Project Report

Agentic Blog Writing System Using LangGraph

---

1 Introduction

This project implements a multi-agent autonomous blog generation system built using LangGraph.

Unlike traditional LLM applications that rely on a single prompt-response interaction, this system is architected as a state-driven multi-agent pipeline where multiple specialized agents collaborate to generate publication-ready blog content.

The system integrates:

- **Strategic routing decisions**
- **Optional web-grounded research**
- **Structured task decomposition**
- **Parallel section generation**
- **AI-powered diagram generation**
- **Real-time UI streaming**
- **Downloadable Markdown outputs**

The goal of this project is to demonstrate practical multi-agent orchestration using large language models and tool integrations.

---

2 Problem Statement

Standard LLM blog generators suffer from:

- **No planning stage**
- **No research grounding**
- **Hallucinated citations**
- **Poor structural consistency**
- **No modular content generation**
- **No visual augmentation**

This project solves these issues by introducing:

- **Agent specialization**
- **Structured outputs**
- **Tool integration**
- **Conditional execution**
- **Parallelism**
- **Deterministic merging**

3️⃣ System Overview

The system follows this execution pipeline:

User Input (Topic, Date)

↓

Router Agent

↓

(Conditional)

Research Agent

↓

Orchestrator Agent

↓

Parallel Worker Agents

↓

Reducer Subgraph

├─ Merge Content

├─ Image Planning Agent

└─ Image Generation Agent

↓

Final Markdown + Images

The orchestration is handled by LangGraph, which manages state transitions between agents.

4️⃣ Core Architectural Components

4.1 LangGraph State

The system is built around a shared `State` object.

The state acts as the memory layer across all agents.

It includes:

- **topic**
- **mode**
- **needs_research**
- **queries**
- **evidence**
- **plan**
- **sections**
- **merged_md**
- **image_specs**
- **final**

Each agent:

- **Reads from state**
- **Writes back to state**
- **Influences downstream execution**

This design ensures coordination across multiple agents.

4.2 Router Agent (Strategic Decision Layer)

The Router Agent is the first node executed.

Its responsibilities:

- **Analyze the topic**
- **Determine whether research is required**
- **Classify execution mode**

The system defines three modes:

🔹 closed_book

Used for evergreen topics that do not require recent information.

🔹 hybrid

Used when topic is mostly conceptual but benefits from recent examples.

🔹 open_book

Used for volatile topics such as:

- **Latest news**
- **Weekly updates**
- **Policy changes**
- **Pricing updates**

The router outputs:

- **mode**
- **needs_research**
- **queries**
- **recency window**

This decision controls the entire pipeline.

4.3 Research Agent (Tool-Integrated Agent)

Activated only when `needs_research = True`.

Responsibilities:

1. **Call Tavily API**
2. **Fetch search results**
3. **Extract structured evidence using LLM**
4. **Deduplicate by URL**
5. **Apply recency filtering**

In `open_book` mode, evidence is restricted to recent results (e.g., last 7 days).

This prevents outdated or hallucinated citations.

4.4 Orchestrator Agent (Planning Layer)

This agent performs task decomposition.

Instead of writing the blog directly, it generates a structured plan:

- **Blog title**
- **Audience**
- **Tone**
- **Blog type**
- **5–9 Tasks**

Each Task includes:

- **Section title**
- **Goal**
- **Bullet points**
- **Target word count**
- **Citation requirement**
- **Code requirement**

This stage improves structure and coherence.

4.5 Parallel Worker Agents

For each task, a worker agent is spawned using LangGraph's `Send()`.

Each worker:

- **Writes exactly one section**
- **Follows bullet instructions**
- **Meets word count constraints**
- **Cites only approved evidence URLs**
- **Includes code if required**

Workers execute independently and in parallel.

This improves modularity and scalability.

4.6 Reducer Subgraph

After all workers finish, a reducer subgraph runs.

This subgraph consists of three internal nodes:

4.6.1 Merge Node

- **Sorts sections by ID**
- **Combines them into full Markdown**
- **Adds blog title**

4.6.2 Image Planning Agent

Reads full blog content and decides:

- **Whether images are needed**
- **Maximum of 3 images**
- **Insert placeholders (e.g., [[IMAGE_1]])**
- **Create structured image specifications**

This is an editorial reasoning step.

4.6.3 Image Generation Agent

For each image specification:

- **Calls Gemini Image Model**
- **Saves image to `images/`**
- **Replaces placeholder with Markdown image tag**

Graceful fallback is implemented in case image generation fails.

5️⃣ Frontend (Streamlit UI)

The frontend allows:

- **Topic input**
- **As-of date selection**
- **Real-time execution streaming**
- **State summary display**
- **Evidence table**
- **Markdown preview with images**

- **Download Markdown**
- **Download image bundle**
- **Load past blogs**

It interacts with the compiled LangGraph app.

---

6 Multi-Agent Characteristics

This project is multi-agentic because:

- **It contains multiple autonomous decision-making units**
- **Each unit has a specialized responsibility**
- **Agents interact through shared state**
- **There is conditional branching**
- **Tool usage is integrated**
- **Parallel execution is implemented**
- **Execution is not linear**

This is not prompt chaining.
It is orchestrated collaboration.

---

7 Technology Stack

- **LangGraph – Multi-agent orchestration**
- **OpenAI GPT-4.1-mini – Reasoning & writing**
- **Tavily API – Web research**
- **Gemini Image Model – Diagram generation**
- **Streamlit – Frontend UI**
- **Pydantic – Structured LLM outputs**

---

8 Key Design Decisions

Structured Outputs

All major agents use Pydantic schemas to prevent malformed responses.

Conditional Execution

Research runs only when required.

Parallelism

Section writing is parallelized using LangGraph fan-out.

Recency Filtering

Open-book topics are restricted to recent evidence.

Tool Isolation

Workers only cite pre-approved evidence.

Graceful Failure Handling

Image generation failures do not break pipeline.

9️⃣ Strengths of the System

- **Modular design**
- **Scalable architecture**
- **Strong hallucination control**
- **Clean separation of concerns**
- **Tool integration**
- **UI transparency**
- **Deterministic reducer logic**

🔟 Limitations

- **No reflection/critic loop**
- **No persistent memory across sessions**
- **No SEO optimization layer**
- **No automated scheduling**
- **No citation validation pass**

1️⃣1️⃣ Future Enhancements

- **Add Critic Agent for quality scoring**
- **Add citation verification module**
- **Add autonomous publishing scheduler**

- **Add SEO optimization agent**
- **Add long-term memory for writing style**
- **Convert to API-based backend deployment**

1️⃣2️⃣ Conclusion

This project demonstrates a practical implementation of multi-agent orchestration using LangGraph.

It moves beyond simple prompt engineering and introduces:

- **Decision agents**
- **Planning agents**
- **Tool-using agents**
- **Parallel workers**
- **Reducer subgraphs**
- **State-driven coordination**

The system showcases how structured agent collaboration can produce high-quality, grounded, and modular AI-generated content.

It serves as a strong demonstration of:

- **Agentic system design**
- **LLM orchestration**
- **Tool integration**
- **Parallel execution**
- **Production-style architecture**
