**GitHub Link:** <https://github.com/Lonelypheonix/IRC>

**IRC – Intelligent Research Companion**

**Research Paper RAG with Hierarchical Multi-Agent System.**

**Student ID:** 313834006  
**Student Name:** Kevin Lee   
**Course Name:** Recurrent Neural Networks  
**Instructor:** 黃仁竑

**Table of Contents**

1. [Executive Summary](#executive-summary)
2. [System Architecture](#system-architecture)
3. [Core Components](#core-components)
4. [Multi-Agent System](#multi-agent-system)
5. [External Services Integration](#external-services-integration)
6. [User Interface](#user-interface)
7. [Installation & Setup](#installation--setup)
8. [API Documentation](#api-documentation)
9. [Configuration](#configuration)
10. [Troubleshooting](#troubleshooting)
11. [Conclusion](#Conclusion)

**Executive Summary**

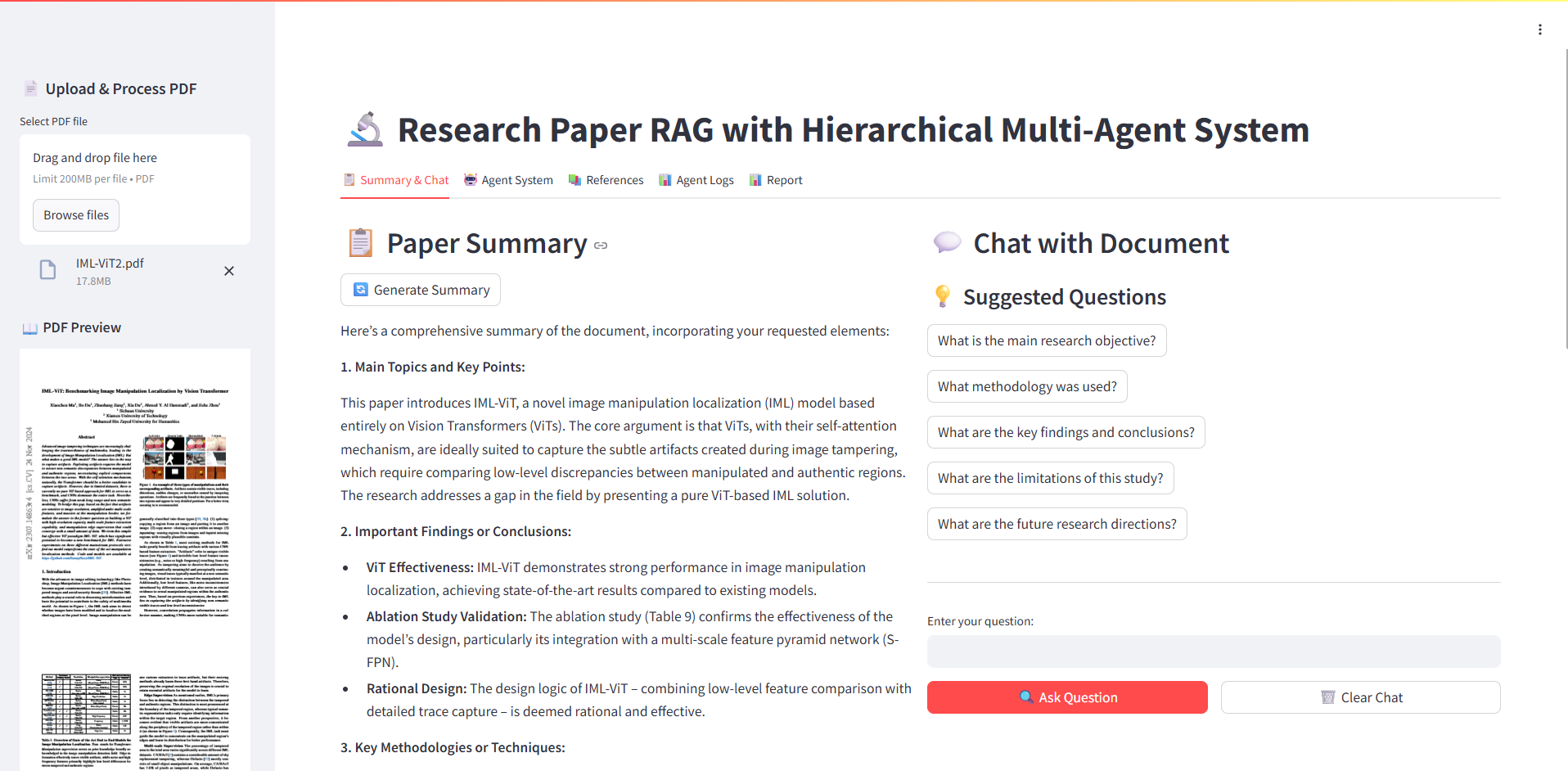
The IRC (Intelligent Research Companion) system is a sophisticated document analysis platform that combines advanced OCR technology, vector databases, and a hierarchical multi-agent system to provide comprehensive research paper analysis. The system enables users to upload PDF research papers, extract and analyze their content, automatically download and summarize referenced papers, and generate comprehensive reports.

