# **Consolidated Technical Design Specification**

Version: 1.0

Status: As-Built (reflects v2 codebase)

## **1. Architectural Overview**

This document provides the definitive technical specification for the implemented v2 agentic system. The system is a stateful, iterative research agent built in Python using the Streamlit and LangGraph frameworks.

The core architecture follows a Plan -> Synthesize -> Evaluate -> Refine workflow orchestrated by a central Supervisor. A Student agent performs the planning and synthesis tasks, while an independent Teacher agent performs the evaluation. Human-in-the-Loop (HITL) gates after the planning and evaluation phases ensure user control and supervision. The entire system is designed to be configurable via a central prompts.yaml file.

Key Components:

* supervisor\_v2.py: The main Streamlit application entry point. It manages the UI, orchestrates the sequence of graph invocations, and handles the state transitions for the iterative feedback loop.
* agent\_core\_v2.py: A library containing the core logic and StateGraph definitions for the three distinct computational graphs: planning\_graph, synthesis\_graph, and evaluation\_graph.
* prompts.yaml: A configuration file containing all system and node-level prompts that guide the behavior of the LLMs.

## **2. State Management**

The system operates on a central state object defined by the GraphState TypedDict in agent\_core\_v2.py.

class GraphState(TypedDict):

user\_prompt: str # The initial high-level request from the user.

documents: List[Dict] # Documents retrieved from the knowledge base.

plan: Optional[Dict] # The JSON plan generated by the planning graph.

output: str # The final Markdown report generated by the synthesis graph.

evaluation\_report: Optional[Dict] # The JSON audit report from the evaluation graph.

user\_feedback: Optional[str] # User's text feedback for the refinement loop.

messages: List[BaseMessage] # Used for internal message passing in some graph designs.

working\_memory: Optional[Dict] # Stores content of individual sections during synthesis.

completed\_plan\_items: List[str] # Tracks progress through the synthesis plan.

parsed\_claims: List[Dict] # Stores claims extracted from the output for auditing.

## **3. Computational Graphs**

The system logic is modularized into three distinct, single-agent graphs.

### **3.1. Planning Graph**

* File: agent\_core\_v2.py, build\_planning\_graph()
* Purpose: To create a structured JSON plan based on the user's request and retrieved documents.
* Nodes:
  1. retrieve\_context: Fetches relevant documents from the FAISS vector store.
  2. generate\_plan: Takes the user prompt and context, along with any feedback from a previous run, and invokes the Student LLM with the create\_plan\_prompt to generate a JSON object representing the report's table of contents.

### **3.2. Synthesis Graph**

* File: agent\_core\_v2.py, build\_synthesis\_graph()
* Purpose: To execute the user-approved plan and generate a complete, cited report.
* Nodes:
  1. execute\_synthesis\_step: In a loop, this node generates the content for a single section from the plan. The LLM is guided by the synthesis\_step\_prompt, which mandates the inclusion of [Source: ...] tags.
  2. assemble\_draft: After the loop completes, this node assembles the individual sections from working\_memory into the final output string. It then programmatically adds a unique numeric index to every citation tag (e.g., [Source: ...][1]) for traceability.
* Routing: A conditional edge checks if all plan items are complete to decide whether to loop execute\_synthesis\_step or move to assemble\_draft.

### **3.3. Evaluation Graph**

* File: agent\_core\_v2.py, build\_evaluation\_graph()
* Purpose: To perform a tool-augmented audit of the synthesized report.
* Structure: This graph has been simplified to a single primary node for robustness.
* Nodes:
  1. generate\_evaluation\_node: This is the sole node. The Teacher LLM provided to it is pre-bound with the citation\_retriever tool. The node is guided by the evaluation\_prompt\_v2, which instructs the LLM to perform the entire audit process internally (read the report, use the tool to verify all indexed citations, assess consistency/alignment) and then generate a single, final JSON object as its output.

## **4. Iterative Refinement Loop**

* File: supervisor\_v2.py, main() function within the FINAL\_REVIEW phase.
* Purpose: To allow the user to guide revisions of the generated report.
* Logic:
  1. The UI displays the final draft and the evaluation report.
  2. An expander allows the user to enter text feedback.
  3. When the "Re-Plan with This Feedback" button is clicked, the supervisor captures the current evaluation\_report and the new user\_feedback.
  4. It resets the current\_state, preserving the original user\_prompt but adding the captured feedback.
  5. It sets the run\_phase back to None, which re-triggers the planning graph.
  6. The generate\_plan node detects the presence of feedback in the state and amends its prompt to the LLM, ensuring the new plan is designed to correct the previous version's flaws.