SynGraph: An Agentic Workflow for Deep Research

Berkay Orhan[1]

Abstract
SynGraph is a modular, agentic workflow for deep research that leverages a hierarchical LangGraph architecture to conduct end-to-end investigations. The system orchestrates a supervisor-worker pattern where a top-level graph manages request routing, clarification, and synthesis, while parallel researcher subgraphs execute targeted information retrieval. By integrating Google's Gemini 2.5 Pro for reasoning and Gemini Flash for high-throughput tasks, alongside Tavily and Exa for comprehensive web and semantic search, SynGraph achieves a balance of depth and speed. I present the architectural design, including typed state management and middleware-enforced budgets, and discuss how this approach mitigates common LLM hallucinations through rigorous citation tracking and iterative reflection.

Source code: https://github.com/berkay2002/researcher

Authors
[1] *Berkay Orhan,* `beror658@student.liu.se`

Keywords: LangChain — Deep Research — Multi-Agent Systems (MAS)

Contents

1. Introduction

Large language models (LLMs) have demonstrated remarkable capabilities in summarisation and question answering, yet they often struggle with multi-step research tasks that require sustained planning, deep information retrieval, and synthesis across disparate sources. Single-shot generation strategies frequently succumb to hallucinations or superficiality when faced with complex queries. To address these limitations, I introduce *SynGraph*, an agentic workflow built on LangGraph Campos, 2025 that formalises the research process as a structured, stateful program.

The backend of *SynGraph* implements a retrieval-augmented generation (RAG) workflow Lewis et al., 2021 that goes beyond simple vector search. It employs a directed cyclic graph to steer request routing, clarification, research execution, and report writing. This architecture coordinates multiple Gemini models—leveraging the reasoning power of Gemini 2.5 Pro and the speed of Gemini Flash—with distinct search providers to ensure both timeliness and semantic depth. By maintaining a strictly typed state and enforcing clear separation of concerns through subgraphs, SynGraph provides an auditable, configurable, and robust platform for academic and technical research.

2. Architecture Overview

SynGraph couples Google's Gemini family with modern search APIs to keep reasoning grounded and up to date. As defined in the system configuration, `gemini-2.5-pro` handles long-form analysis, including the critical "Supervisor" and "Researcher" roles, while `gemini-flash-latest` delivers low-latency structured outputs for planning, summarisation, and routing decisions. This hybrid model strategy optimises both cost and performance: complex reasoning tasks receive the full capacity of the Pro model, while high-volume, schema-constrained tasks are offloaded to the faster Flash variant.

The architecture follows a supervisor-worker pattern LangChain, 2025d. A central supervisor node creates a research plan and delegates sub-tasks to parallel researcher agents. Each researcher operates independently, gathering information and reporting back to the supervisor, which then aggregates the results. This parallelisation allows the system to explore multiple facets of a query simultaneously, significantly reducing the total time to insight compared to linear execution.
Figure 1 summarises how these architectural elements fit together, from routing and clarification through supervision and reporting.

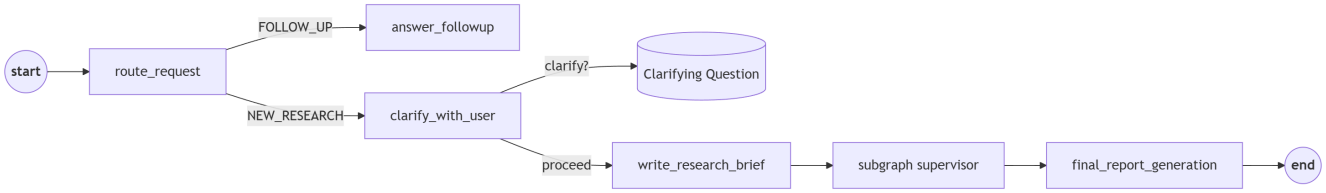

Figure 1. SynGraph research workflow. The main loop routes requests, clarifies intent, prepares a brief, and coordinates supervision before synthesising the final report.

