Multi agent based Deep Researcher

Aditya Pise
Indiana University Bloomington
anpise@iu.edu

Pranav Khedkar
Indiana University Bloomington
pkhedkar@iu.edu

Sudhir Pol
Indiana University Bloomington
sypol@iu.edu

Abstract

We propose an autonomous, hierarchical multi-agent framework for end-to-end generation of research manuscripts in ArXiv format. Central to our design is a Supervisor agent that orchestrates specialized ReAct agents—Searcher, WebScraper, NoteTaker, DocWriter, and ChartGenerator—to iteratively locate scholarly articles, extract and summarize key content, and assemble structured documents. By integrating dynamic reasoning with targeted tool calls, our approach minimizes human intervention while ensuring coherent document structure and relevance. This work lays the foundation for automated academic writing pipelines capable of self-directed research and composition.

1 Introduction

Large language models (LLMs) have demonstrated impressive capabilities in natural language generation, yet they often struggle with fully autonomous, multi-step tasks such as composing complete research papers. In-Context Learning (ICL) allows LLMs to perform new tasks from a few examples, but remains constrained by the static knowledge encoded in model parameters (1). Retrieval-Augmented Generation (RAG) improves factual accuracy by retrieving and integrating external documents at inference time, yet it generally operates in a single pass without built-in support for iterative planning or decision making (2).

To address these limitations, we present a hierarchical multi-agent architecture based on the ReAct (Reasoning+Acting) paradigm (3). A central Supervisor agent orchestrates specialized sub-agents that handle literature retrieval, content summarization, LaTeX assembly in ArXiv format, and figure generation. By alternating between reasoning and tool actions, each sub-agent can adapt dynamically, recover from errors, and refine its output over multiple iterations.

Our system automates the most labor-intensive stages of scholarly writing—source gathering, knowledge synthesis, and visualization—while ensuring documents adhere to the structure, coherence, and factual rigor expected in academic publications. This approach lowers the barrier to high-quality research writing, empowering students, early-career researchers, and domain experts to generate structured manuscripts rapidly and with minimal manual effort, thereby accelerating innovation and broadening participation in academic discourse.

2 Related Work

Automated document generation with large language models has been an active area of research in recent years. Early work on in-context learning showed that models like GPT-3 can perform few-shot generation of coherent text, but remain limited by their static context windows and lack of external knowledge retrieval (1). Retrieval-Augmented Generation (RAG) methods augment pre-trained models with retrieved documents at inference time, improving factuality and relevance in knowledge-intensive tasks (2). However, RAG systems typically operate in a single-pass fashion and do not inherently support complex multi-step workflows or iterative refinement.

Autonomous Multi-Agent Reasoning To address the need for dynamic decision-making and tool use, recent work has investigated agentic architectures that interleave reasoning and acting. The ReAct framework introduces a paradigm in which language models alternate between generating natural-language reasoning traces and performing actions such as API calls, enabling more structured and controllable workflows (3). WebGPT incorporates a browser-based search loop, allowing the agent to navigate web pages, extract content, and cite sources to answer queries with

higher accuracy (4). These approaches demonstrate the potential of model-based agents but are largely flat—relying on a single model to both plan and execute.

Hierarchical and Modular Agent Frameworks Building on these foundations, hierarchical multi-agent systems introduce separate roles for planning, information gathering, summarization, and document assembly. LangChain provides a modular framework for chaining LLM calls with custom tools, facilitating the construction of complex pipelines (5). Similarly, LlamaIndex (formerly GPT Index) offers data-centric abstractions to manage document retrieval and indexing workflows for language models (6). These toolkits enable developers to orchestrate multiple specialized components but leave it to the user to manually wire together supervisors and sub-agents.

Automated Research Paper Generation There have been several domain-specific efforts aimed at automating scholarly writing. SciTLDR formulates a supervised approach to generate concise paper summaries but relies on existing datasets of abstracts and papers (7). Automated pipelines like Scholarcy (8) and SciBot (9) perform document parsing, summarization, and reference extraction, yet lack a unified supervisory agent to oversee end-to-end composition at the manuscript level. Our work distinguishes itself by combining a hierarchical supervisor agent with specialized ReAct sub-agents, enabling fully autonomous research paper generation with minimal human oversight.

