Architecture & Agent Design Report

LangGraph Agentic Workflow Project

Prepared by: Joris Jose

Date: 14 September 2025

1. Technical Approach & LangGraph Components Used

The project was redesigned to use **LangGraph** for orchestrating agentic workflows. Instead of linear pipelines, LangGraph enables graph-based execution where each node is an agent and edges define information flow. This improves flexibility, memory persistence, and error handling.

Component	Purpose	How It Is Applied
LangGraph Agent Nodes	Define modular capabilities	Agents for retrieval, NLP, summarization, and synthesis.
Memory Module	State persistence	src/memory/persistence.py ensures results/logs are stored across sessions.
Tool Integration	External service calls	Connectors to GCP NLP API, Vertex AI, Gemini for entity & summarization.
Custom Tools	Supporting functions	Data prep, report generation, and memory setup under src/tools/

2. Results, Challenges, and Trade-offs

Results:

- Clearer orchestration with LangGraph's state graph. - Memory persistence allowed continuity across multiple steps. - Seamless GCP integration maintained while benefiting from agent graph logic.

Challenges:

Challenge	Description
Restructuring	Adapting from linear workflows to graph-based orchestration.
Debugging	Tracing execution across multiple agents was more complex.
Compatibility	Aligning LangGraph memory states with GCP API responses.

Trade-offs:

- **LangGraph**: Superior modularity, state management, but added complexity. - **Google-only pipelines**: Simpler, faster to implement, but lacked flexibility and resilience.

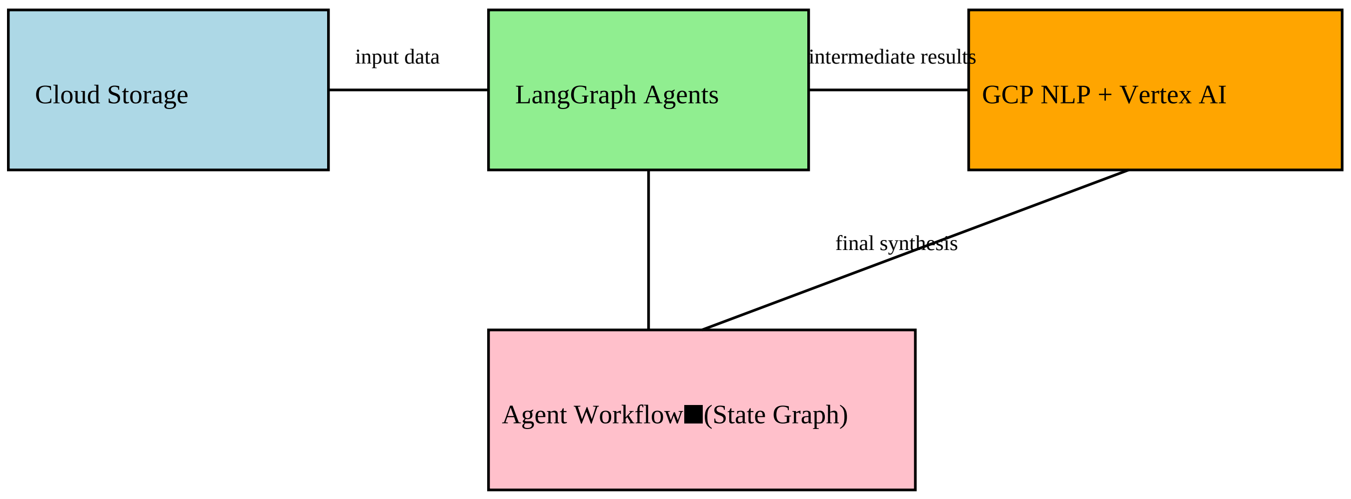
3. Agentic Workflow & LangGraph Architecture

The workflow is modeled as a **LangGraph state graph**. Each state transition maps to an agent action, providing explicit routing, retries, and fallbacks. This design is resilient, modular, and extensible.

Stage	Description
Data Preparation	Raw data cleaning and formatting via data_prep.py.
Retrieval	Ranking candidates with keyword/FAISS retrieval agents.
NLP Analysis	Entity & sentiment extraction via GCP NLP API.

