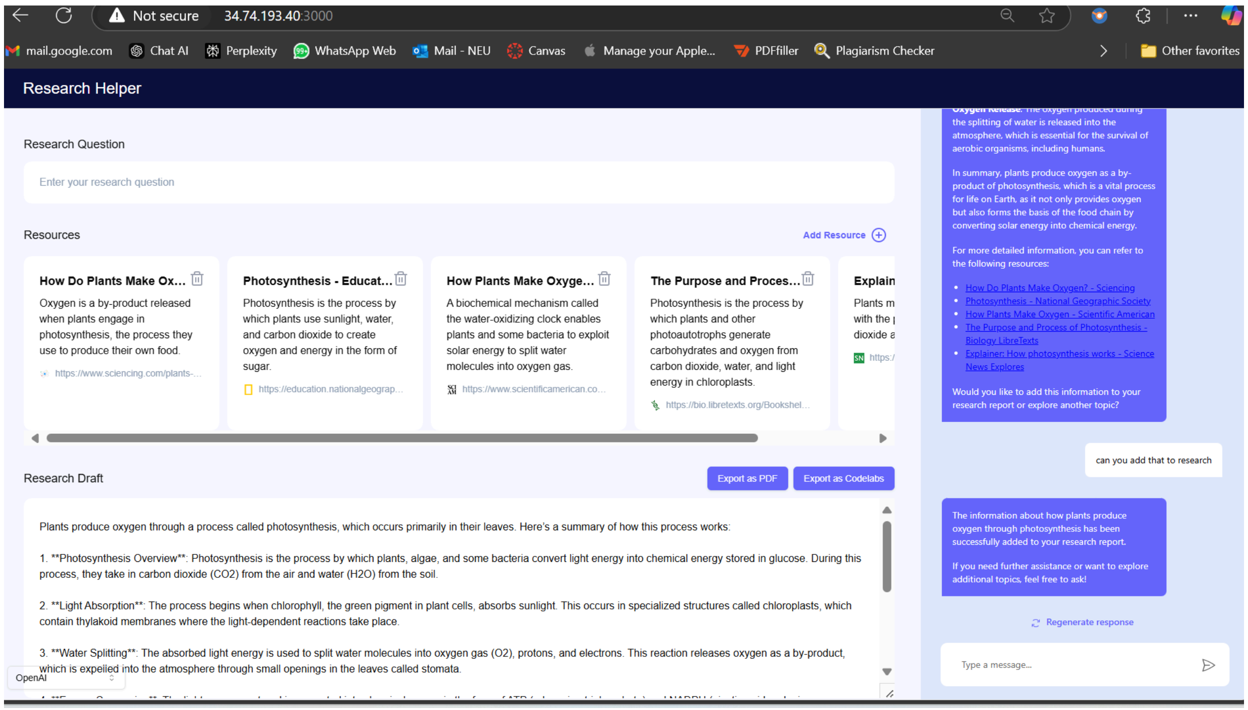
Plants produce oxygen through a process called photosynthesis, which occurs primarily in their leaves. Here's a summary of how this process works:

Photosynthesis Overview: Photosynthesis is the process by which plants, algae, and some bacteria convert light energy into chemical energy stored in glucose. During this process, they take in carbon dioxide (CO2) from the air and water (H2O) from the soil.
Light Absorption: The process begins when chlorophyll, the green pigment in plant cells, absorbs sunlight. This occurs in specialized structures called chloroplasts, which contain thylakoid membranes where the light-dependent reactions take place.
Water Splitting: The absorbed light energy is used to split water molecules into oxygen gas (O2), protons, and electrons. This reaction releases oxygen as a by-product, which is expelled into the atmosphere through small openings in the leaves called stomata.
Energy Conversion: The light energy captured is converted into chemical energy in the form of ATP (adenosine triphosphate) and NADPH (nicotinamide adenine dinucleotide phosphate). These molecules store energy that will be used in

Type a message...

Image 5: Adding Information to Research Draft

In this screenshot, the user has asked the assistant to add the retrieved information on photosynthesis to the research draft. The assistant has successfully appended the details to the "Research Draft" section, where it provides a structured summary of how plants produce oxygen through photosynthesis, broken down into key points.



Research Helper

Research Question

Enter your research question

Resources | Add Resource

How Do Plants Make Ox…
Oxygen is a by-product released when plants engage in photosynthesis, the process they use to produce their own food.
https://www.sciencing.com/plants-…

Photosynthesis - Educat…
Photosynthesis is the process by which plants use sunlight, water, and carbon dioxide to create oxygen and energy in the form of sugar.
https://education.nationalgeograp…

How Plants Make Oxyge…
A biochemical mechanism called the water-oxidizing clock enables plants and some bacteria to exploit solar energy to split water molecules into oxygen gas.
https://www.scientificamerican.co…

The Purpose and Proces…
Photosynthesis is the process by which plants and other photoautotrophs generate carbohydrates and oxygen from carbon dioxide, water, and light energy in chloroplasts.
https://bio.libretexts.org/Bookshel…

Explain
Plants m
with the
dioxide a
https:/

Research Draft | Export as PDF | Export as Codelabs

Plants produce oxygen through a process called photosynthesis, which occurs primarily in their leaves. Here's a summary of how this process works:

1. **Photosynthesis Overview**: Photosynthesis is the process by which plants, algae, and some bacteria convert light energy into chemical energy stored in glucose. During this process, they take in carbon dioxide (CO2) from the air and water (H2O) from the soil.

2. **Light Absorption**: The process begins when chlorophyll, the green pigment in plant cells, absorbs sunlight. This occurs in specialized structures called chloroplasts, which contain thylakoid membranes where the light-dependent reactions take place.

3. **Water Splitting**: The absorbed light energy is used to split water molecules into oxygen gas (O2), protons, and electrons. This reaction releases oxygen as a by-product, which is expelled into the atmosphere through small openings in the leaves called stomata.

OpenAI

the splitting of water is released into the atmosphere, which is essential for the survival of aerobic organisms, including humans.

In summary, plants produce oxygen as a by-product of photosynthesis, which is a vital process for life on Earth, as it not only provides oxygen but also forms the basis of the food chain by converting solar energy into chemical energy.

For more detailed information, you can refer to the following resources:

- How Do Plants Make Oxygen? - Sciencing
- Photosynthesis - National Geographic Society
- How Plants Make Oxygen - Scientific American
- The Purpose and Process of Photosynthesis - Biology LibreTexts
- Explainer: How photosynthesis works - Science News Explores

Would you like to add this information to your research report or explore another topic?