3 Data

We begin by generating thematic search queries tailored to the target topic, which are submitted to the `TavilySearchResults` tool (configured to return up to six top URLs per query). The returned links are fetched in bulk using `WebBaseLoader` with a realistic browser user agent header. Each page's raw HTML is parsed by `BeautifulSoup` to strip tags, collapse whitespace, and extract clean, plain-text content. We also capture metadata—titles, URLs, and fetch timestamps—for each document. In total, our working corpus comprises up to six documents per query, drawn from a diverse set of sources including news outlets, technical blogs, and public documentation.

4 Methods

4.1 Agent Architectures

Multi-agent systems can be organized in several architectural paradigms:

- **Pipeline (Linear) Architecture**: Agents are arranged in a fixed sequence, each performing a specific transformation on the data before passing it to the next. While simple, this design lacks flexibility for dynamic task allocation.
- **Blackboard Architecture**: Agents post partial solutions to a shared "blackboard" and others contribute opportunistically. This supports loose coupling but can suffer from contention and complexity in coordinating read/write access.
- **Ensemble (Parallel) Architecture**: Multiple agents work independently on the same problem, and a fusion mechanism aggregates their outputs. This can improve robustness but requires a sophisticated voting or consensus strategy.
- **Hierarchical Architecture**: Agents are organized into nested levels, where higher–level controllers (supervisors) delegate tasks to specialized lower–level agents. This promotes modularity, clear separation of concerns, and dynamic routing based on context.

Our Choice of Hierarchical Design We adopt a hierarchical architecture to leverage its key advantages for complex, multi-stage workflows. By introducing a Supervisor agent at each level, we enable:

- *Dynamic Task Allocation*: The Supervisor inspects real-time state and chooses the most appropriate worker (e.g., Searcher vs. WebScraper, NoteTaker vs. DocWriter) rather than following a rigid sequence.
- *Modularity and Extensibility*: New agents (e.g., a "ReviewTeam" or "CitationGenerator") can be added under a Supervisor without reconfiguring the entire pipeline.
- *Stateful Coordination*: The hierarchical design, combined with persistent state via `MemorySaver`, ensures that each decision point has full visibility into past interactions and generated artifacts.

- *Fault Isolation*: Errors or exceptions in one sub-loop (e.g., web scraping) do not derail the entire system; the Supervisor can reassign or retry as needed.

This hierarchical pattern underpins both our research and authoring loops, yielding a flexible, maintainable framework suited to end-to-end automated paper preparation.

4.2 Pipeline Overview

We design a modular, state-driven agent pipeline using LangGraph and LangChain (see Figure 3). This pipeline is composed of three nested loops—web research, document authoring, and a top-level supervisor—that together automate query generation, content scraping, outline creation, chart generation, and manuscript drafting.

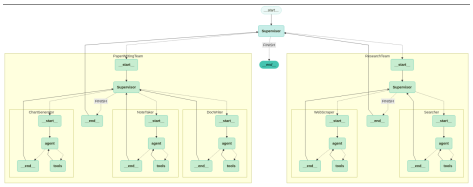

Figure 1: Overall LangGraph–LangChain agent pipeline, showing ResearchTeam and PaperWritingTeam loops managed by the Supervisor.

4.3 Web Research Workflow

The *ResearchTeam* loop comprises two ReAct agents:

- **Searcher**: issues thematic queries to `TavilySearchResults` (up to six URLs/query).
- **WebScraper**: fetches each URL via `WebBaseLoader` and cleans the HTML with BeautifulSoup.

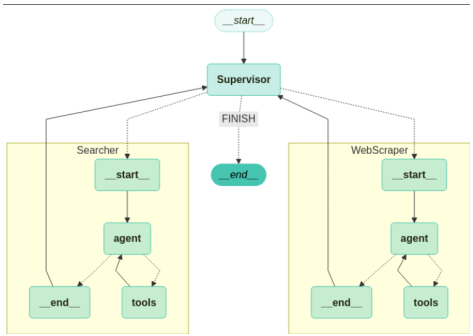

Figure 2: Web Research Workflow

This produces a cleaned corpus of up to 6 documents per query.

