Multi agent based Deep Researcher

Team PAS

1) Pranav Khedkar
2) Aditya Pise
3) Sudhir Pol

Project Goal

Objective:

Automatically create a research document in Arxiv format using an autonomous multi-agent system.

Key Focus:

Minimize human intervention.

Maximize document quality, structure, and relevance.

System Architecture

Hierarchical Supervisor Agent (a.k.a. the Big Boss):

- Observes the environment and decides the next action.
- Follows a **ReAct** (Reasoning + Acting) policy loop.
- Delegates work to specialized agents.

Specialized Agents:

- Each agent is a laser-focused mini-expert

Team of Specialist Agents

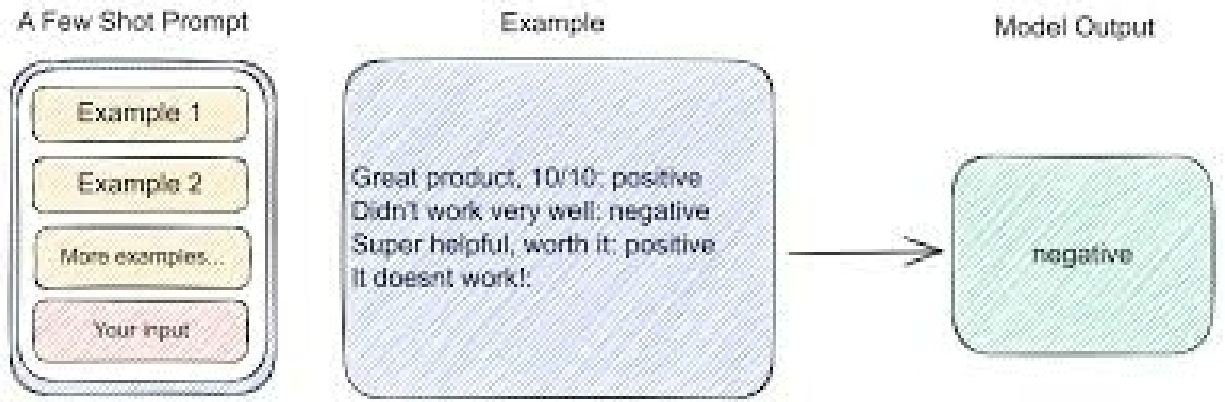
Agent	Role
Searcher	Finds relevant research articles and content.
WebScraper	Scrapes detailed information from URLs.
NoteTaker	Summarizes key points from the scraped material.
DocWriter	Formats content into a .mal file styled like Arxiv papers.
ChartGenerator	Creates graphs and visualizations with Python for embedding.

Evolution of Language Model Capabilities

In-Context Learning (ICL):

Model learns patterns from *examples provided in the prompt* — no external memory.

Limitation**:** Bound by model's internal knowledge; can't fetch new or updated information.



Source: https://images.app.goo.gl/d9eVnb7Pu5bzdZgW6

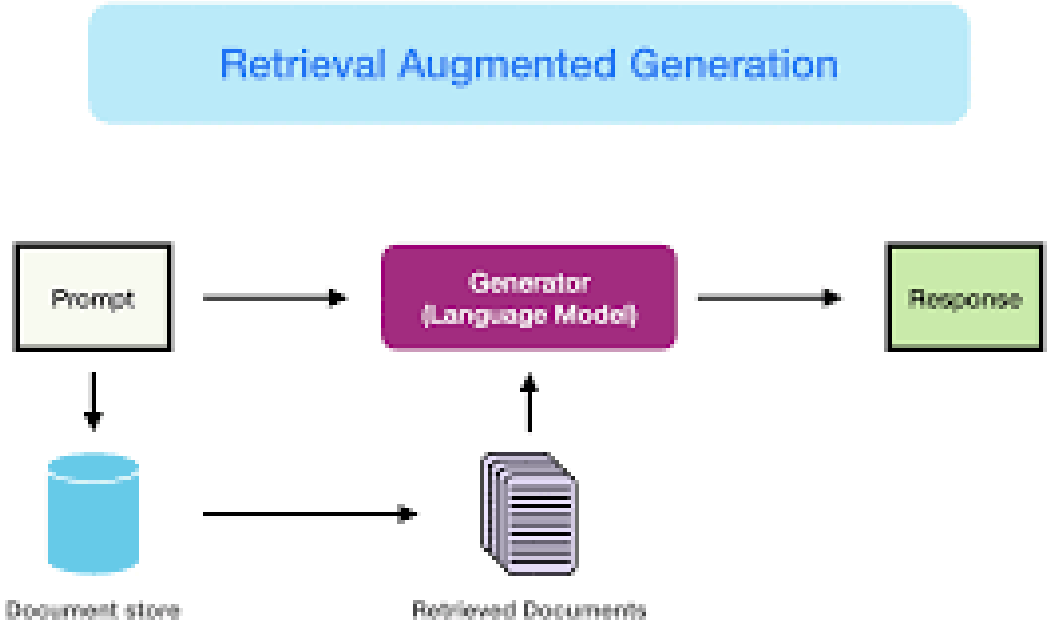
Evolution of Language Model Capabilities