3. Tooling and Services

The workflow relies on a toolkit of specialised research services. Dedicated connectors wrap the **Tavily** and **Exa** search APIs. Tavily is utilised for its ability to provide broad, up-to-the-minute web coverage, while Exa is leveraged for its semantic search capabilities, offering higher recall for academic and technical sources. These connectors are exposed as LangChain tools that record provider metadata—including query strings, timestamps, and run durations—directly into the graph state for later audit.

To ensure robustness, the system employs custom middleware. A usage-limiting middleware enforces budgets on agent actions, preventing runaway loops, while a fallback mechanism provides resilience by automatically retrying failed generations with alternative models. Furthermore, summarisation tools use structured output schemas to trim raw provider payloads before they enter the context window. This ”context engineering” LangChain, 2025b ensures that the limited context window is filled with high-value information, maintaining a high signal-to-noise ratio throughout the research session.

4. Workflow Execution

At runtime, a LangGraph state machine orchestrates the end-to-end flow with a persistent checkpointer, ensuring that every conversation can be replayed or inspected post hoc LangChain, 2025e. The graph execution begins with a conditional routing step: if a prior report exists, the system evaluates whether the user's new prompt is a follow-up or a fresh request. For new inquiries, the workflow transitions to a **Clarification** node, where the system analyses the ambiguity of the request. If the intent is unclear, the graph pauses execution using LangGraph interrupts LangChain, 2025c, soliciting direct feedback from the user before proceeding.

Once the research goal is solidified, a **Research Brief** node synthesises a comprehensive instruction set. This brief is passed to the **Supervisor** subgraph, which manages the parallel execution of **Researcher** agents. Each researcher performs iterative cycles of search, reading, and reflection Madaan et al., 2023; Shinn, Lyu, et al., 2023. The supervisor monitors these agents, enforcing a maximum iteration limit to prevent infinite loops. Finally, the **Report Generation** node aggregates the structured notes and citations from all researchers, producing a cohesive document. If the content exceeds the context window of the primary model, the system automatically falls back to a larger-context variant to ensure completion.

Figure 2 illustrates these decision points, highlighting when the system returns to prior work versus launching a new research cycle.

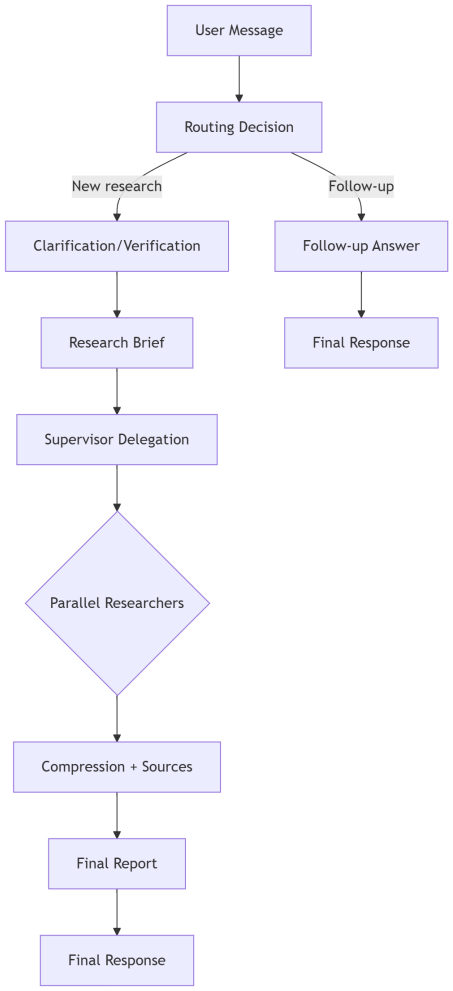

Figure 2. Control flow from user prompt to cited report. Routing handles follow-ups, while new questions traverse clarification, briefing, supervision, and synthesis stages.