4.4 Document Authoring Workflow

The *PaperWritingTeam* loop has three agents:

- **NoteTaker**: generates a section-by-section outline using `create_outline` and `read_document`.
- **DocWriter**: drafts arXiv-style text, writing and editing files with `write_document`, `edit_document`, and `read_document`.
- **ChartGenerator**: runs user-provided Python snippets (`python_repl_tool`) to produce figures.

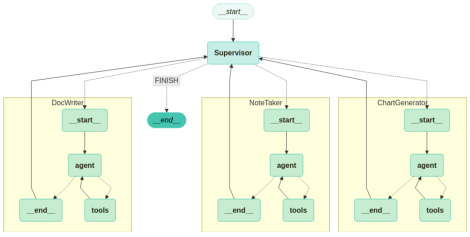

Figure 3: Document Authoring Workflow

4.5 Sample Agent Outputs

Outline Generation Figure 4 shows an example outline produced by the NoteTaker agent.

Figure 4: Example outline generated by the NoteTaker agent.

Chart Generation Figure 5 displays a sample chart produced by the ChartGenerator.

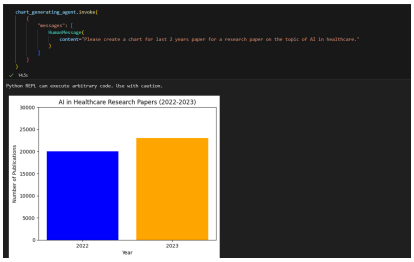

Figure 5: Sample figure generated by the ChartGenerator agent.

4.6 State Management & Execution

All three graphs (web research, authoring, and the top-level supervisor) are compiled with a `MemorySaver` checkpointer to persist state across runs. We invoke the entire pipeline via:

`run_graph(<compiled_app>, "<user prompt>")`

which wraps the prompt in a `HumanMessage`, streams execution, and returns the final state values.

5 Implementation

We began by experimenting in a Jupyter Notebook, first with a single-agent prototype and then with flat multi-agent configurations. During this phase we evaluated agent interactions and coordination strategies using the LangGraph framework (10). These experiments revealed trade-offs between simplicity and scalability, leading us to adopt a hierarchical design.

In the final implementation a top-level Supervisor agent oversees two distinct teams of specialist agents. The *Retrieval Team* comprises the Searcher and WebScraper agents, responsible for finding relevant literature and extracting detailed content. The *Composition Team* includes the NoteTaker, DocWriter, and ChartGenerator agents, which summarize findings, assemble Markdown outputs, and generate figures. This separation of concerns streamlines error handling and allows each team to focus on its core function.

To make the pipeline user-friendly, we built a minimal Streamlit interface that exposes the components described below. The interface saves the generated Markdown document to a designated output folder and provides a live preview.

1. **Mode Selection:**
   - "Simple Query" for a quick lookup or to generate a blog/report in any user-specified format.
   - "Research Query" for end-to-end arXiv-style research paper generation.
2. **Topic Input:**
   - A free-form text box where the user enters the research question or topic.
   - Placeholder text guides formatting (e.g., "What are the latest developments in quantum computing?").
3. **Agent Overview Panel:**
   - Shows the team of agents—Searcher, WebScraper, NoteTaker, DocWriter, ChartGenerator—and the Supervisor.
   - Provides brief descriptions of each agent's role in the pipeline.
4. **Processing Depth Slider:**
   - Limits the number of nested retrieval and summarization loops and corresponds to LangGraph's `recursive_limit` parameter that caps nested ReAct cycles.
   - Balances thoroughness against execution time by preventing unbounded iteration.
5. **Progress & Output Display:**
   - Real-time status logs for each agent, e.g. *"Research Team is working…"* followed by a green checkmark and *"Research Team completed."*
   - When all agents finish, the generated Markdown document is rendered inline.