**Key Capabilities**

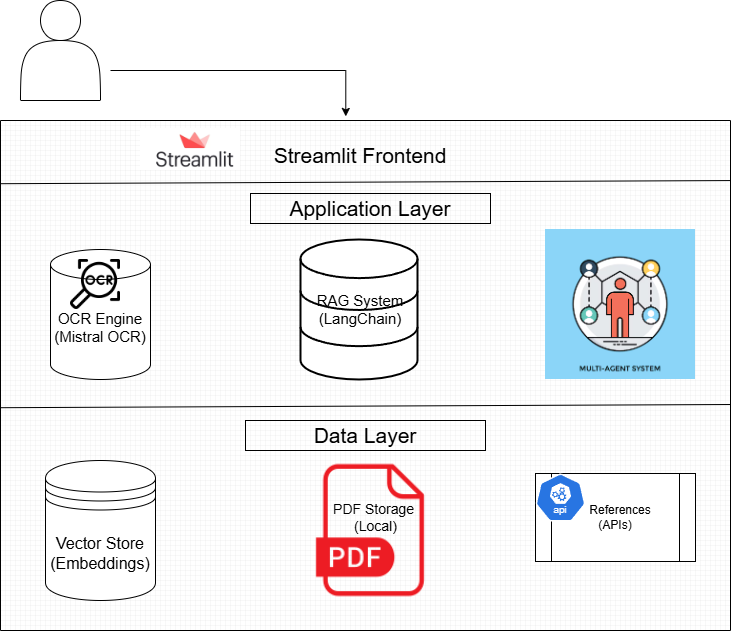
* **Advanced PDF Processing**: OCR extraction using Mistral AI with image analysis
* **Intelligent Document Chat**: RAG-based Q&A system with semantic search
* **Automated Reference Analysis**: Multi-agent system for downloading and summarizing cited papers
* **Comprehensive Reporting**: Automated generation of detailed research analysis reports
* **Multi-Source Integration**: Access to arXiv, Semantic Scholar, Google Scholar, and web sources

**System Architecture**

**System UI**

****

**High-Level Architecture**



**Technology Stack**

* **Frontend**: Streamlit web framework
* **Backend**: Python with LangChain
* **OCR**: Mistral AI OCR service
* **LLM Manager:** Ollama (Local model inference)
* **LLM Models:** Gemma 3 , Qwen2.5vl
* **Embeddings:** Nomic-embed-text (Vector representations)
* **Vector Database**: LangChain InMemoryVectorStore
* **Agents**: Custom multi-agent framework using LangChain
* **PDF Processing**: PyMuPDF (fitz), PIL
* **Web Scraping**: BeautifulSoup, requests
* **Search APIs**: arXiv, Semantic Scholar, DuckDuckGo

**Core Components**

**1. PDF Processing Engine**

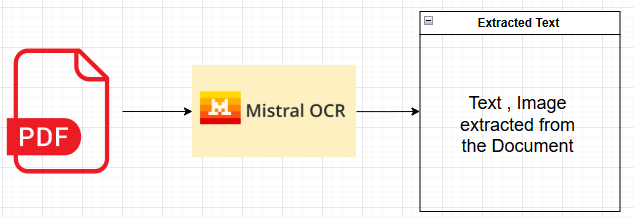
**Purpose**: Extract text and images from uploaded research papers

**Key Functions**:

* ocr\_pdf(pdf\_path): Performs OCR using Mistral AI
* extract\_images\_with\_pymupdf(): Extracts figures with captions
* get\_combined\_text(): Consolidates OCR results