Retrieval-Augmented Generation (RAG):

Extends ICL by *retrieving external documents* during response generation.

Advantage**:** Access to up-to-date information without retraining.

Limitation**:** Retrieval is passive — no dynamic interaction, no feedback or correction.



Source: https://images.app.goo.gl/FE7mE87QqzSZ1V259

RAG Drawbacks

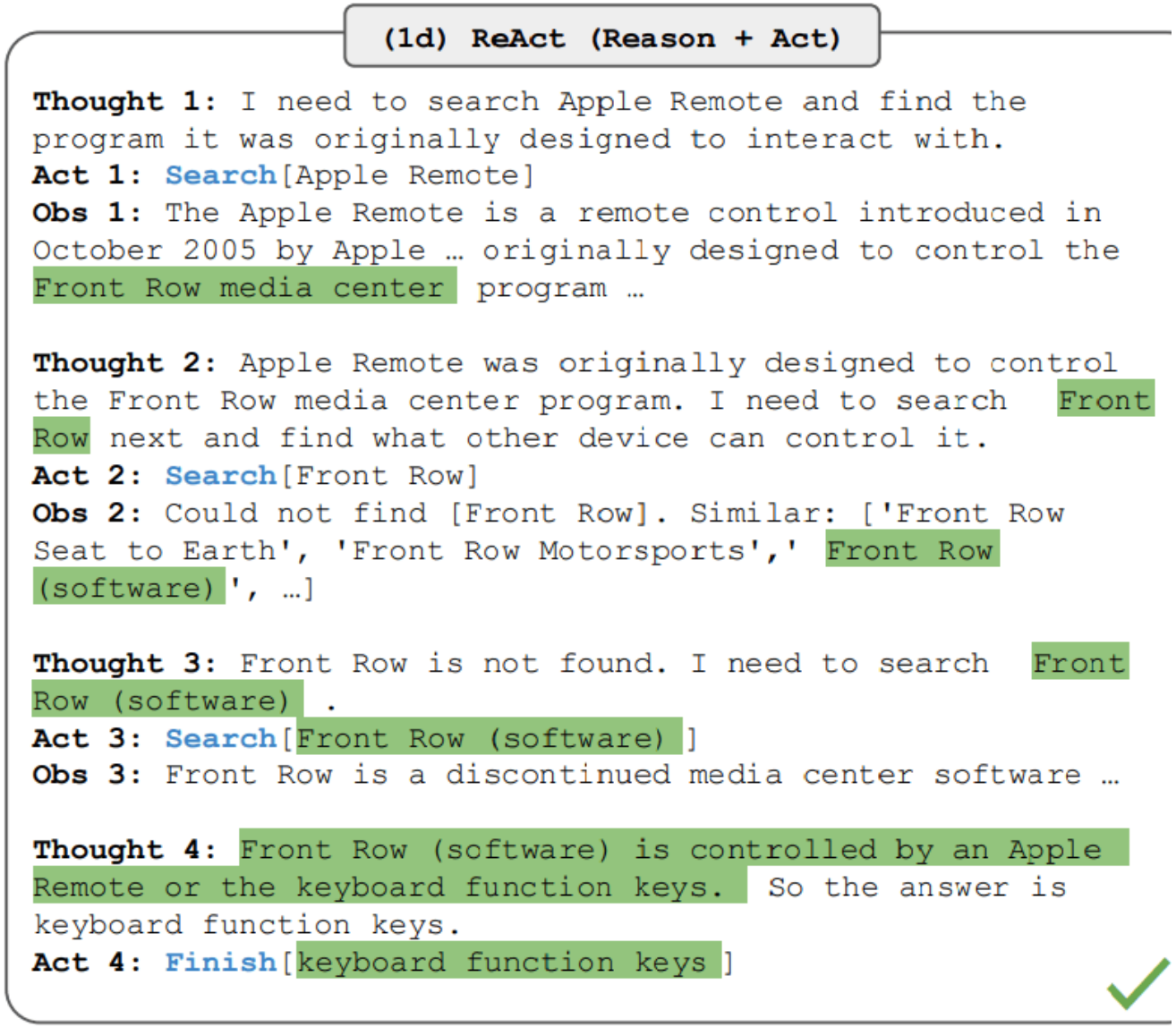
In Retrieval-Augmented Generation (RAG), tool usage must be explicitly triggered the model doesn't autonomously decide when or how to call external tools.