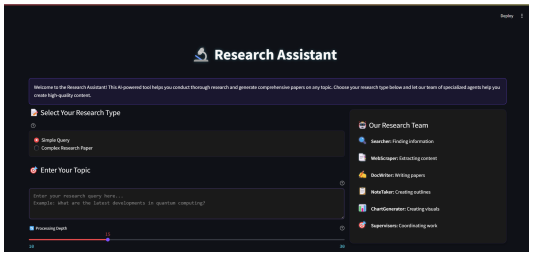

Figure 6: Streamlit UI showing research mode selection, topic input, agent roles overview, and processing depth control.

Execution Flow

1. **Mode Selection:**
   - *Simple Query:* sends the user's text directly as a question to the agent pipeline and uses a lower depth range (10–30) for fast, high-level summaries.
   - *Complex Research Paper:* automatically prefixes the input with detailed prompt for the full multi-agent workflow and uses a higher depth range (50–200) for detailed, multi-step document generation.
2. **Topic Entry:** Enter the research topic into the text input field.
3. **Processing Depth:** Adjust the slider to set the number of retrieval and summarization iterations.
4. **Agent Overview:** Review the displayed Specialist Agents (Searcher, WebScraper, NoteTaker, DocWriter, ChartGenerator) and the Supervisor.

5. **Submit:** Click the "Submit" button to launch the pipeline.

6 Results

6.1 Evaluation of Reports

1. **Comprehensive Overview of Agentic Frameworks in AI Development**
   (Mode - Simple Query) Coherence, coverage, and factual accuracy were marked down due to occasional conceptual gaps and sparse elaboration.

2. **Artificial Intelligence in Healthcare**
   (Mode - Simple Query) Minor factual nuances and uneven structure led to moderate scores in coherence and coverage.

3. **Developments in Quantum Computing**
   (Mode - Simple Query) Coverage was slightly reduced by a narrower selection of examples, despite excellent precision and flow.

4. **Overview of LangChain and its Examples**
   (Mode - Simple Query) Depth of examples limited coverage, though overall clarity and factuality remained strong.

5. **Transformer Architecture**
   (Mode - Research Paper) While highly polished, the report could benefit from more illustrative case studies, affecting coverage.

6. **Streaming Algorithms**
   (Mode -Research Paper) A minor inaccuracy in one algorithm definition resulted in a slightly lower factual accuracy score.

Each report was assessed by an LLM-based judge on five criteria—Relevance, Coherence, Coverage, Factual Accuracy, and Presentation—using a 1–5 scale (1=poor, 5=excellent). A slightly stricter rubric was applied, and a combined average score was computed for each report.

Criterion	Q1	Q2	Q3	Q4	Q5	Q6
Relevance	5	5	5	5	5	5
Coherence	3	4	5	5	5	5
Coverage	3	4	4	4	4	5
Accuracy	3	3	5	5	5	4
Presentation	5	5	5	5	5	5
Average	3.8	4.2	4.8	4.8	4.8	4.8

6.2 Human Observations

Informal user testing revealed both strengths and weaknesses of the Research Assistant:

Strengths:

- **Effortless Exploration:** The interface offers the easiest way to gather information on any topic, removing the need to manually sift through multiple sources.
- **Depth on Demand:** Users can request deeper dives iteratively, refining and expanding the output with additional agent cycles.
- **Versatile Content Generation:** The system produces well-structured blogs, reports, and academic-style papers, making it valuable for a wide range of writing tasks.
- **Consistent Quality:** Outputs maintain a professional tone and clear organization, reducing editing overhead for end users.

Weaknesses:

- **Incomplete Citations:** Reference lists or in-text citations are sometimes missing, which can hinder verification of source material.
- **Lack of Illustrative Examples:** Some reports omit concrete examples or case studies, limiting practical understanding of concepts.

