# **Technical Design Specification: A Robust, Iterative Agentic System**

Version: 4.0

Date: 16 June 2025

Status: Implemented (Based on supervisor\_v2.py and agent\_core\_v2.py)

## **1. Architectural Overview**

This document provides the technical design for the stable v2 agentic system, which implements a full Plan -> Synthesize -> Evaluate -> Refine workflow. This architecture is defined by three key systems: a stateful iterative feedback loop managed by the Supervisor, programmatic citation indexing for traceability, and a robust, single-call, tool-augmented evaluation graph. This specification serves as the definitive guide to the implemented v2 codebase.

* Supervisor (supervisor\_v2.py): Orchestrates the four-phase workflow. It is responsible for initializing a session-based logger, managing the Human-in-the-Loop (HITL) gates, and crucially, capturing the evaluation report and user feedback to inject into the state for the refinement loop.
* Agent Core (agent\_core\_v2.py): Contains the logic for the three agentic graphs. The build\_planning\_graph now implicitly handles feedback by checking for its presence in the state. The build\_synthesis\_graph now includes a programmatic step to index citations. The build\_evaluation\_graph is simplified to a single, robust node.
* Configuration (prompts.yaml): Contains all prompts, including the comprehensive evaluation\_prompt\_v2 that guides the single-call evaluator.

## **2. System Components and Logic**

### **2.1. Iterative Feedback Loop**

The refinement cycle is managed by the Supervisor and the Planning Graph.

* Supervisor Logic (supervisor\_v2.py): In the FINAL\_REVIEW phase, the UI presents an expander for user feedback. If the "Re-Plan with This Feedback" button is clicked, the supervisor creates a new initial state for the next run. This new state preserves the original user\_prompt but critically adds the evaluation\_report and the new user\_feedback text [cite: supervisor\_v2.py]. The run\_phase is then reset to None, triggering a new planning cycle.
* Planning Node Logic (agent\_core\_v2.py): The generate\_plan node now begins by checking if user\_feedback or evaluation\_report exists in the GraphState. If so, it dynamically prepends a detailed prefix to its main prompt, instructing the LLM that it is in a revision cycle and MUST address the provided critiques. This ensures feedback is prioritized [cite: agent\_core\_v2.py].

### **2.2. Citation Indexing**

To enhance human readability and traceability of the audit, citation indices are added programmatically post-synthesis.

* Synthesis Node Logic (agent\_core\_v2.py): Within the build\_synthesis\_graph, the assemble\_draft function is responsible for this task. After assembling the full draft from the working\_memory, it uses a regular expression (re.sub) to find all instances of [Source: ...] tags and appends a sequential numeric index (e.g., [1], [2]) to each one. This ensures that every citation in the final output has a unique identifier that can be cross-referenced with the audit report [cite: agent\_core\_v2.py].

### **2.3. Simplified Evaluation Graph**

The evaluation graph is simplified to a single, powerful node to enhance robustness, as documented in TRR v19.0.

* Graph Structure (agent\_core\_v2.py): The build\_evaluation\_graph function now constructs a graph with only one primary node: generate\_evaluation. There are no conditional edges or intermediate state-passing steps [cite: agent\_core\_v2.py].
* Node Logic (generate\_evaluation\_node):
  1. The node receives the teacher\_llm instance, which has been pre-bound with the citation\_retriever tool by the build\_evaluation\_graph function.
  2. It uses the evaluation\_prompt\_v2 from prompts.yaml, which instructs the LLM to perform the entire reasoning chain (read report, use tools to verify citations, assess consistency, check goal alignment) in one internal process.
  3. The LLM uses the citation\_retriever tool as needed to gather evidence.
  4. The LLM's final action is to synthesize all its findings directly into the required JSON format as specified in the prompt.
  5. The node parses this JSON from the LLM's response and places it into the evaluation\_report key in the GraphState.