Summarization	Summaries generated by Vertex AI or Gemini models.
Synthesis	LangGraph merges results into a coherent final output.
Persistence	Memory nodes record analysis state and outputs.

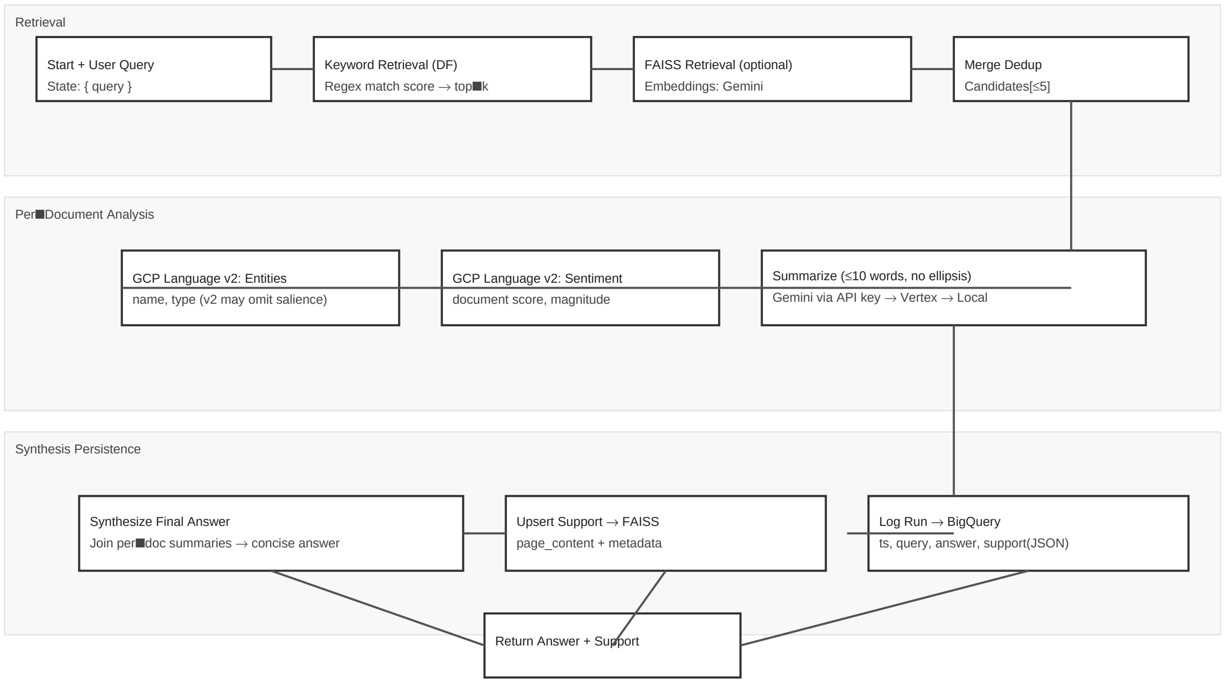
4. High-level LangGraph + GCP Architecture



5. Agent Flow Diagram

This diagram illustrates the retrieval → analysis → synthesis process as modeled in LangGraph. It highlights branching, memory persistence, and resilience mechanisms.

Agent Flow: Retrieve → Analyze → Synthesize (with Fallbacks Memory)


Retrieval
Start + User Query
State: { query }
Keyword Retrieval (DF)
Regex match score → top■k
FAISS Retrieval (optional)
Embeddings: Gemini
Merge Dedup
Candidates[≤5]
Per■Document Analysis
GCP Language v2: Entities
name, type (v2 may omit salience)
GCP Language v2: Sentiment
document score, magnitude
Summarize (≤10 words, no ellipsis)
Gemini via API key → Vertex → Local
Synthesis Persistence
Synthesize Final Answer
Join per■doc summaries → concise answer
Upsert Support → FAISS
page_content + metadata
Log Run → BigQuery
ts, query, answer, support(JSON)
Return Answer + Support

Errors are captured per step; agent continues with available signals (graceful degradation).