RAG lacks an inherent feedback loop: once information is retrieved and a response is generated, there’s no built-in mechanism for the model to evaluate the quality of its retrievals or iteratively improve its answers based on outcomes.

ReAct Framework

- ReAct is a framework where language models don't just generate answers — they think, decide, and act through an interactive reasoning process.

- Unlike standard RAG, where tool usage is manually orchestrated and there's no feedback loop, ReAct allows the model to reason about what it needs, call external tools, observe the results, and adjust its next steps based on what it learns.
- This creates a dynamic loop of thought → action → observation, enabling much more complex and intelligent behavior.



Source: Yao et al., 2022

ReAct Framework

- ReAct is a framework where language models don't just generate answers — they think, decide, and act through an interactive reasoning process.
- Unlike standard RAG, where tool usage is manually orchestrated and there's no feedback loop, ReAct allows the model to reason about what it needs, call external tools, observe the results, and adjust its next steps based on what it learns.
- This creates a dynamic loop of thought → action → observation, enabling much more complex and intelligent behavior.

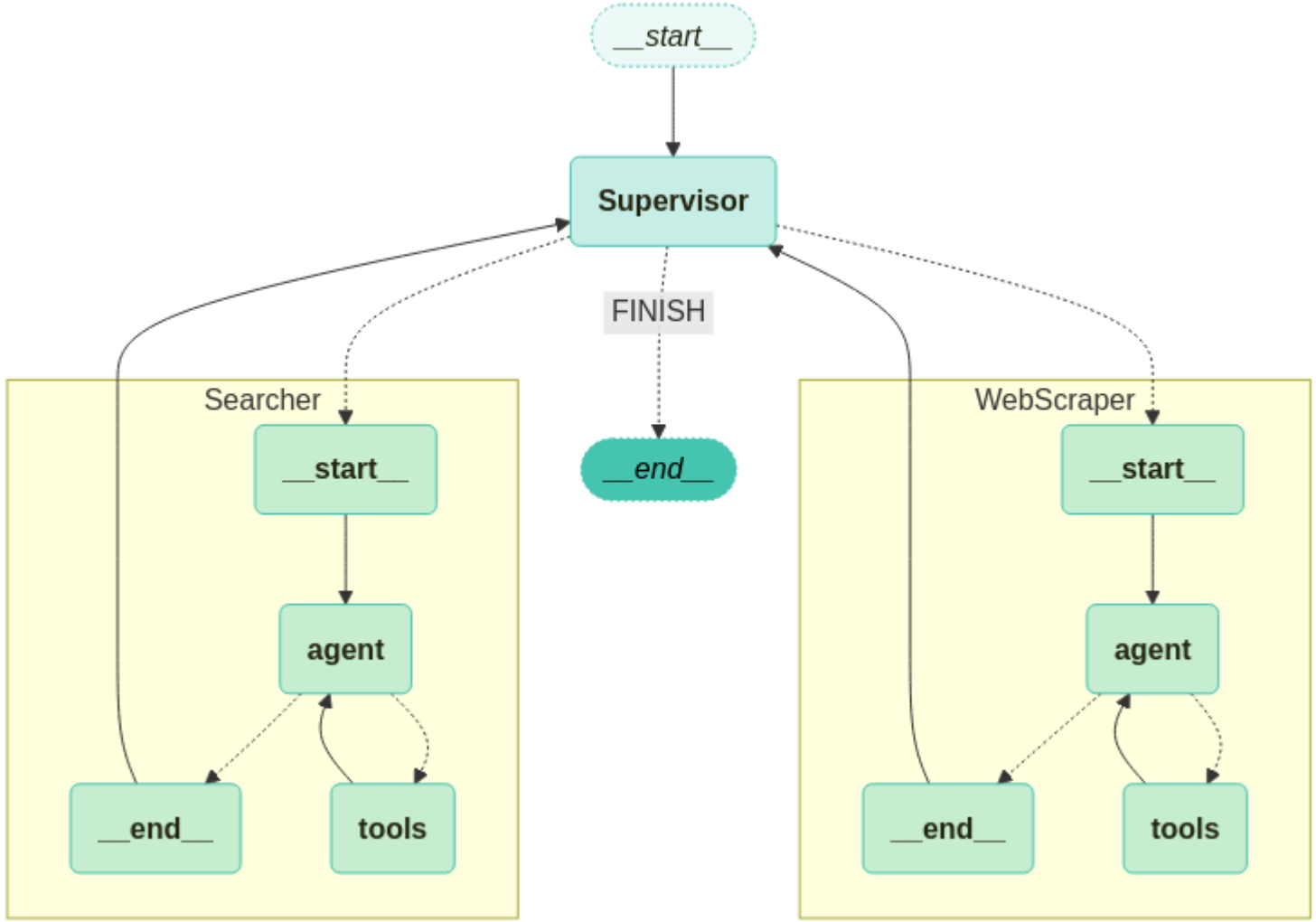
ReAct Framework

- ReAct is a framework where language models don't just generate answers — they think, decide, and act through an interactive reasoning process.

- Unlike standard RAG, where tool usage is manually orchestrated and there's no feedback loop, ReAct allows the model to reason about what it needs, call external tools, observe the results, and adjust its next steps based on what it learns.