**Features**:

* High-accuracy OCR with Mistral AI
* Automatic image extraction and cropping
* Vision-based figure description using Ollama
* Metadata preservation for extracted content



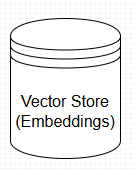
**2. Vector Database & RAG System**

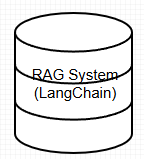
**Purpose**: Enable semantic search and question-answering on document content

**Key Functions**:

* create\_documents\_with\_metadata(): Chunks text with metadata
* index\_documents(): Stores chunks in vector database
* retrieve\_relevant\_docs(): Semantic similarity search
* answer\_question\_with\_rag(): Generates contextual answers

**Features**:

* Recursive text splitting with overlap
* Nomic embeddings for semantic representation
* Context-aware answer generation
* Source citation and relevance scoring



**3. Reference Extraction Engine**

**Purpose**: Automatically identify and extract paper citations

**Key Functions**:

* extract\_references(): Multi-strategy reference parsing
* extract\_paper\_sections(): Identifies paper structure
* Reference format support for multiple citation styles

**Extraction Strategies**:

1. Pattern-based section identification
2. Quoted title extraction
3. Year-based parsing
4. Author-title-journal format recognition

**4. Document Analysis & Reporting**

**Purpose**: Generate comprehensive analysis reports

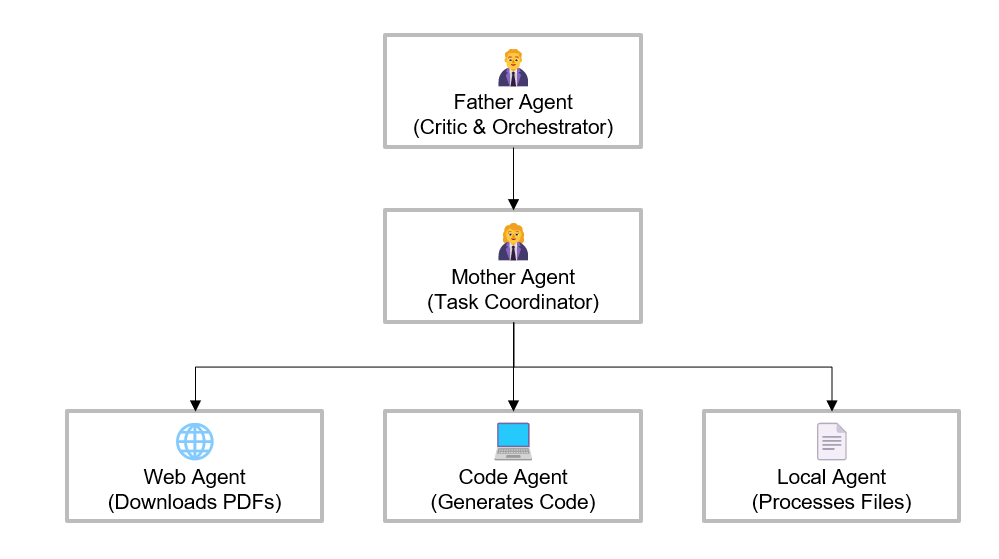
**Key Functions**:

* generate\_summary(): Creates paper summaries
* generate\_comprehensive\_report(): Full analysis report
* generate\_insights(): Cross-reference analysis
* save\_report(): Export to multiple formats

**Multi-Agent System**

**Agent Hierarchy**

The system implements a hierarchical multi-agent architecture for automated reference processing:



**Agent Descriptions**

**Father Agent**

* **Role**: Critic and orchestrator
* **Responsibilities**:
  + Extract references from main paper
  + Coordinate overall process
  + Evaluate final results
* **Function**: father\_agent\_func()

**Mother Agent**

* **Role**: Child agent coordinator
* **Responsibilities**:
  + Manage child agent execution
  + Collect and aggregate results
  + Handle inter-agent communication
* **Function**: mother\_agent\_func()

**Web Agent**

* **Role**: Paper downloader
* **Responsibilities**:
  + Search multiple academic sources
  + Download PDFs when available
  + Handle different file formats
* **Sources**: arXiv, Semantic Scholar, Google Scholar, DuckDuckGo
* **Function**: web\_agent\_func()