can you add that to research

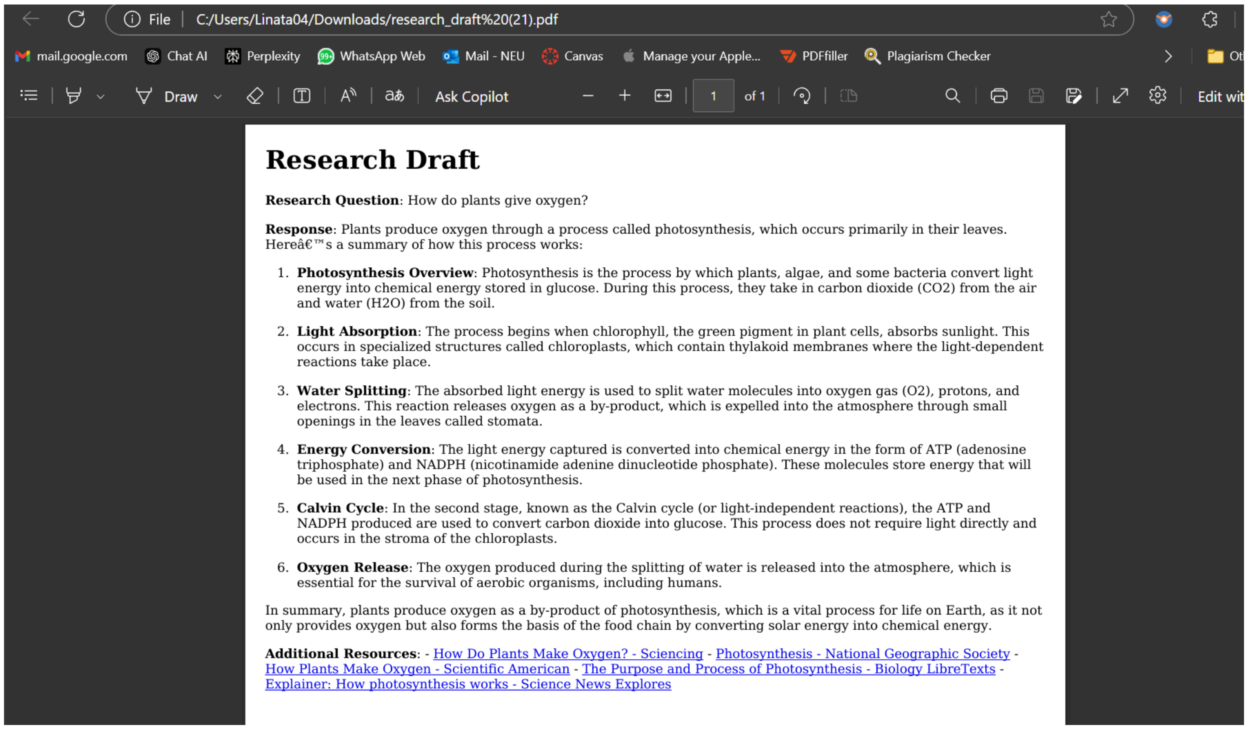
The information about how plants produce oxygen through photosynthesis has been successfully added to your research report.

If you need further assistance or want to explore additional topics, feel free to ask!

Regenerate response

Type a message...

This screenshot displays the exported PDF of the research draft. The draft also provides additional resources as clickable links, allowing the reader to explore more in-depth information on related topics. This exported document serves as a formalized output, ideal for presentations, reports, or further research reference, demonstrating the tool's capability to produce publication-ready content. In addition to PDF export, this tool allows you to generate a structured Codelabs document from the research draft.



Research Draft

Research Question: How do plants give oxygen?

Response: Plants produce oxygen through a process called photosynthesis, which occurs primarily in their leaves. Hereâ€™s a summary of how this process works:

1. **Photosynthesis Overview**: Photosynthesis is the process by which plants, algae, and some bacteria convert light energy into chemical energy stored in glucose. During this process, they take in carbon dioxide (CO2) from the air and water (H2O) from the soil.
2. **Light Absorption**: The process begins when chlorophyll, the green pigment in plant cells, absorbs sunlight. This occurs in specialized structures called chloroplasts, which contain thylakoid membranes where the light-dependent reactions take place.
3. **Water Splitting**: The absorbed light energy is used to split water molecules into oxygen gas (O2), protons, and electrons. This reaction releases oxygen as a by-product, which is expelled into the atmosphere through small openings in the leaves called stomata.
4. **Energy Conversion**: The light energy captured is converted into chemical energy in the form of ATP (adenosine triphosphate) and NADPH (nicotinamide adenine dinucleotide phosphate). These molecules store energy that will be used in the next phase of photosynthesis.
5. **Calvin Cycle**: In the second stage, known as the Calvin cycle (or light-independent reactions), the ATP and NADPH produced are used to convert carbon dioxide into glucose. This process does not require light directly and occurs in the stroma of the chloroplasts.
6. **Oxygen Release**: The oxygen produced during the splitting of water is released into the atmosphere, which is essential for the survival of aerobic organisms, including humans.

In summary, plants produce oxygen as a by-product of photosynthesis, which is a vital process for life on Earth, as it not only provides oxygen but also forms the basis of the food chain by converting solar energy into chemical energy.

Additional Resources: - How Do Plants Make Oxygen? - Sciencing - Photosynthesis - National Geographic Society - How Plants Make Oxygen - Scientific American - The Purpose and Process of Photosynthesis - Biology LibreTexts - Explainer: How photosynthesis works - Science News Explores

Application Workflow (Data Engineering Work + Code Explanation)

1. Full Data Flow: From Frontend Input to Backend Processing and Final Output

Frontend Input:

- **User Actions**: The user interacts with the UI by entering a research question, adding resources, or asking specific questions related to preloaded publications.
- **State Management**: The application's state (managed by **CopilotKit** and **React hooks**) ensures that each interaction is processed and maintained per user session. This includes handling the current research question, selected resources, and logs of the ongoing processes.

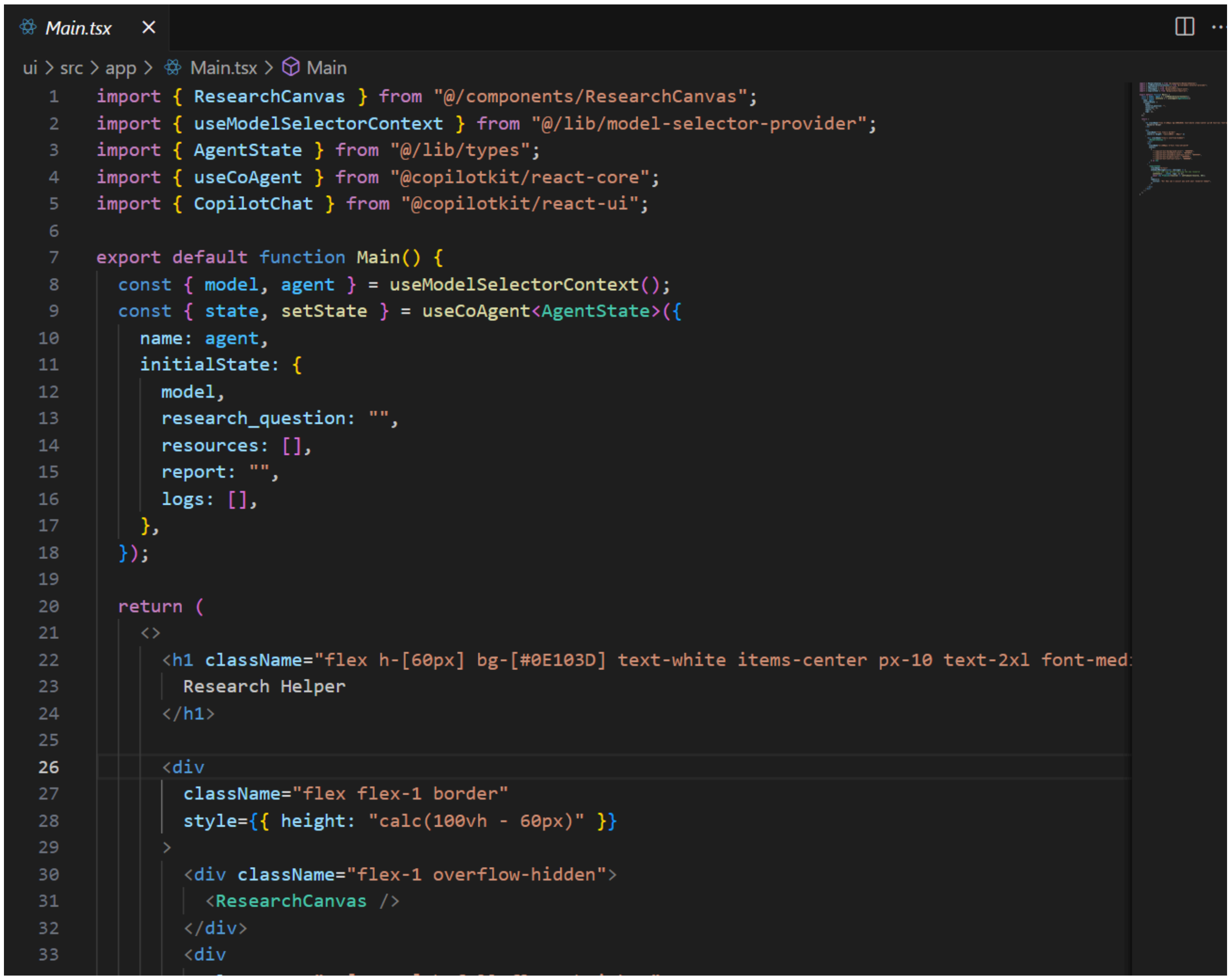
Data Flow from Frontend to Backend:

- **API Requests**: When a user asks a question or requests a resource addition, the frontend sends an API request to the backend using the configured endpoints.

