Agentic Research Assistant System

ABHISHEK CHANDURKAR BT22CSE104

Topic

Design and Implementation of an Agentic AI-based Research Assistant that helps researchers automate literature reviews, paper analysis, citation generation, and content drafting.

Agentic AI Framework Used

CrewAI — a lightweight Python framework to manage multiple autonomous agents collaborating toward a goal. The Coordinator Agent is implemented using CrewAI to delegate tasks to domain-specific agents.

Tools Used

- **arXiv/Semantic Scholar API** – for paper search
- **PDFLoader (LangChain)** – for extracting content from uploaded PDFs
- **Vector Store (FAISS/Chroma)** – to enable retrieval-augmented generation (RAG) for Q&A
- **Crossref/DOI Parser** – for generating citations
- **Memory Store (JSON/DB)** – for structured note-taking

CrewAI Support for Tools

CrewAI supports the integration of various tools using its `Tool` abstraction. Each tool can be a simple Python function or an advanced LangChain-based utility. The following table summarizes the tools used and their compatibility:

Tool	**Purpose**	**CrewAI Support**
arXiv / Semantic Scholar API	Search and retrieve academic papers based on user queries.	Supported via a custom Python function wrapped as a `Tool` class that makes API requests.
PDFLoader (LangChain)	Load and split research papers into manageable sections.	Directly usable by creating a LangChain-based tool and integrating it with an agent.
Vector Store (FAISS / Chroma)	Store and retrieve context for RAG (Retrieval-Augmented Generation).	Fully compatible using LangChain retriever tools wrapped for CrewAI.
Crossref / DOI Parser	Generate citations from DOIs or metadata.	Can be implemented as a custom `Tool` that fetches citation info and formats it.
Memory Store (JSON)	Save user notes, summaries, and annotations.	Easily implemented using file I/O or JSON store wrapped as a lightweight tool.

Integration Example

```
from crewai import Agent
from crewai_tools import BaseTool

class PDFTool(BaseTool):
    name = "pdf_summarizer"
    description = "Summarize␣PDF␣content␣using␣LangChain"

    def _run(self, file_path: str) -> str:
        from langchain.document_loaders import PyPDFLoader
        loader = PyPDFLoader(file_path)
        pages = loader.load_and_split()
        return pages[0].page_content

pdf_agent = Agent(
    role="PDF␣Summarization␣Agent",
    goal="Summarize␣uploaded␣research␣papers",
    backstory="An␣expert␣at␣understanding␣and␣extracting␣knowledge␣from␣PDFs.",
    tools=[PDFTool()],
    verbose=True
)
```

Why Use JSON for Notes?

- **Structured Yet Flexible:** Stores structured data like summaries, key points, annotations.
- **Easy to Read/Edit:** Useful during prototyping stages.
- **Portable:** Works well with web apps, Python scripts, and future DB upgrades.
- **Future-Ready:** Can be easily upgraded to vector stores or databases.

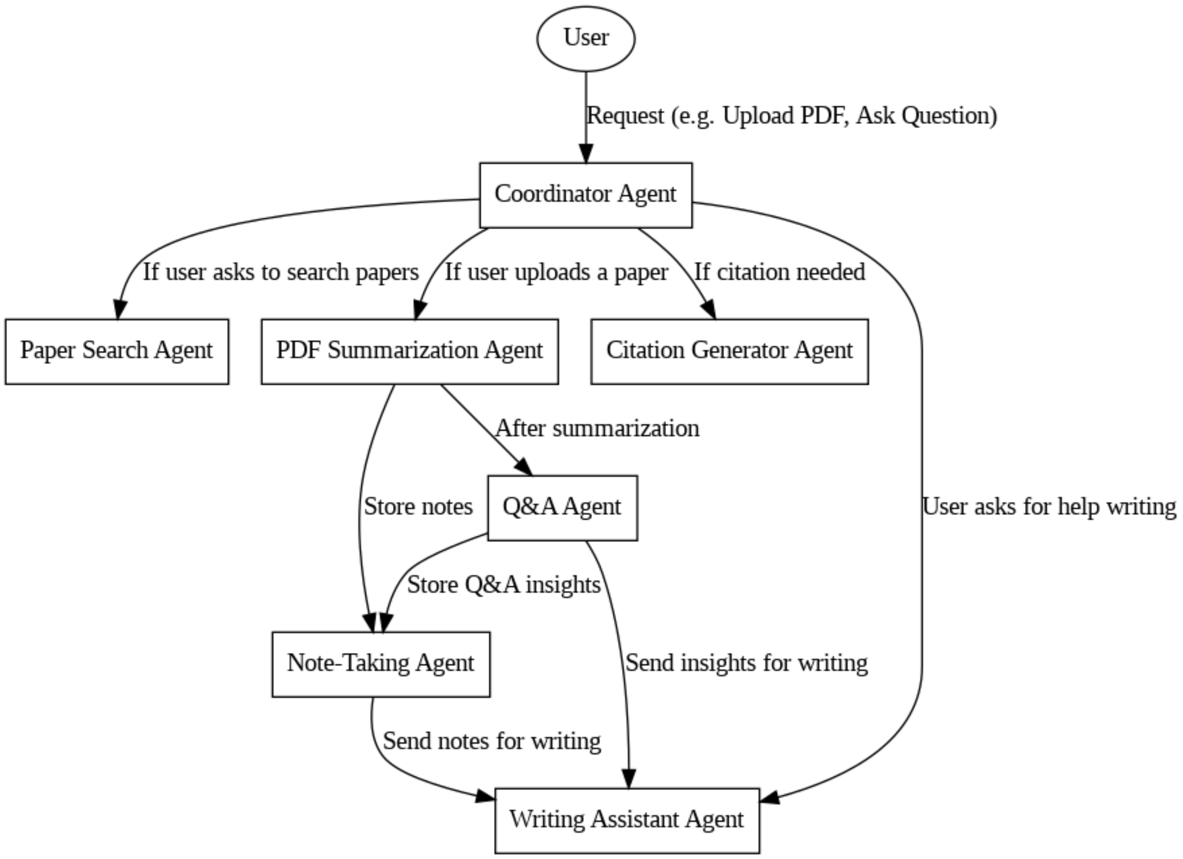
Sample JSON Schema

```
{
  "paper_title": "Agentic AI in Research",
  "summary": "This paper explores the use of autonomous agents...",
  "key_points": [
    "Agent-based orchestration",
    "Use of CrewAI framework",
    "Tool chaining via LangChain"
  ],
  "citations": [
    "Smith et al., 2023",
    "Doe and Lee, 2024"
  ],
  "user_annotations": [
    "Review methods section",
    "Highlight assumption in para 3"
  ]
}
```