- This creates a dynamic loop of thought → action → observation, enabling much more complex and intelligent behavior.

Key Features of ReAct:

- **Reasoning:** The model generates intermediate thoughts ("I need to look up the definition of X.") instead of jumping straight to an answer.
- **Acting:** It can autonomously choose to perform actions, like calling an API, querying a database, or using a calculator.
- **Feedback Loop:** After every action, the model observes the outcome and updates its reasoning, leading to better final answers.
- **Flexibility:** It can chain multiple tool calls and thoughts together to solve complicated, multi-step tasks.



Workflow

In this system, a **Grand Supervisor** coordinates the activities of two specialized **ReAct** agents:

- **Searcher Agent**:
  Actively queries the web to find relevant sources and links based on a user's request or a specific topic.
- **Web Scraper Agent**:
  Takes the links provided by the Searcher Agent, scrapes detailed content from those webpages, and extracts useful information.

Supervisor Control Logic

Dynamic Decision-Making:

- Checks outputs of agents after every task.
- Decides if a task needs retry, expansion, or is complete.

Adaptive Workflow:

- If content is missing ➔ sends Searcher or Scraper back into the wild.
- If enough data is collected ➔ triggers NoteTaker and DocWriter.
- Ends project only after document meets Arxiv quality standards.

Why This Design Rocks

Efficiency:

- No bloated agent memory — each one stays sharp and focused.