- **Backend Processing**: The backend processes the request based on the specified endpoint:
    - **Document Retrieval**: If the question is related to a publication, it checks the indexed documents first.
    - **Web Search**: If there's no match in the publications, it initiates an agent search to retrieve relevant resources online, using APIs integrated into the backend.
- **Response**: The backend returns either a document-based answer or web-based search results. This response is rendered in the chat window for the user to view, with options to add specific findings to the research notes.

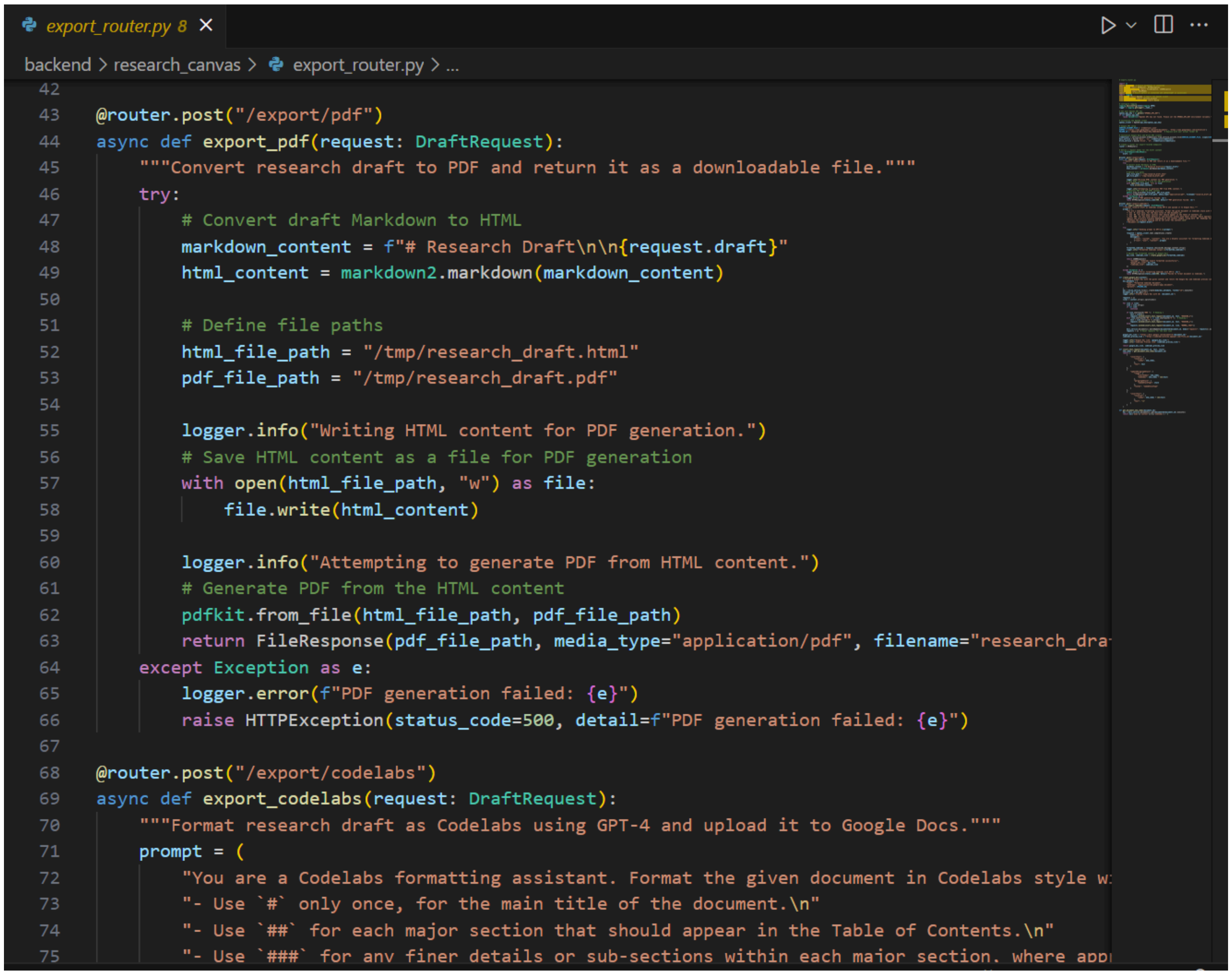
Final Output:

- **Research Draft Generation**: Users can compile all relevant responses and notes into the research draft section. Once finalized, they can export this draft as either a **PDF** or **Codelabs** document for easy reference or sharing.



```tsx
import { ResearchCanvas } from "@/components/ResearchCanvas";
import { useModelSelectorContext } from "@/lib/model-selector-provider";
import { AgentState } from "@/lib/types";
import { useCoAgent } from "@copilotkit/react-core";
import { CopilotChat } from "@copilotkit/react-ui";

export default function Main() {
  const { model, agent } = useModelSelectorContext();
  const { state, setState } = useCoAgent<AgentState>({
    name: agent,
    initialState: {
      model,
      research_question: "",
      resources: [],
      report: "",
      logs: [],
    },
  });

  return (
    <>
      <h1 className="flex h-[60px] bg-[#0E103D] text-white items-center px-10 text-2xl font-med
        Research Helper
      </h1>

      <div
        className="flex flex-1 border"
        style={{ height: "calc(100vh - 60px)" }}
      >
        <div className="flex-1 overflow-hidden">
          <ResearchCanvas />
        </div>
        <div
```

The Main component integrates ResearchCanvas for managing research workflows and CopilotChat for AI-powered assistance. It initializes the app state using useCoAgent, managing fields like research_question and logs. The layout features a resource management area and a chat assistant, enabling seamless interaction with dynamic state updates and AI support.



```python

@router.post("/export/pdf")
async def export_pdf(request: DraftRequest):
    """Convert research draft to PDF and return it as a downloadable file."""
    try:
        # Convert draft Markdown to HTML
        markdown_content = f"# Research Draft\n\n{request.draft}"
        html_content = markdown2.markdown(markdown_content)

        # Define file paths
        html_file_path = "/tmp/research_draft.html"
        pdf_file_path = "/tmp/research_draft.pdf"

        logger.info("Writing HTML content for PDF generation.")
        # Save HTML content as a file for PDF generation
        with open(html_file_path, "w") as file:
            file.write(html_content)

        logger.info("Attempting to generate PDF from HTML content.")
        # Generate PDF from the HTML content
        pdfkit.from_file(html_file_path, pdf_file_path)
        return FileResponse(pdf_file_path, media_type="application/pdf", filename="research_dra
    except Exception as e:
        logger.error(f"PDF generation failed: {e}")
        raise HTTPException(status_code=500, detail=f"PDF generation failed: {e}")

@router.post("/export/codelabs")
async def export_codelabs(request: DraftRequest):
    """Format research draft as Codelabs using GPT-4 and upload it to Google Docs."""
    prompt = (
        "You are a Codelabs formatting assistant. Format the given document in Codelabs style w
        "- Use `#` only once, for the main title of the document.\n"
        "- Use `##` for each major section that should appear in the Table of Contents.\n"
        "- Use `###` for any finer details or sub-sections within each major section, where app
```