LLMs Used (May change if we can find a better LLM)

- **GPT-3.5** – for summarization, Q&A, and writing assistance

System Flowchart



Description

- The **User** interacts with the system through a UI or CLI.
- The **Coordinator Agent (CrewAI)** routes tasks:
  - Sends search queries to the **Paper Search Agent**.
  - Sends PDFs to the **Summarization Agent**.
  - Delegates questions to the **Q&A Agent**, which uses RAG.
  - Stores key insights via the **Note-Taking Agent**.
  - Invokes the **Citation Generator Agent** for references.
  - Finally, routes all collected data to the **Writing Assistant Agent** to produce draft content.

Generic Agent Template vs Coordinator Agent

Generic Agent Template

```
my_agent = Agent(
    role="Role of the Agent",
    goal="What this agent is responsible for.",
    backstory="A short narrative that defines the agents personality.",
    tools=[MyTool()],
    llm=llm,
    verbose=True
)
```

Example: Coordinator Agent

```
coordinator_agent = Agent(
    role="Research Coordinator",
    goal="Coordinate the research workflow: fetch papers, assign tasks, compile
       outputs.",
    backstory="""
        You are a senior AI researcher leading a virtual research team.
        You are efficient, decisive, and experienced in orchestrating multi-agent
           tasks.
    """,
    tools=[ResearchFetcher(), PDFSummarizer(), NoteSaver()],
    llm=llm,
    verbose=True
)
```

Task Assignment in CrewAI

In CrewAI, each task is explicitly assigned to an agent using the `Task()` class. The task describes the goal, and the agent is responsible for fulfilling it.

```
from crewai import Task

# Task 1: Search for Papers
search_task = Task(
    description="Search arXiv for the latest papers on 'agentic AI frameworks'.",
    agent=fetcher_agent
)

# Task 2: Summarize PDFs
summarize_task = Task(
    description="Summarize the main contributions of each paper.",
    agent=summarizer_agent
)

# Task 3: Save the notes
save_notes_task = Task(
    description="Save summaries and metadata in structured JSON format.",
    agent=note_saver_agent
)
```

Crew Template (CrewAI)

The following is a generic template for creating a Crew in CrewAI. It defines a set of agents and their collaborative tasks.

```python
from crewai import Crew

# Define your agents above...
# from agents import my_agent_1, my_agent_2, ...

crew = Crew(
    agents=[my_agent_1, my_agent_2],
    tasks=[task1, task2],
    verbose=True
)

# Run the crew
crew.kickoff()
```

Example: Research Assistant Crew (CrewAI)

This crew consists of four agents working collaboratively to search, summarize, and save scientific papers.

```python
from crewai import Crew

# Agents (defined earlier)
from agents import coordinator_agent, fetcher_agent, summarizer_agent,
    note_saver_agent

# Tasks (defined earlier)
from tasks import search_task, summarize_task, save_notes_task

research_assistant_crew = Crew(
    agents=[
        coordinator_agent,
        fetcher_agent,
        summarizer_agent,
        note_saver_agent
    ],
    tasks=[
        search_task,
        summarize_task,
        save_notes_task
    ],
    verbose=True
)

# Start the crew
research_assistant_crew.kickoff()
```