Scalability:

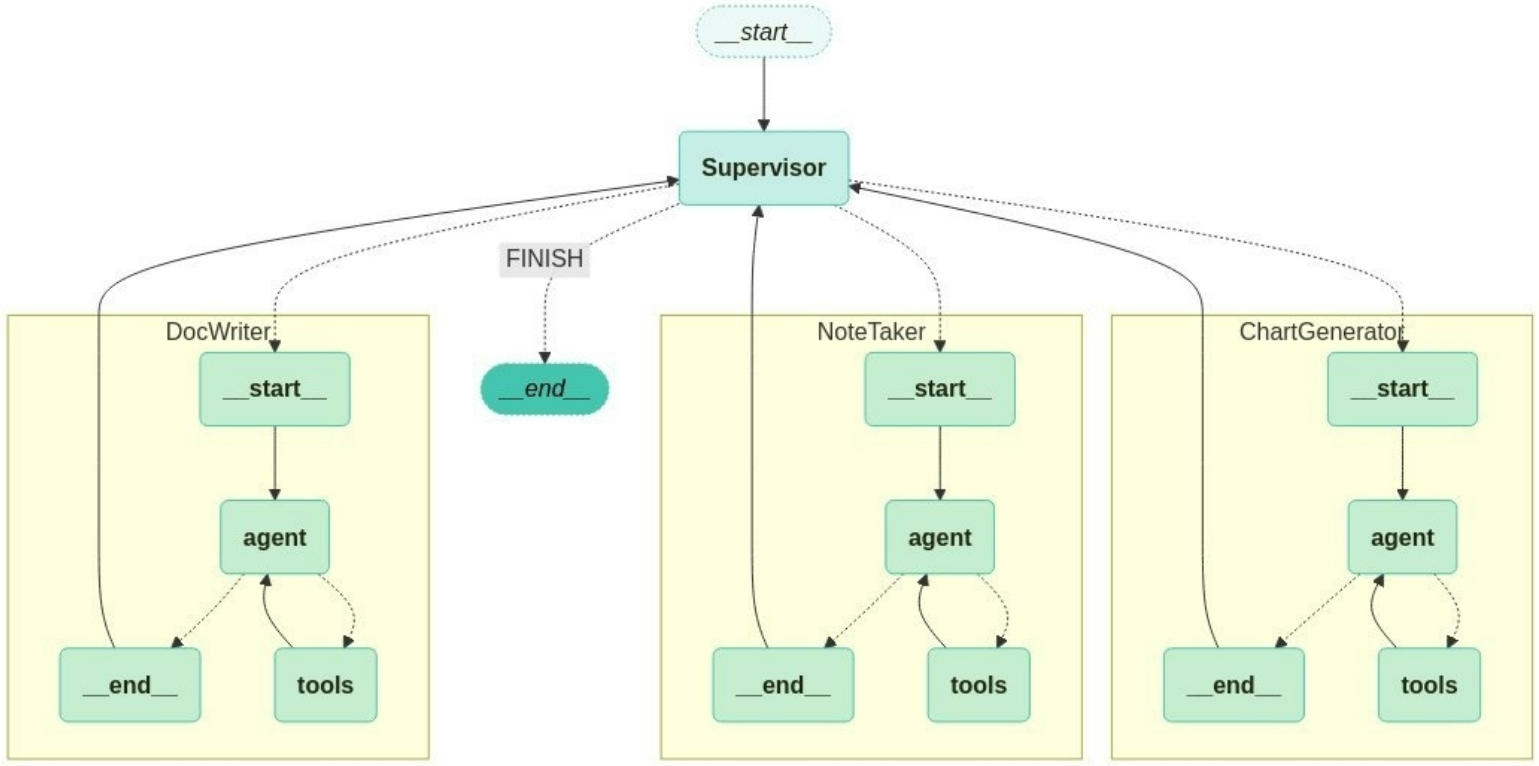
- Can easily add more agents (ex: Proofreader, Citation Fixer).

Flexibility:

- Supervisor adapts based on real-time task results.

Reliability:

- Quality control at every step, not just at the end.