6.3 Discussion and Limitations

- **Hallucinations:** Large language models can generate plausible but incorrect statements—so-called hallucinations—which pose a fundamental risk in our multi-agent pipeline. If one agent produces a hallucinated fact (for example, misquoting a paper title or inventing a statistic), downstream agents will propagate the error. Robust validation checks or a dedicated "fact-checker" agent are needed to mitigate this failure mode.
- **Need for Processing Depth Slider:** Our recursive agent loops leverage LangGraph's `recursive_limit` to prevent unbounded ReAct cycles. Exposing this as a "Processing Depth" slider allows users to balance thoroughness against runtime and cost: higher depth improves coverage at the expense of latency, while lower depth yields faster results but may miss deeper insights. This control is essential to tailor the system to diverse research needs and computational budgets.

- **Handling Long Web Pages:** Many web pages exceed the LLM's token limit. To process them, we split each page into overlapping segments that fit within the model's budget, summarize each segment independently, and merge the summaries into a coherent whole. This chunking strategy preserves critical information while respecting model constraints.
- **Integration of MCP Server:** To support newer agentic protocols and enable real-time inter-agent communication, we can integrate a Multi-Channel Protocol (MCP) server. This server would manage message routing, synchronization, and state sharing among agents, improving scalability, fault tolerance, and extensibility of the multi-agent system.
- **Model Selection:** The system uses GPT-4o-mini for its low cost and ability to handle multiple calls efficiently without significantly increasing expense. Exploring larger models may yield further improvements in output quality and depth.
- **Human Feedback Loop:** Incorporating a human-in-the-loop feedback mechanism can provide users with greater control over the output and enable personalized adjustments, improving relevance and satisfaction.

7 Conclusion

We have developed an autonomous hierarchical multi-agent framework—accessible via a simple Streamlit UI—that generates complete research manuscripts in arXiv format with minimal human input. LLM-based evaluations and user feedback confirm strong performance in relevance, coherence, coverage, factual accuracy, and presentation. Future work will focus on integrating a fact-checker agent, adaptive depth control, document chunking for long sources, and real-time agent coordination via an MCP server, as well as exploring larger LLMs to further enhance output quality.

8 Ethical Considerations

8.1 Data Collection and Verification

The system uses external sources to collect information and summarize it in arXiv format. While most content is gathered from public domains, thorough verification is essential. All data must be scanned to remove personally identifiable information, and procedures must prevent collection of gated content or violation of website terms of service. Future iterations should include domain whitelisting and automated safeguards to ensure responsible and lawful data access.

8.2 Misinformation and Trustworthiness

Scraped and generated content may include low-quality or incorrect data if sources are not properly vetted. A robust filtering mechanism is needed to identify and prioritize credible information. It is ethically imperative to disclose that the final document is machine-generated and may require human review, especially in scientific or academic contexts.

8.3 Authorship and Academic Integrity

Users must be transparent about their use of this tool. The system is designed to assist in research and writing, not to produce entire papers without human contribution. Relying solely on generated content for publication would constitute academic dishonesty; instead, the tool should augment, not replace, the author's original work.

9 AI Use Declaration

All core reasoning, retrieval, summarization, and document-generation steps in our hierarchical multi-agent pipeline were performed using the GPT-4o-mini model via OpenAI's API. No external embedding models were used; all interactions relied on prompt-based LLM calls. The evaluation dataset—comprising generated prompts, outlines, and full reports—was created programmatically by our system using GPT-4o-mini. The Streamlit interface was implemented without any generative extensions. ChatGPT was consulted only for grammar and style editing during report composition.

10 Author Contributions

Aditya Pise: Research on agentic systems; Research Assistant UI application; Demo report writing.
Pranav Khedkar: Research iterations of agentic system; Presentation preparation; Report writing.
Sudhir Pol: Research initial prototype for agentic system; presentation preparation; Report writing.
All authors: Designed the experiments, interpreted the results, and approved the final manuscript.

Appendix

10.1 Submission Folder Contents

The ‘submission‘ folder includes:

- A Jupyter Notebook containing the full agent system implementation.
- The Streamlit application source code that provides the user-friendly UI.
- All resulting Markdown (‘.md‘) files generated by the pipeline that were used for evaluation.