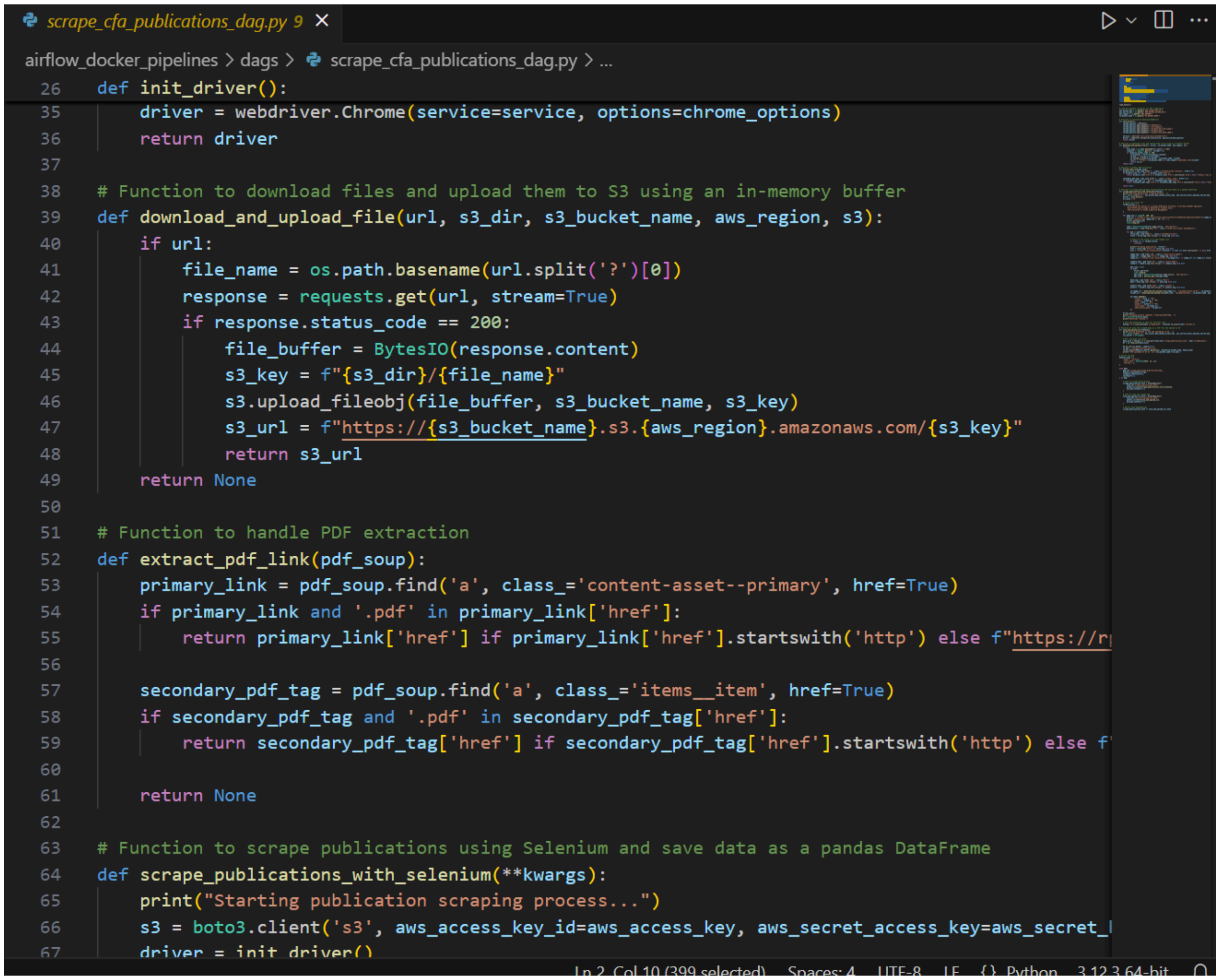
The export_router.py script provides FastAPI endpoints to export research drafts as PDFs or Codelabs documents. The /export/pdf endpoint converts Markdown drafts to PDFs using markdown2 and pdfkit, while /export/codelabs uses GPT-4 to format drafts in Codelabs style and uploads them to Google Docs via the Drive API. It returns links to the Google Doc and its Codelabs preview. The script integrates OpenAI and Google APIs for seamless formatting, document creation, and sharing, with robust logging and error handling throughout.

2. Pipeline Orchestration with Airflow DAGs (Data Extraction, Transformation, and Loading)

The data engineering pipeline is orchestrated using **Apache Airflow**, which manages the **Extraction**, **Transformation**, and **Loading (ETL)** process across multiple DAGs. Here's a breakdown of each key pipeline:

- **DAG 1**: scrape_cfa_publications_dag.py

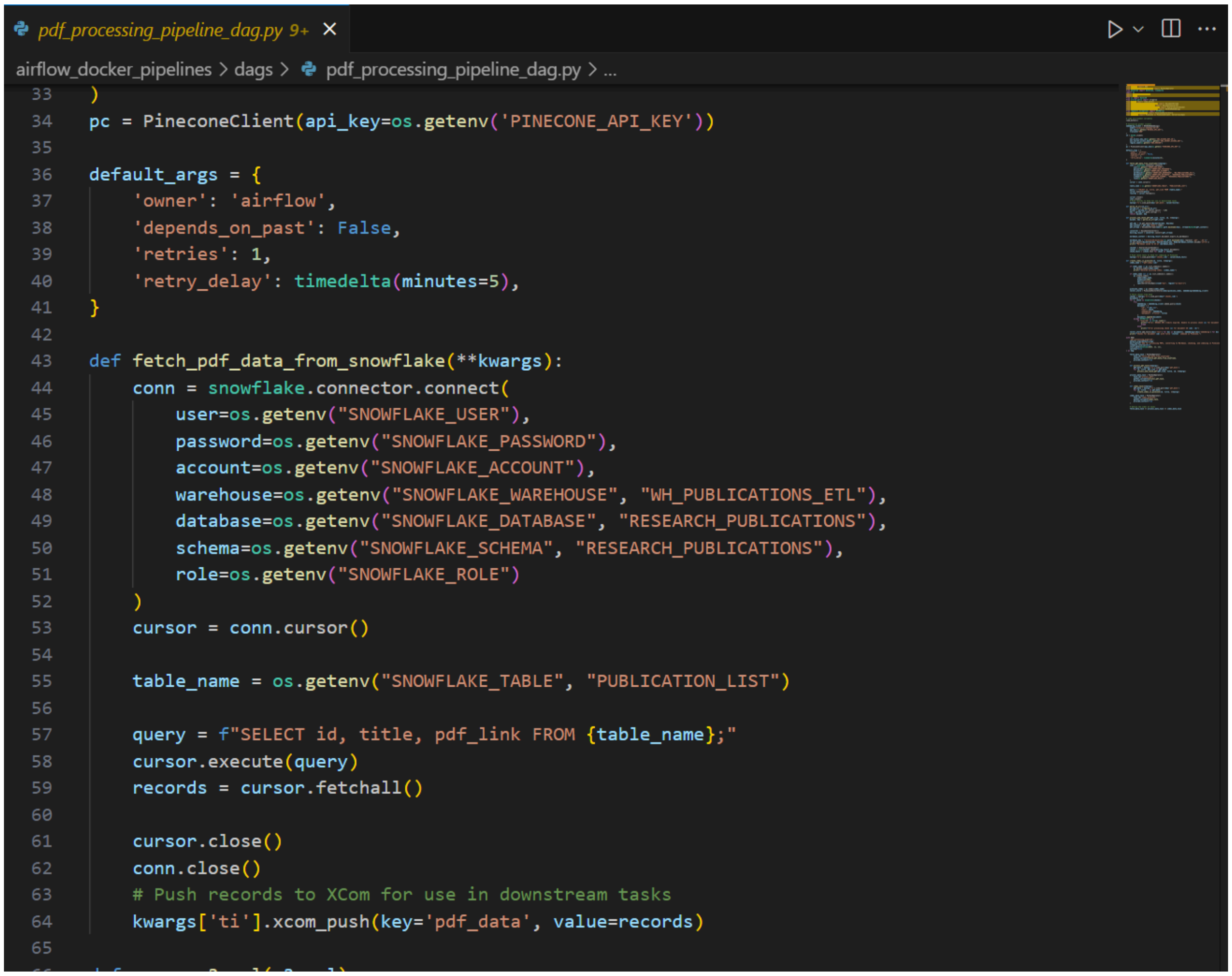
    - **Purpose**: Scrapes publications from the CFA website, extracts key metadata (title, summary, publication date, etc.), and uploads this data to Amazon S3.
    - **Key Task Scheduling**:
        - scrape_task: A task that scrapes data from the CFA site.
        - upload_task: Uploads extracted metadata and publications to S3 for storage.
    - **Data Flow**: The scraped data is stored as a CSV in S3, which the other pipelines can access.
- **DAG 2**: snowflake_setup_dag.py
    - **Purpose**: Sets up a **Snowflake** database, creating necessary tables and schemas for storing publication data.
    - **Key Task Scheduling**:
        - create_table_task: Creates tables for storing publication metadata.
        - config_task: Configures the database and sets permissions.
    - **Data Flow**: This DAG prepares the Snowflake environment for storing and querying publication data.
- **DAG 3**: snowflake_load_dag.py
    - **Purpose**: Loads publication metadata from the S3 CSV into the Snowflake database, allowing efficient querying.
    - **Key Task Scheduling**:
        - download_from_s3_task: Downloads data from S3.
        - load_to_snowflake_task: Loads data into Snowflake.
    - **Data Flow**: Transforms data from the S3 bucket and imports it into Snowflake.
- **DAG 4**: pdf_processing_pipeline_dag.py
    - **Purpose**: Processes PDFs by converting them to Markdown, chunking the content, and indexing in Pinecone for quick retrieval.
    - **Key Task Scheduling**:
        - pdf_to_md_task: Converts PDFs to Markdown.
        - chunk_and_index_task: Splits content into chunks and indexes in Pinecone.
    - **Data Flow**: Processed document data is stored in Pinecone, enabling quick retrieval based on document context.


scrape_cfa_publications_dag.py 9
airflow_docker_pipelines > dags > scrape_cfa_publications_dag.py > ...

```python
def init_driver():
    driver = webdriver.Chrome(service=service, options=chrome_options)
    return driver

# Function to download files and upload them to S3 using an in-memory buffer
def download_and_upload_file(url, s3_dir, s3_bucket_name, aws_region, s3):
    if url:
        file_name = os.path.basename(url.split('?')[0])
        response = requests.get(url, stream=True)
        if response.status_code == 200:
            file_buffer = BytesIO(response.content)
            s3_key = f"{s3_dir}/{file_name}"
            s3.upload_fileobj(file_buffer, s3_bucket_name, s3_key)
            s3_url = f"https://{s3_bucket_name}.s3.{aws_region}.amazonaws.com/{s3_key}"
            return s3_url
    return None

# Function to handle PDF extraction
def extract_pdf_link(pdf_soup):
    primary_link = pdf_soup.find('a', class_='content-asset--primary', href=True)
    if primary_link and '.pdf' in primary_link['href']:
        return primary_link['href'] if primary_link['href'].startswith('http') else f"https://r

    secondary_pdf_tag = pdf_soup.find('a', class_='items__item', href=True)
    if secondary_pdf_tag and '.pdf' in secondary_pdf_tag['href']:
        return secondary_pdf_tag['href'] if secondary_pdf_tag['href'].startswith('http') else f'

    return None

# Function to scrape publications using Selenium and save data as a pandas DataFrame
def scrape_publications_with_selenium(**kwargs):
    print("Starting publication scraping process...")
    s3 = boto3.client('s3', aws_access_key_id=aws_access_key, aws_secret_access_key=aws_secret_
    driver = init_driver()
```

Ln 2, Col 10 (399 selected) Spaces: 4 UTF-8 LF {} Python 3.12.3 64-bit

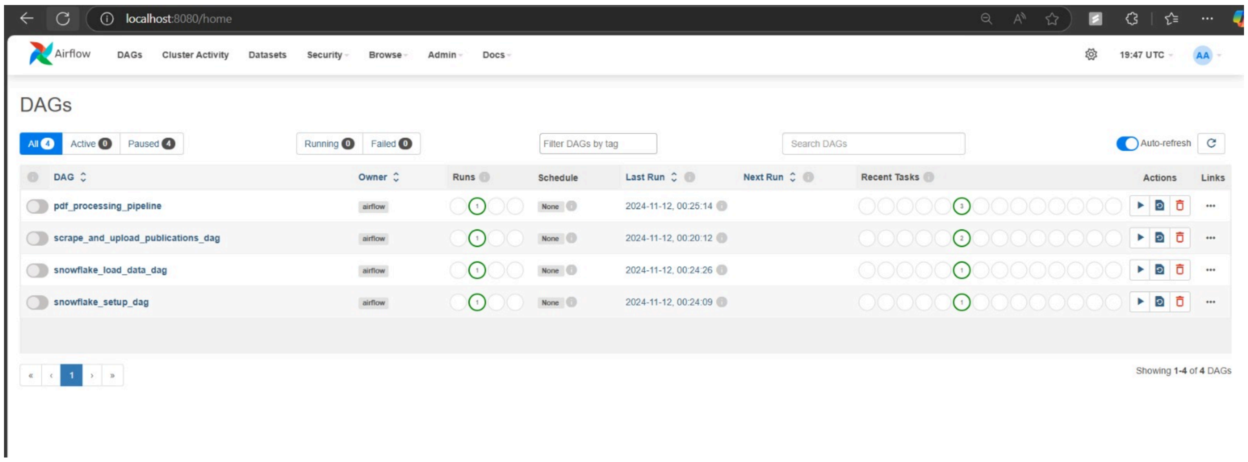
The scrape_cfa_publications_dag.py defines an Airflow DAG to scrape publication data from the CFA Institute website using Selenium and BeautifulSoup, process it, and upload it to AWS S3. It extracts metadata, cover images, and PDFs, saves the data as a CSV using Pandas, and uploads it to S3. The workflow involves two tasks: scraping publications and saving/uploading the CSV, executed sequentially with data passed via XCom.