Workflow

In this system, a **Grand Supervisor** orchestrates three specialized **ReAct agents**:

1. **DocWriter Agent:**

Role: Drafts structured documents (reports, summaries, specs) from raw inputs

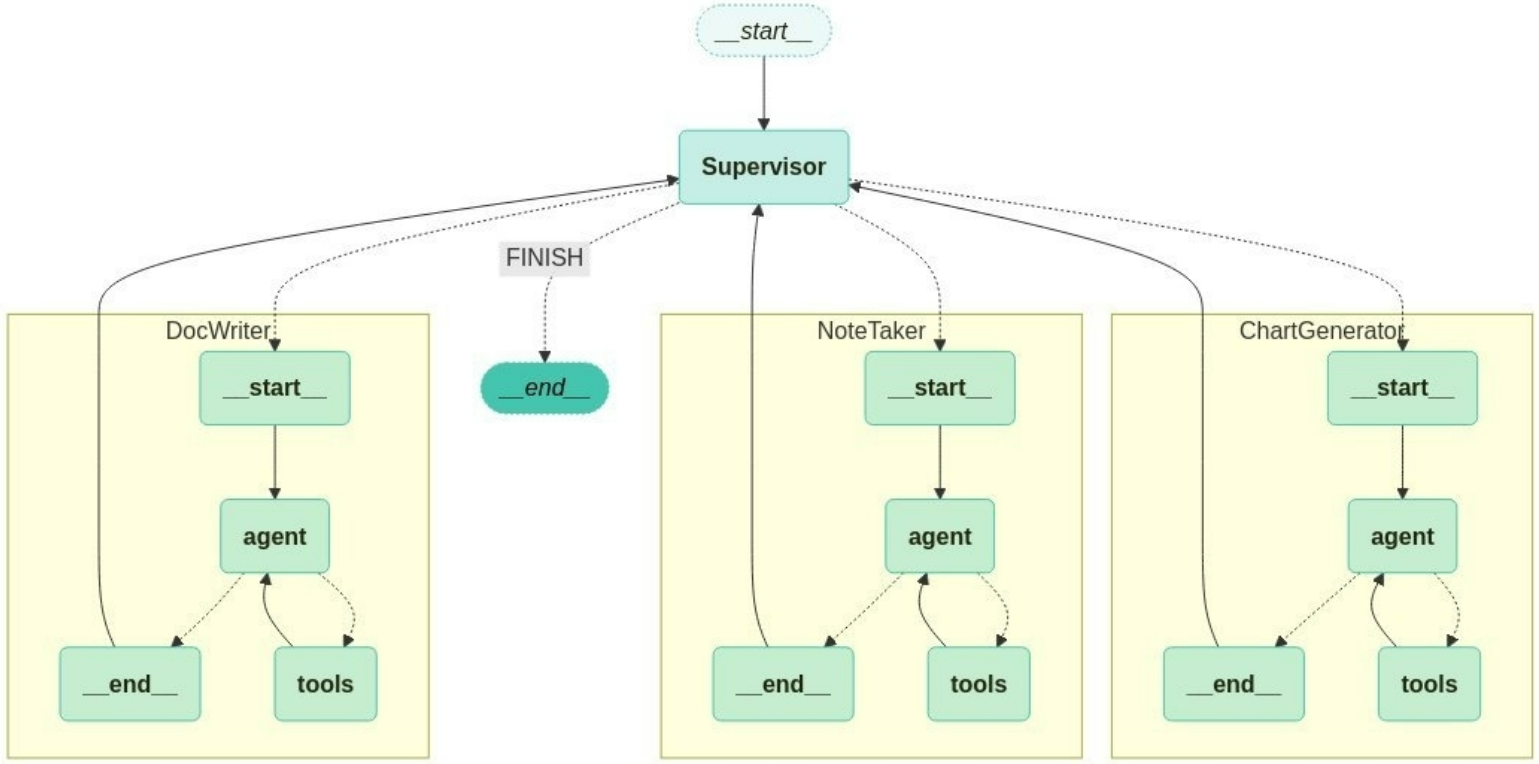
ReAct loop:

Thought: Plans document outline

Action: Calls template & content-retrieval tools

Observation: Integrates returned text

Repeat until complete



Workflow

2. NoteTaker Agent

Role: Listens to discussion transcripts or document streams, extracts key points

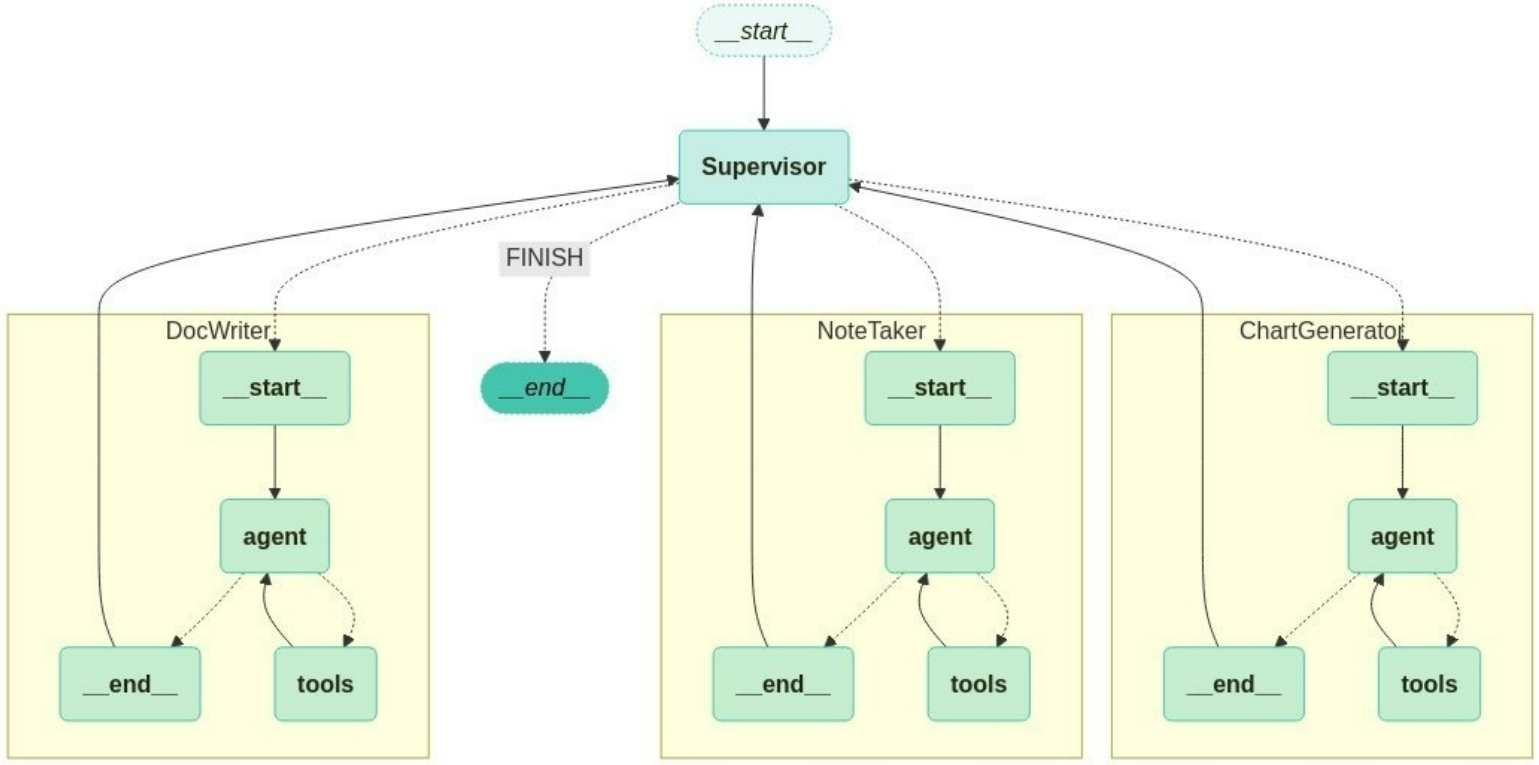
ReAct loop:

Thought: Identifies salient information

Action: Invokes summarization & keyword-extraction tools

Observation: Refines notes

Repeat until final summary



Workflow

3. ChartGenerator Agent:

- **Role**: Produces charts & visualizations on demand
- **ReAct + Python Executor**:

  ***Thought**: Chooses chart type (line, bar, scatter…)*

  Action: Generates Python plotting code

  Observation: Executes code via backend Python executor

  Repeat until final figure

Once each agent emits its __end__ signal, the Supervisor collects their outputs and signals **FINISH**.

THANK YOU!