5. State and Data Management

Typed state annotations make every mutation explicit LangChain, 2025a. Separate stores track conversation messages, supervi-

sor deliberations, researcher notes, compressed summaries, and source lists. Reducers distinguish between append-only logs and fields that should be overwritten, ensuring that prompts stay focused while tool outputs accumulate transparently. Because search tools log their observations directly into state, auditors can reconstruct which queries produced each citation.

6. Operational Considerations

All critical levers, such as search provider choice, parallelism, tool budgets, model selection, and follow-up confidence thresholds, are exposed as validated configuration options. Invocations can override these defaults to experiment with different mixes of Gemini models or swap between Tavily and Exa without modifying the workflow code.

Table 1. Key config options.

Setting	Controls
Search provider	Select Tavily, Exa
Parallelism	Number of agents
Tool budget	Max calls per agent
Clarification	Ask prep questions
Model fallback	Retry if failed

7. Discussion

Strengths. The modular graph architecture isolates responsibilities, making the system easier to debug and extend. The use of middleware for tool budgeting and model fallbacks significantly increases reliability in production environments. Furthermore, the separation of ”Supervisor” and ”Researcher” roles allows for specialised prompting, where the supervisor focuses on breadth and coordination while researchers focus on depth and verification.

Limitations. Despite the robust architecture, the system faces inherent challenges. Latency can be high during deep research cycles due to the sequential nature of some reflection steps and the sheer volume of tokens processed. Additionally, while the task-tracking tool helps track progress, complex queries may still result in redundant searches across parallel agents. The current implementation also lacks a persistent caching layer for search results, which could reduce costs for repetitive queries.

Opportunities. Future work will focus on integrating the Model Context Protocol (MCP) to allow dynamic attachment of domain-specific tools (e.g., internal databases, code repositories). I also plan to implement a more sophisticated rate limiter backed by persistent storage to handle higher concurrency levels. Finally, the existing frontend interface could be further enhanced to provide even more granular control over the research process during interrupts.

8. Conclusion

SynGraph demonstrates how agentic LangGraph programs can turn LLMs into transparent research workflows rather than opaque answer engines. By pairing validated configuration, reusable prompts, audited tools, and typed state, the backend consistently delivers cited reports while remaining adaptable to new domains and deployment constraints. The same architecture can support due diligence, literature reviews, or market analysis tasks that demand reproducible, reviewable reasoning.

Acknowledgments

Thanks to LangChain/LangGraph for foundational tools and documentation.

References

Campos, N. (2025). *Building langgraph: Designing an agent runtime from first principles*. LangChain Research Blog. Retrieved October 21, 2025, from https://blog.langchain.com/building-langgraph/

LangChain. (2025a). *Application structure (langgraph documentation)*. Retrieved October 20, 2025, from https://docs.langchain.com/oss/javascript/langgraph/application-structure

LangChain. (2025b). *Context engineering in agents (langchain documentation)*. Retrieved October 20, 2025, from https://docs.langchain.com/oss/javascript/langchain/context-engineering

LangChain. (2025c). *Interrupts (langgraph documentation)*. Retrieved October 20, 2025, from https://docs.langchain.com/oss/javascript/langgraph/interrupts

LangChain. (2025d). *Multi-agent systems (langchain documentation)*. Retrieved October 20, 2025, from https://docs.langchain.com/oss/javascript/langchain/multi-agent

LangChain. (2025e). *Persistence (langgraph documentation)*. Retrieved October 20, 2025, from https://docs.langchain.com/oss/javascript/langgraph/persistence

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.-t., Rocktäschel, T., Riedel, S., & Kiela, D. (2021). Retrieval-augmented generation for knowledge-intensive nlp tasks. https://arxiv.org/abs/2005.11401

Madaan, A., et al. (2023). Self-refine: Iterative refinement with llms. *arXiv preprint arXiv:2303.17651*.

Shinn, N., Lyu, Q., et al. (2023). Reflexion: An autonomous agent with dynamic memory and self-reflection. *arXiv preprint arXiv:2303.11366*.