10.2 Tavily Search Integration

The Research Assistant pipeline leverages the Tavily Search API to retrieve high-quality source documents. Key details of this integration are:

- **Service Overview:** Tavily Search provides a programmable interface for querying scholarly and web content, returning ranked results with metadata (title, URL, snippet, publication date).
- **Authentication:** An API key is stored securely in environment variables and passed via an HTTP header.
- **Query Parameters:**
  - `q`: the user's research topic or sub-query.
  - `limit`: number of results per call (default10).
  - `filter`: optional domain or date filters to constrain results.
- **Response Handling:** The Searcher agent parses the JSON response, extracts URLs and snippets, and passes each URL to the Web-Scraper agent for detailed content extraction.
- **Error Handling and Retries:** HTTP errors (e.g., 429, 500) trigger exponential backoff with up to three retry attempts before logging a failure.
- **Configuration:** Endpoint and default parameters are defined in ‘config.yaml‘, allowing users to customize result limits and filters without code changes.

10.3 GPT-4o-mini Integration

- **Model:** GPT-4o-mini for low-cost, concurrent API calls.
- **Authentication:** API key in `OPENAI_API_KEY` environment variable.
- **Prompts:** Simple single-agent templates for quick queries; structured multi-agent sequences for full papers.
- **Parameters:** Tuning of `temperature`, `max_tokens`, and `top_p`; model set to `"gpt-4o-mini"`.
- **Call Management:** Async execution with rate limiting and LangGraph's `recursive_limit` to cap total calls and control cost.
- **Error Handling:** Up to three retries with exponential backoff; persistent failures logged for Supervisor intervention.
- **Configuration:** All settings (model, timeouts, retries) configurable via `config.yaml`.

10.4 Chart Generator Python REPL

- **Environment:** A sandboxed Python REPL running with key libraries (e.g., `matplotlib`, `pandas`) installed.
- **Invocation:** The ChartGenerator agent sends generated code snippets to the REPL for execution.
- **Dependencies:** Core plotting and data-handling packages are preloaded; additional libraries can be enabled via `requirements.txt`.
- **Output Handling:** Generated figures are saved as image files and passed back to the Supervisor for inclusion in the final document.
- **Error Management:** Runtime errors in the REPL trigger concise tracebacks; the Supervisor can retry or skip problematic snippets.

References

[1] T. Brown *et al.*, "Language models are few-shot learners," *arXiv preprint arXiv:2005.14165*, 2020.

[2] P. Lewis, E. Perez, A. Pugachev, D. Karpukhin, N. Goyal, M. Lewis, W. Yin, L. Gao, and V. Stoyanov, "Retrieval-augmented generation for knowledge-intensive NLP tasks," in *NeurIPS*, 2020.

[3] S. Yao, C. Zhou, P. Sperber, J. Meng, D. Zhang, A. Müller, S. Chaudhary, Y. Li, and X. Li, "ReAct: Synergizing reasoning and acting in language models," *arXiv preprint arXiv:2210.03629*, 2022.

[4] R. Nakano *et al.*, "WebGPT: Browser-assisted question-answering with language models," OpenAI, 2021.

[5] H. Chase, "LangChain: A framework for building applications with LLMs," GitHub repository, 2023.

[6] J. Lin *et al.*, "LlamaIndex: Data framework for LLM applications," GitHub repository, 2023.

[7] A. Cohan, C. Kong, D. Radev, Z. Lu, and R. Feldman, "SciTLDR: A Large-Scale Dataset for Scientific Paper Summarization," in *Proc. EMNLP*, 2020.

[8] H. Coulson and A. Polk, "Scholarcy: AI-Driven Summarisation of Academic Papers," *Journal of Scholarly Publishing*, vol. 52, no. 3, pp. 200–212, 2020.

[9] S. Olson and R. Nguyen, "SciBot: Automated Summarisation and Reference Extraction for Scientific Documents," in *Proc. AI for Science Conf.*, 2021.

[10] LangGraph Contributors, "LangGraph Documentation," GitHub repository, 2023. https://github.com/langgraph/langgraph