SynGraph: An Agentic Workflow for Deep Research

Berkay Orhan[1]

Abstract
Standard Large Language Models (LLMs) frequently struggle with hallucinations and superficiality when faced with complex, multi-step research tasks. To address these reliability issues, I present *SynGraph*, a modular, agentic workflow 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 3 Pro for deep reasoning (utilizing explicit thinking processes) 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 detail the architectural design, including typed state management and middleware-enforced budgets, demonstrating how this approach mitigates common errors 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 1
2 Architecture Overview 1
3 Tooling and Services 2
4 Workflow Execution 2
5 State and Data Management 3
6 Operational Considerations 3
7 Evaluation & Discussion 3
8 Conclusion 3
Acknowledgments 3

1. Introduction

The promise of Artificial Intelligence lies in its ability to synthesise vast amounts of human knowledge instantly. However, standard Large Language Models (LLMs) frequently falter when faced with deep, multi-step research tasks. Single-shot generation strategies often crumble under the weight of complex queries, succumbing to hallucinations, superficiality, or a loss of coherence. To resolve this tension between potential and performance, 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 Pro and the speed of Gemini Flash—with distinct search providers to ensure both timeliness and semantic depth. Underpinning this coordination is a robust **State and Data Management** system; 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 Google DeepMind, 2025b with modern search APIs to keep reasoning grounded and up to date. As defined in the system configuration, `gemini-3-pro-preview` handles long-form analysis, including the critical "Supervisor" and "Researcher" roles, utilizing the model's native `thinking` capabilities Google AI Studio, 2025 to plan and reflect on search results. Meanwhile, `gemini-flash-latest` (based on the Gemini 2.5 architecture Google DeepMind, 2025a) is architected to deliver 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

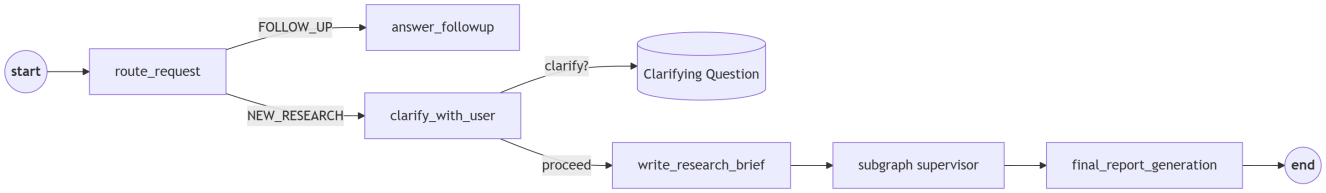

Figure 1. SynGraph research workflow. The main loop routes requests, clarifies intent, prepares a brief, and coordinates supervision before synthesising the final report.

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.

3. Tooling and Services

The workflow relies on a toolkit of specialised research services. Dedicated connectors wrap the **Tavily** Tavily, 2025 and **Exa** Exa, 2025 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. Additionally, a dedicated `think_tool` allows agents to record structured reflections and strategic plans before and after tool execution, enforcing a "Chain of Thought" process. 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. A `ResearchStandards` module further validates output quality against fixed academic benchmarks, such as citation density and source recency. 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 new query. For new queries, the workflow transitions to a **Clarification** node, where the system analyses the ambiguity of the request. The system may pause execution using LangGraph interrupts LangChain, 2025c during both the routing phase (if the distinction between follow-up and new research is ambiguous) and the clarification phase (if the research requirements are vague), soliciting direct feedback from the user before proceeding. Figure 2 illustrates these decision points, highlighting when the system returns to prior work versus launching a new research cycle.

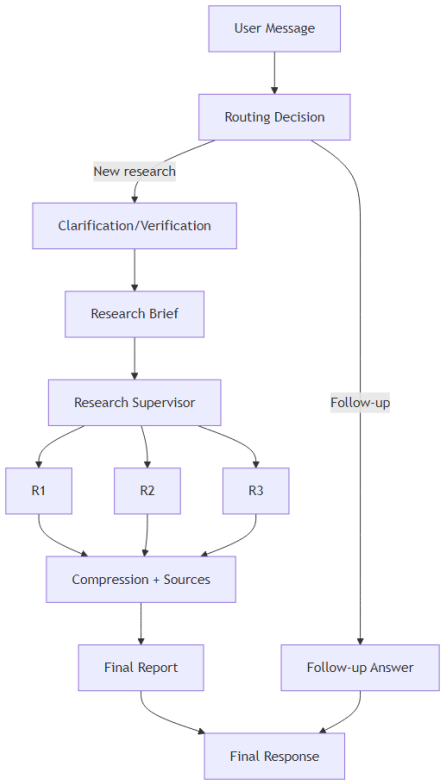

Figure 2. Control flow from user prompt to cited report. Routing handles follow-ups, while new queries traverse clarification, research brief, supervision, context compression, and report generation stages.

Once the research goal is solidified, a **Research Brief** node synthesises a comprehensive instruction set. This brief is passed to the **Research Supervisor** node, which manages the parallel execution of **Researcher** (eg. R1) 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 **Final Report** node aggregates the structured notes and citations from all researchers. The workflow employs a context compression step via the **Compression** node to synthesise intermediate findings before producing the **Final Response**. If the content exceeds the context window of the primary model, the system automatically falls back to a larger-context variant to ensure completion.

5. State and Data Management

To keep the research process organised, SynGraph manages data like a shared digital notebook rather than a simple chat log. Technically, this is achieved through typed state annotations LangChain, 2025a, which define exactly what kind of information can be stored. The system maintains distinct ”pages” for different types of data: one for the conversation history, another for the supervisor’s strategic plans, and separate sections for the raw notes and compressed summaries gathered by researchers.

This structured approach solves the problem of ”context pollution.” Instead of feeding every raw search result into the next step, the system uses ”reducers”—functions that decide how to update the state. Some updates simply append new information (like adding a new source), while others overwrite outdated data (like updating the current step of the plan). This ensures that agents always have a clear, focused view of the task at hand. Furthermore, because every tool usage is automatically logged into this state, the final report includes a transparent audit trail, allowing readers to trace any citation back to the specific query that uncovered it.

6. Operational Considerations

All critical settings, 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. Evaluation & 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 fully integrating the Model Context Protocol (MCP) to allow dynamic attachment of domain-specific tools (e.g., internal databases, code repositories); configuration support for MCP is already implemented in the system. 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/

Exa. (2025). *Exa: The search engine for ai*. Retrieved October 20, 2025, from https://exa.ai/

Google AI Studio. (2025). *Thinking capabilities in gemini models*. Retrieved October 20, 2025, from https://ai.google.dev/gemini-api/docs/thinking

Google DeepMind. (2025a). *Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities* (Technical Report) (Released June 2025). Google DeepMind. https://storage.googleapis.com/deepmind-media/gemini/gemini_v2_5_report.pdf

Google DeepMind. (2025b). *Gemini 3 pro model card* (Model Card). Google DeepMind. https://storage.googleapis.com/deepmind-media/Model-Cards/Gemini-3-Pro-Model-Card.pdf

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*.

Tavily. (2025). *Tavily api: Search engine for ai agents*. Retrieved October 20, 2025, from https://tavily.com/