# **LangGraph Multi-Agent System Explanation**

## **Overview**

This code implements a sophisticated multi-agent system using LangGraph that coordinates multiple AI agents to perform research and document creation tasks. The system consists of two main teams (ResearchTeam and PaperWritingTeam) supervised by a top-level supervisor.

## **Key Components**

### **1. Tools Definition**

The system provides several specialized tools:

**Web Tools:**

* scrape\_webpages(): Uses WebBaseLoader to scrape content from URLs
* TavilySearchResults(): Performs web searches using Tavily API

**Document Tools:**

* create\_outline(): Creates and saves document outlines
* read\_document(): Reads documents from the filesystem
* write\_document(): Writes content to documents
* edit\_document(): Edits documents by inserting text at specific lines

**Code Execution Tool:**

* python\_repl(): Executes Python code (with safety warnings)

### **2. Agent Teams**

#### **ResearchTeam**

* **Search Agent**: Uses Tavily search tool to find information
* **WebScraper Agent**: Scrapes web pages for detailed content
* **Supervisor**: Coordinates between search and scraping agents

#### **PaperWritingTeam**

* **DocWriter**: Handles document writing and editing
* **NoteTaker**: Creates outlines and takes notes
* **ChartGenerator**: Generates charts using Python code execution
* **Supervisor**: Coordinates the writing process

### **3. State Management**

The system uses typed state dictionaries to manage:

* Message history between agents
* Team member information
* Next action routing
* File system state

### **4. Graph Architecture**

**Research Graph:**

text

START → Supervisor → [Search ↔ WebScraper] → END

**Authoring Graph:**

text

START → Supervisor → [DocWriter ↔ NoteTaker ↔ ChartGenerator] → END

**Super Graph:**

text

START → Supervisor → [ResearchTeam ↔ PaperWritingTeam] → END

## **Teaching Points**

### **1. Multi-Agent Coordination**

The system demonstrates how to:

* Create specialized agents with specific tools
* Use supervisors to route tasks between agents
* Maintain conversation history across multiple agents
* Handle conditional branching based on agent outputs

### **2. State Management**

* Using TypedDict for type-safe state management
* Annotated lists for message accumulation with operator.add
* Maintaining file system context across agents

### **3. Tool Design**

* Creating reusable, specialized tools
* Handling file I/O operations safely
* Executing code with proper error handling
* Web scraping and search integration

### **4. Graph Construction**

* Adding nodes and edges to state graphs
* Creating conditional edges for dynamic routing
* Compiling graphs into executable chains
* Handling recursion limits for complex workflows

## **Key Code Patterns to Highlight**

### **1. Agent Node Pattern**

python

def agent\_node(state, agent, name):  
 result = agent.invoke(state)  
 return {"messages": [HumanMessage(content=result["messages"][-1].content, name=name)]}

### **2. Supervisor Creation**

python

def create\_team\_supervisor(llm, system\_prompt, members):  
 *# Creates routing function with JSON output parsing*

### **3. Graph Construction**

python

graph = StateGraph(State)  
graph.add\_node("agent\_name", agent\_function)  
graph.add\_edge("node1", "node2")  
graph.add\_conditional\_edges("supervisor", routing\_function)

### **4. Context Awareness**

python

context\_aware\_agent = prelude | agent *# Pipe operator for composition*

## **Teaching Approach**

### **Step 1: Basic Concepts**

1. Explain what LangGraph is and why it's useful for multi-agent systems
2. Demonstrate simple agent creation with tools
3. Show basic graph construction

### **Step 2: Team Coordination**

1. Explain supervisor pattern and routing
2. Show how agents communicate through shared state
3. Demonstrate conditional workflow routing

### **Step 3: Advanced Patterns**

1. File system integration and context management
2. Code execution safety considerations
3. Recursion limits and error handling

### **Step 4: Real-world Application**

1. Research and report generation workflow
2. Team specialization and tool allocation
3. End-to-end system demonstration

## **Safety Considerations**

* The python\_repl tool executes code locally (highlight the risks)
* Web scraping should respect robots.txt and terms of service
* File system operations need proper sandboxing in production
* Recursion limits prevent infinite loops

## **Extension Ideas**

* Add more specialized agents (data analysis, image generation)
* Implement persistence for long-running tasks
* Add user approval steps for critical operations
* Implement rate limiting and usage tracking

This system provides an excellent foundation for teaching advanced multi-agent concepts and could be extended for various educational and practical applications