pdf_processing_pipeline_dag.py 9+

airflow_docker_pipelines > dags > pdf_processing_pipeline_dag.py > ...

```python
)
pc = PineconeClient(api_key=os.getenv('PINECONE_API_KEY'))

default_args = {
    'owner': 'airflow',
    'depends_on_past': False,
    'retries': 1,
    'retry_delay': timedelta(minutes=5),
}

def fetch_pdf_data_from_snowflake(**kwargs):
    conn = snowflake.connector.connect(
        user=os.getenv("SNOWFLAKE_USER"),
        password=os.getenv("SNOWFLAKE_PASSWORD"),
        account=os.getenv("SNOWFLAKE_ACCOUNT"),
        warehouse=os.getenv("SNOWFLAKE_WAREHOUSE", "WH_PUBLICATIONS_ETL"),
        database=os.getenv("SNOWFLAKE_DATABASE", "RESEARCH_PUBLICATIONS"),
        schema=os.getenv("SNOWFLAKE_SCHEMA", "RESEARCH_PUBLICATIONS"),
        role=os.getenv("SNOWFLAKE_ROLE")
    )
    cursor = conn.cursor()

    table_name = os.getenv("SNOWFLAKE_TABLE", "PUBLICATION_LIST")

    query = f"SELECT id, title, pdf_link FROM {table_name};"
    cursor.execute(query)
    records = cursor.fetchall()

    cursor.close()
    conn.close()
    # Push records to XCom for use in downstream tasks
    kwargs['ti'].xcom_push(key='pdf_data', value=records)

```

The pdf_processing_pipeline_dag.py defines an Airflow DAG to process, chunk, and index PDFs for efficient retrieval. It fetches metadata from Snowflake, downloads PDFs from S3, converts them to Markdown, chunks the content using Docling, and stores it back in S3. The chunks are embedded with NVIDIA's embeddings API and indexed in Pinecone for searchability. Tasks are executed sequentially: fetching data, processing PDFs, and indexing chunks, integrating Snowflake, AWS S3, NVIDIA, and Pinecone for an automated document processing workflow.



DAGs

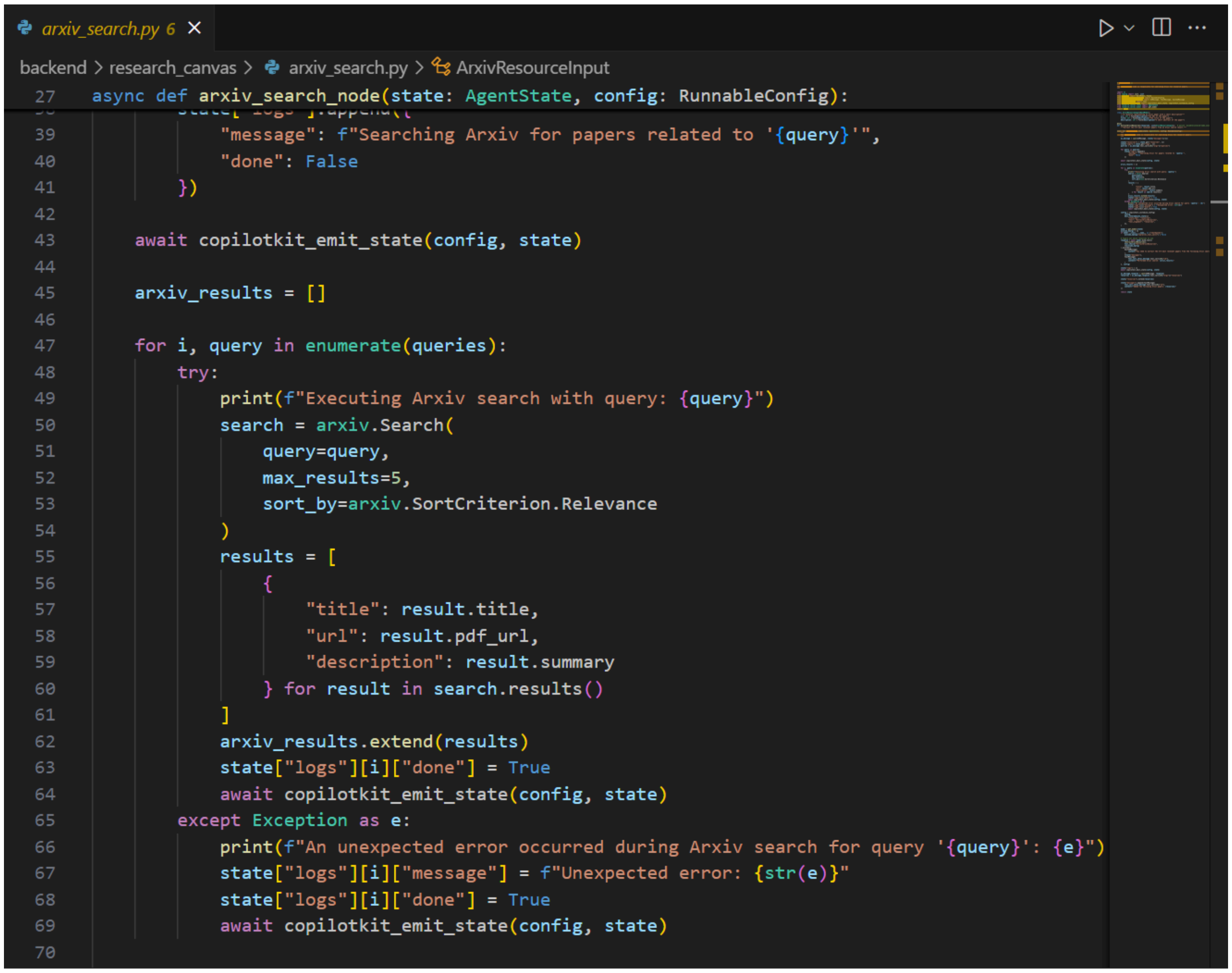
DAG	Owner	Runs	Schedule	Last Run	Next Run	Recent Tasks	Actions	Links
pdf_processing_pipeline	airflow	1	None	2024-11-12, 00:25:14		3		...
scrape_and_upload_publications_dag	airflow	1	None	2024-11-12, 00:20:12		2		...
snowflake_load_data_dag	airflow	1	None	2024-11-12, 00:24:26		1		...
snowflake_setup_dag	airflow	1	None	2024-11-12, 00:24:09		1		...