**Code Agent**

* **Role**: Code generator
* **Responsibilities**:
  + Generate download scripts
  + Create custom extraction code
  + Provide alternative approaches
* **Function**: code\_agent\_func()

**Local File Agent**

* **Role**: Content processor
* **Responsibilities**:
  + Process downloaded PDFs
  + Generate paper summaries
  + Create hypothetical summaries when PDFs unavailable
* **Function**: local\_file\_agent\_func()

**Agent Communication Hub**

**Purpose**: Central coordination and logging system

**Key Features**:

* Message logging between agents
* Result storage and aggregation
* Failed download tracking
* Performance monitoring

**External Services Integration**

**1. Mistral AI OCR Service**

**Purpose**: High-accuracy PDF text extraction

**Configuration**:

python

api\_key = "YOUR\_MISTRAL\_API\_KEY"

client = Mistral(api\_key=api\_key)

**Features**:

* Multi-page PDF processing
* Image embedding support
* Markdown output format
* High accuracy for academic papers

**2. Ollama Integration**

**Purpose**: Local LLM deployment for embeddings and generation

**Models Used**:

* nomic-embed-text:latest - Text embeddings
* gemma3:latest - Text generation
* qwen2.5vl:latest - Vision analysis

**Benefits**:

* Local deployment (privacy)
* No API costs
* Customizable models

**3. Academic Database APIs**

**arXiv API**

* **Purpose**: Download academic papers
* **Library**: arxiv Python package
* **Search**: Title, author, subject-based queries

**Semantic Scholar API**

* **Purpose**: Access open-access papers
* **Endpoint**: https://api.semanticscholar.org/graph/v1/paper/search
* **Features**: Metadata retrieval, PDF links

**DuckDuckGo Search**

* **Purpose**: General web search for papers
* **Library**: duckduckgo-search or LangChain wrapper
* **Use Case**: Fallback when academic sources fail

**User Interface**

**Streamlit Application Structure**

The application provides a comprehensive web interface with 5 main tabs:

**Tab 1: Summary & Chat**

**Features**:

* PDF summary generation
* Interactive Q&A chat
* Suggested questions
* Source citation display
* Document statistics

**Layout**: Two-column design with summary and chat sections

**Tab 2: Agent System**

**Features**:

* Agent hierarchy visualization
* Reference extraction control
* Processing status monitoring
* Results summary statistics
* CSV export functionality

**Tab 3: References**

**Features**:

* Downloaded file browser
* PDF preview functionality
* File management tools
* Reference organization

**Tab 4: Agent Logs**

**Features**:

* Real-time agent communication
* Message history display
* Error tracking
* Performance monitoring
* Debug information

**Tab 5: Report Generation**

**Features**:

* Comprehensive report creation
* Multiple export formats (Markdown, HTML)
* Status indicators
* Component validation
* Download functionality

**Sidebar Components**

* **PDF Upload Interface**
* **Processing Status**
* **PDF Preview Window**
* **Configuration Options**
* **Debug Tools**

**Installation & Setup**

**Prerequisites**

**System Requirements**:

* Python 3.8+
* 8GB RAM minimum (16GB recommended)
* 2GB free disk space
* Internet connection for API access

**Required Services**:

* Mistral AI API account
* Ollama installation (local LLM)

**Installation Steps**

1. **Clone Repository**

bash

git clone https://github.com/Lonelypheonix/IRC.git

cd IRC

1. **Install Dependencies**

python3 -m venv .venv

source .venv/bin/activate # macOS/Linux

.\.venv\Scripts\activate # Windows PowerShell

pip install --upgrade pip

pip install -r requirements.txt

1. **Setup Ollama**

bash

*# Install Ollama (platform-specific)*

*# Pull required models*

ollama pull nomic-embed-text:latest

ollama pull gemma3:latest

ollama pull qwen2.5vl:latest

1. **Configure API Keys**

python

*# Update in main.py*

api\_key = "YOUR\_MISTRAL\_API\_KEY"

1. **Run Application**

bash

streamlit run main.py

**API Documentation**

**Core Functions**

**PDF Processing**

python

def ocr\_pdf(pdf\_path: str) -> OCRResponse:

"""

Perform OCR on PDF using Mistral AI

Args:

pdf\_path: Path to PDF file

Returns:

OCRResponse: Structured OCR results

"""

**Vector Database**

python

def index\_documents(documents: List[Document]) -> int:

"""

Index documents in vector store

Args:

documents: List of LangChain documents

Returns:

int: Number of indexed chunks

"""

**RAG System**

python

def answer\_question\_with\_rag(question: str) -> Dict[str, Any]:

"""

Answer question using RAG approach

Args:

question: User question string

Returns:

Dict containing answer, sources, and context

"""

**Reference Processing**

python

def extract\_references(full\_text: str) -> List[str]:

"""

Extract reference titles from paper text

Args:

full\_text: Complete paper text

Returns:

List of reference titles

"""

**Agent System API**

**Agent Communication**

python

class AgentCommunicationHub:

def log\_message(self, sender: str, receiver: str, message: str)

def store\_result(self, paper\_title: str, summary: str, source: str)

def get\_results\_df(self) -> pd.DataFrame

**Multi-Agent Orchestration**

python

def father\_agent\_func(full\_text: str) -> Tuple[List[str], pd.DataFrame]:

"""

Orchestrate complete reference analysis

Args:

full\_text: Complete paper content

Returns:

Tuple of (references\_list, results\_dataframe)

"""

**Configuration**

**Directory Structure**

project/

├── main.py # Main application

├── pdfs/ # Uploaded PDFs

├── references/ # Downloaded references

├── output/ # Generated reports

├── agent\_logs/ # Agent communication logs

└── extracted\_images/ # PDF figures

**Configuration Variables**

python

*# Directories*

PDFS\_DIRECTORY = "pdfs/"

REFERENCES\_DIRECTORY = "references/"

SAVE\_DIRECTORY = "output/"

AGENT\_LOGS\_DIR = "agent\_logs/"

*# API Configuration*

api\_key = "YOUR\_MISTRAL\_API\_KEY"

*# Model Configuration*

embedding\_model = "nomic-embed-text:latest"

llm\_model = "gemma3:latest"

vision\_model = "qwen2.5vl:latest"

**Customization Options**

**Text Chunking**

python

*# Modify in create\_documents\_with\_metadata()*

chunk\_size = 1000

chunk\_overlap = 200

separators = ["\n\n", "\n", ".", "!", "?", " ", ""]

**Search Parameters**

python

*# Modify in retrieve\_relevant\_docs()*

similarity\_search\_k = 5 *# Number of relevant chunks*

similarity\_threshold = 0.75 *# Minimum similarity score*

**Agent Behavior**

python

*# Web Agent timeout settings*

download\_timeout = 30

max\_retries = 3

pdf\_sources = ["arxiv", "semantic\_scholar", "google\_scholar"]

**Troubleshooting**

**Common Issues**

**1. OCR Processing Failures**

**Symptoms**:

* "OCR processing failed" errors
* Empty text extraction

**Solutions**:

* Verify Mistral AI API key
* Check internet connectivity
* Ensure PDF file is not corrupted
* Try with smaller PDF files first

**2. Ollama Connection Issues**

**Symptoms**:

* "Failed to connect to Ollama" errors
* Missing embeddings

**Solutions**:

bash

*# Check Ollama status*

ollama list

*# Restart Ollama service*

ollama serve

*# Re-pull models*

ollama pull nomic-embed-text:latest

**3. Agent Download Failures**

**Symptoms**:

* All references marked as "failed"
* No PDFs downloaded

**Solutions**:

* Check internet connectivity
* Verify search API access
* Review rate limiting
* Check paper title formatting

**4. Memory Issues**

**Symptoms**:

* Application crashes during processing
* Slow performance

**Solutions**:

* Reduce chunk size
* Limit concurrent downloads
* Clear vector store periodically
* Use smaller PDF files

**Debug Tools**

**Enable Debug Mode**

python

*# In sidebar*

debug\_mode = st.checkbox("🔧 Debug Reference Extraction")

**Log Analysis**

python

*# View detailed agent logs*

logging.basicConfig(level=logging.DEBUG)

**Performance Monitoring**

python

*# Add timing decorators*

import time

start\_time = time.time()