Showing 1-4 of 4 DAGs

3. Backend Architecture and API Layer

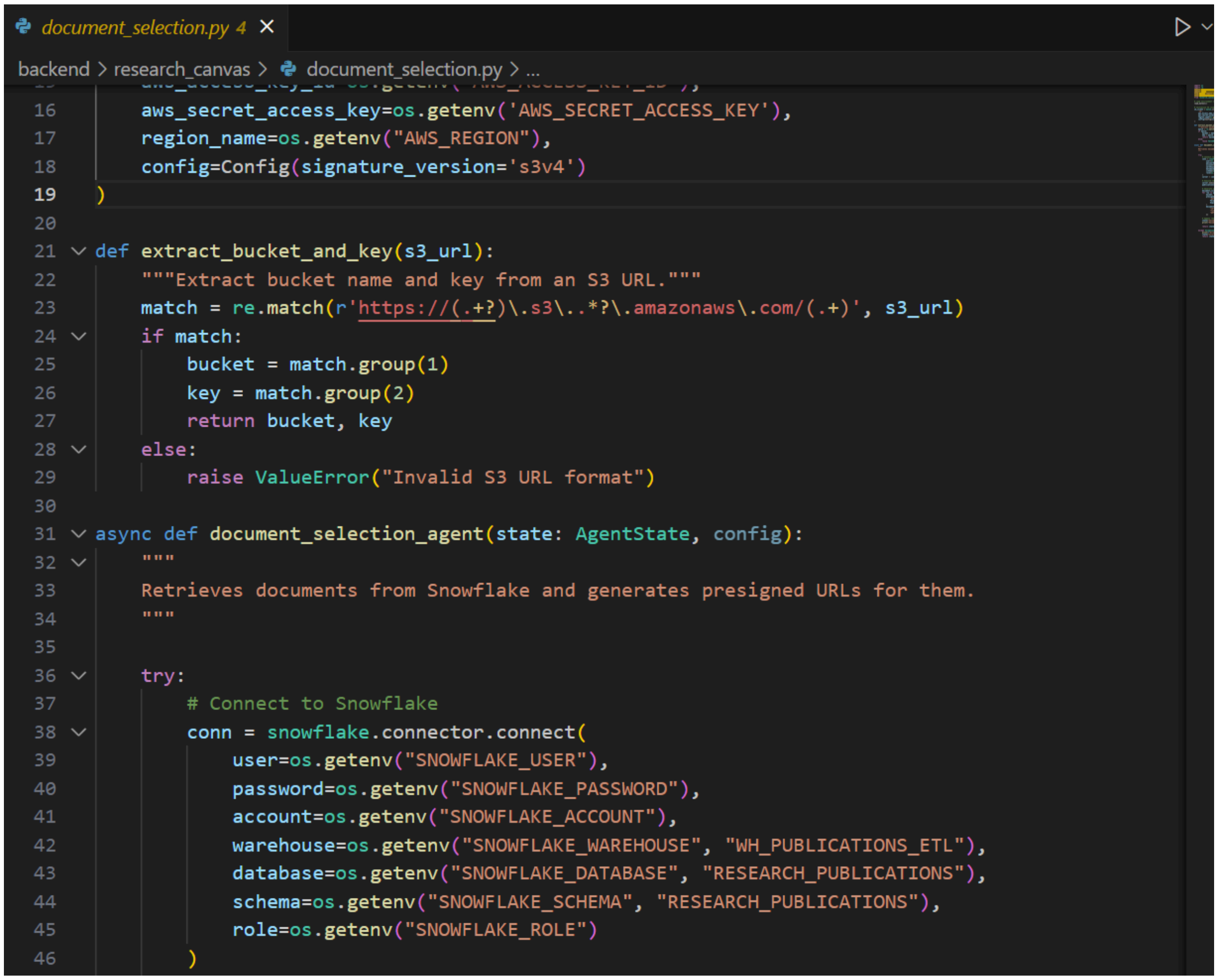
The backend is built with **FastAPI**, providing an API layer that integrates with AWS S3, Snowflake, Pinecone, and NVIDIA's RAG services. Here's an overview of its main components:

- **API Endpoints**:
  - **Document Retrieval** (document_selection.py): Fetches publications stored in Snowflake and provides pre-signed URLs for S3-hosted PDFs.
  - **Q/A and RAG-Based Responses** (rag_router.py): Handles queries and searches for document content using NVIDIA's RAG model integrated with Pinecone.
  - **Arxiv and Web Search** (arxiv_search.py, search.py): Fetches academic articles and broader web content to expand research resources.
  - **Export Functions** (export_router.py): Exports research drafts to either PDF or Codelabs format.
- **Database Interactions**:
  - **Snowflake**: Stores publication metadata, and FastAPI endpoints query it to retrieve document information or save processed data.
  - **AWS S3**: Stores raw data and processed PDFs; the backend fetches and provides links to this data as required.
  - **Pinecone**: Hosts the indexed content from PDFs, enabling RAG model queries to retrieve relevant document content.


arxiv_search.py 6
backend > research_canvas > arxiv_search.py > ArxivResourceInput

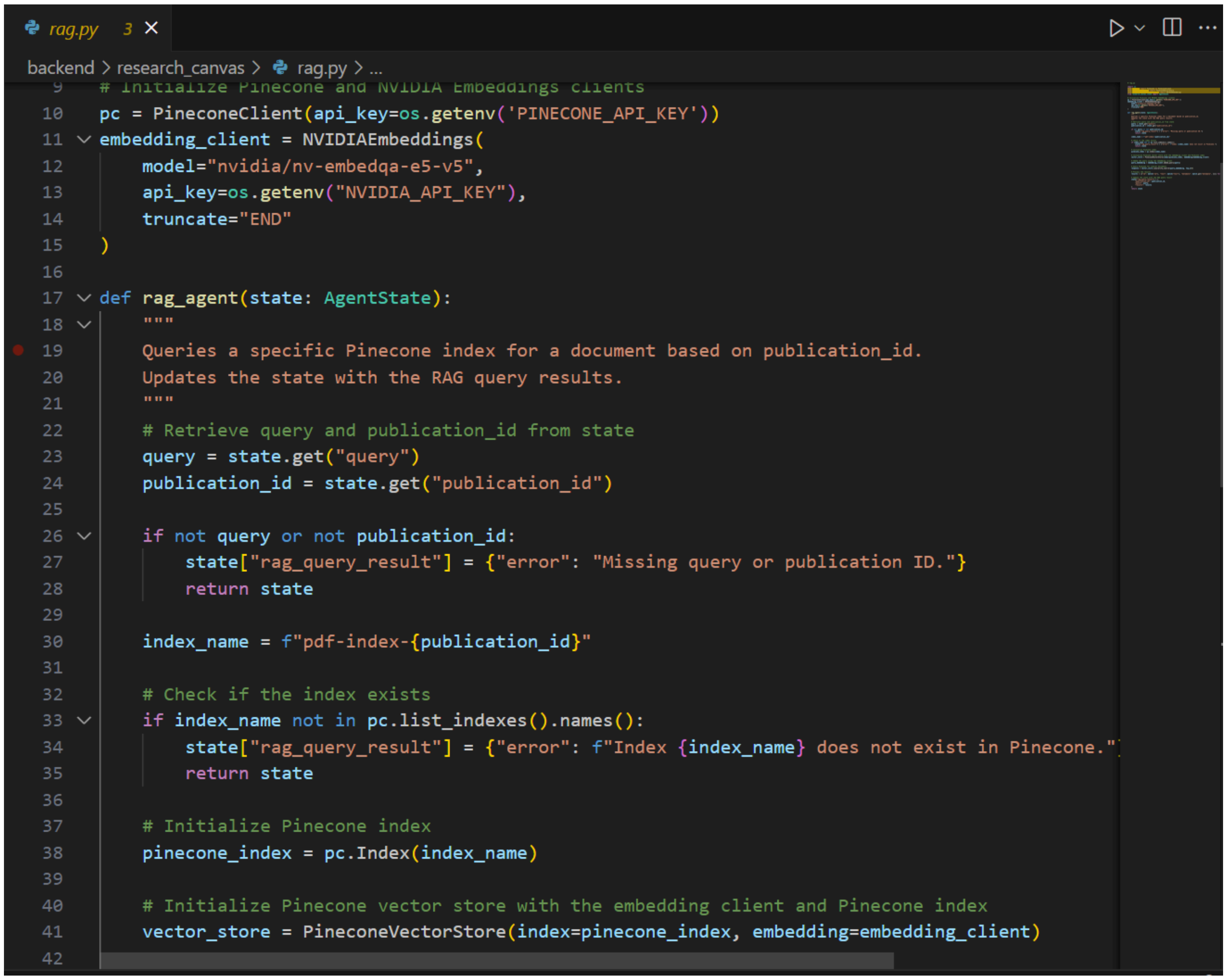
```python
async def arxiv_search_node(state: AgentState, config: RunnableConfig):
            "message": f"Searching Arxiv for papers related to '{query}'",
            "done": False
        })

    await copilotkit_emit_state(config, state)

    arxiv_results = []

    for i, query in enumerate(queries):
        try:
            print(f"Executing Arxiv search with query: {query}")
            search = arxiv.Search(
                query=query,
                max_results=5,
                sort_by=arxiv.SortCriterion.Relevance
            )
            results = [
                {
                    "title": result.title,
                    "url": result.pdf_url,
                    "description": result.summary
                } for result in search.results()
            ]
            arxiv_results.extend(results)
            state["logs"][i]["done"] = True
            await copilotkit_emit_state(config, state)
        except Exception as e:
            print(f"An unexpected error occurred during Arxiv search for query '{query}': {e}")
            state["logs"][i]["message"] = f"Unexpected error: {str(e)}"
            state["logs"][i]["done"] = True
            await copilotkit_emit_state(config, state)
```

The arxiv_search.py script facilitates targeted research by integrating Arxiv searches into a research assistant. It processes user queries, retrieves up to 5 relevant papers per query using the arxiv library, and filters results through the ExtractArxivResources tool. It updates logs, resources, and messages in the application state with copilotkit_emit_state while handling errors robustly, enabling efficient Arxiv-based research workflows.



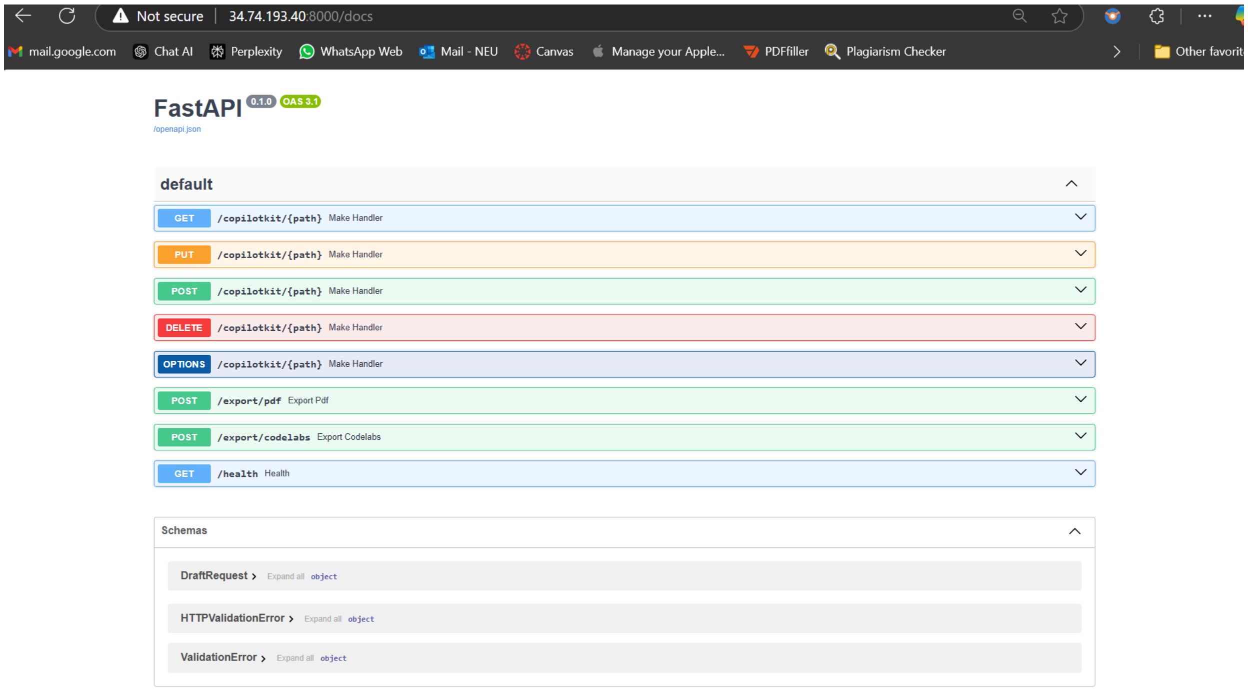
```python
    aws_secret_access_key=os.getenv('AWS_SECRET_ACCESS_KEY'),
    region_name=os.getenv("AWS_REGION"),
    config=Config(signature_version='s3v4')
)

def extract_bucket_and_key(s3_url):
    """Extract bucket name and key from an S3 URL."""
    match = re.match(r'https://(.+?)\.s3\..*?\.amazonaws\.com/(.+)', s3_url)
    if match:
        bucket = match.group(1)
        key = match.group(2)
        return bucket, key
    else:
        raise ValueError("Invalid S3 URL format")

async def document_selection_agent(state: AgentState, config):
    """
    Retrieves documents from Snowflake and generates presigned URLs for them.
    """

    try:
        # Connect to Snowflake
        conn = snowflake.connector.connect(
            user=os.getenv("SNOWFLAKE_USER"),
            password=os.getenv("SNOWFLAKE_PASSWORD"),
            account=os.getenv("SNOWFLAKE_ACCOUNT"),
            warehouse=os.getenv("SNOWFLAKE_WAREHOUSE", "WH_PUBLICATIONS_ETL"),
            database=os.getenv("SNOWFLAKE_DATABASE", "RESEARCH_PUBLICATIONS"),
            schema=os.getenv("SNOWFLAKE_SCHEMA", "RESEARCH_PUBLICATIONS"),
            role=os.getenv("SNOWFLAKE_ROLE")
        )
```

The document_selection_agent retrieves document metadata from Snowflake, generates secure presigned URLs for S3 PDFs, and updates the application state with the document list. It uses regex to extract S3 bucket and key details, ensures URLs are valid for 1 hour, and handles errors gracefully, enabling secure and efficient document access.



```python
# Initialize Pinecone and NVIDIA Embeddings clients
pc = PineconeClient(api_key=os.getenv('PINECONE_API_KEY'))
embedding_client = NVIDIAEmbeddings(
    model="nvidia/nv-embedqa-e5-v5",
    api_key=os.getenv("NVIDIA_API_KEY"),
    truncate="END"
)

def rag_agent(state: AgentState):
    """
    Queries a specific Pinecone index for a document based on publication_id.
    Updates the state with the RAG query results.
    """
    # Retrieve query and publication_id from state
    query = state.get("query")
    publication_id = state.get("publication_id")

    if not query or not publication_id:
        state["rag_query_result"] = {"error": "Missing query or publication ID."}
        return state

    index_name = f"pdf-index-{publication_id}"

    # Check if the index exists
    if index_name not in pc.list_indexes().names():
        state["rag_query_result"] = {"error": f"Index {index_name} does not exist in Pinecone."
        return state

    # Initialize Pinecone index
    pinecone_index = pc.Index(index_name)

    # Initialize Pinecone vector store with the embedding client and Pinecone index
    vector_store = PineconeVectorStore(index=pinecone_index, embedding=embedding_client)

```

The rag.py script implements a Retrieval-Augmented Generation (RAG) agent to query Pinecone indexes for document chunks based on user queries. It retrieves the query and publication_id from the state, generates embeddings using NVIDIA's embedding client, and performs a similarity search in the relevant Pinecone index. Top matches are processed and added to the state, with error handling for missing inputs or non-existent indexes.



FastAPI 0.1.0 OAS 3.1
/openapi.json

default

GET	/copilotkit/{path}	Make Handler
PUT	/copilotkit/{path}	Make Handler
POST	/copilotkit/{path}	Make Handler
DELETE	/copilotkit/{path}	Make Handler
OPTIONS	/copilotkit/{path}	Make Handler
POST	/export/pdf	Export Pdf
POST	/export/codelabs	Export Codelabs
GET	/health	Health

Schemas

DraftRequest > Expand all object

HTTPValidationError > Expand all object

ValidationError > Expand all object

References

Next.js Documentation: https://nextjs.org/docs

FastAPI Documentation: https://fastapi.tiangolo.com

Apache Airflow Documentation: https://airflow.apache.org/docs

Snowflake Documentation: https://docs.snowflake.com

AWS S3 - Getting Started: https://aws.amazon.com/s3/getting-started

Pinecone Vector Database Documentation: https://docs.pinecone.io

OpenAI API Documentation: https://platform.openai.com/docs

LangChain Documentation: https://langchain.readthedocs.io

Google Drive API Documentation: https://developers.google.com/drive

PDFKit Documentation: https://pdfkit.org