*# ... operation ...*

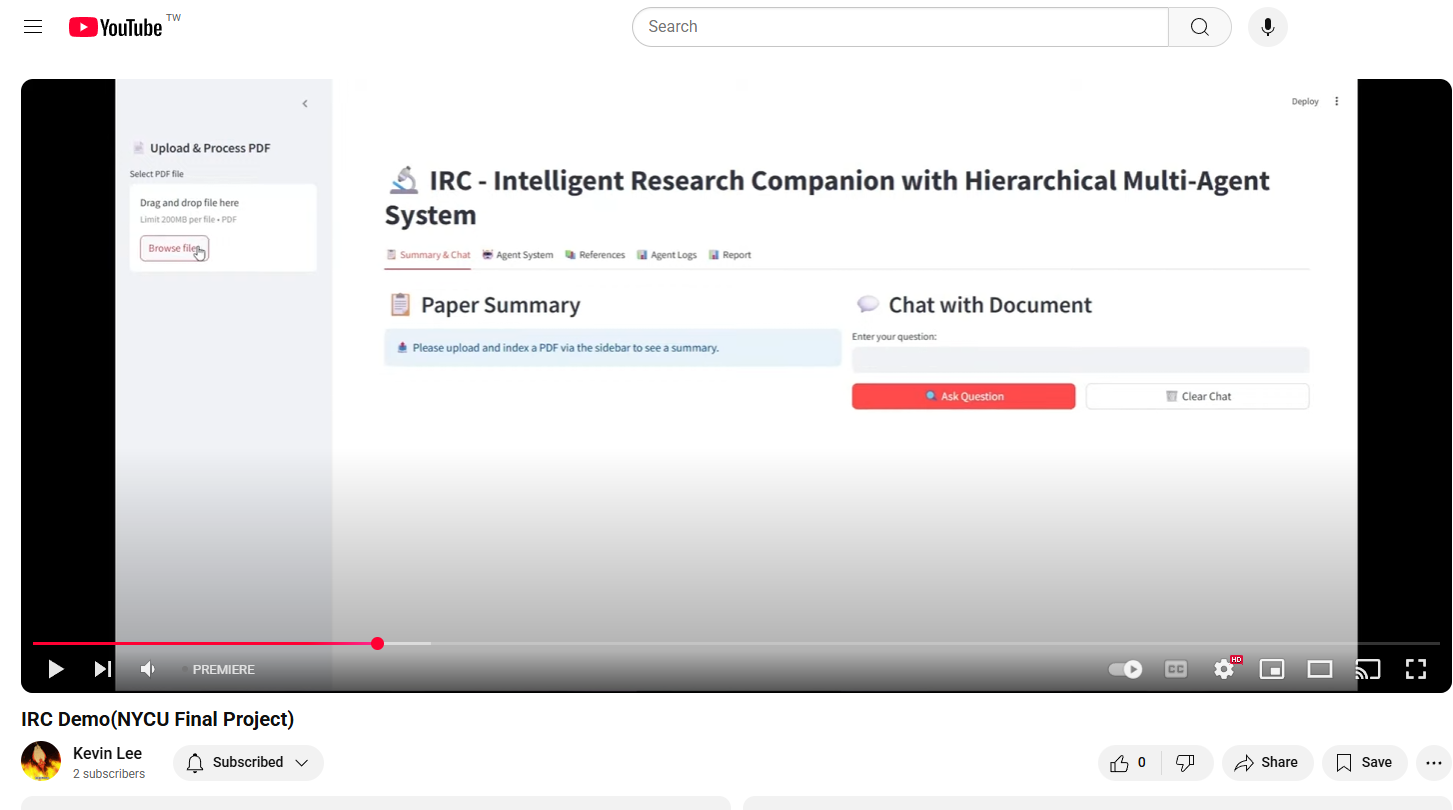
elapsed\_time = time.time() - start\_time

**Conclusion**

The IRC – Intelligent Research Companion with Hierarchical Multi-Agent System represents a comprehensive solution for academic research analysis. By combining advanced OCR, vector databases, and intelligent agents, the system automates many time-consuming aspects of literature review and research analysis.

The modular architecture ensures extensibility and maintainability, while the multi-agent approach provides robust handling of complex reference processing tasks. The system's ability to integrate multiple academic sources and generate comprehensive reports makes it a valuable tool for researchers, academics, and students.

You can check our Demo video here

[](https://www.youtube.com/watch?v=F6vvdqX_zWQ)