# **Technical Research Report: A Simplified, Single-Call, Tool-Augmented Evaluator**

Version: 19.0

Date: 16 June 2025

## **1. Abstract**

This document details a significant architectural refinement of the Teacher-Student model, moving from a complex, multi-node evaluation graph to a simplified, single-call, tool-augmented evaluator. Previous theoretical designs (TDS v3.1, v3.3) and research reports (TRR v16, v18) described a granular evaluation process involving separate nodes for parsing claims, verifying citations, checking consistency, and assembling a report. While theoretically sound, this approach proved fragile in practice, creating multiple points of failure related to state passing between nodes. The implemented and stable v2 architecture (agent\_core\_v2.py) corrects this by collapsing the entire evaluation process into a single, robust node (generate\_evaluation\_node). This node is bound to the necessary tools (e.g., citation\_retriever) and guided by a comprehensive prompt (evaluation\_prompt\_v2) that instructs the LLM to perform the entire multi-step reasoning process internally and return a single, complete JSON artifact. This pivot represents a maturation of the architecture, moving from micro-managing the LLM's process to entrusting it with a holistic task, resulting in a more resilient, efficient, and debuggable system.

## **2. Research: The Fragility of Multi-Node LLM Chains**

The evolution from TRR v16 to TRR v18 established the necessity of a Plan-Synthesize-Evaluate loop with a tool-using "Teacher" agent. The corresponding design specifications (TDS v3.1, v3.3) laid out a logical but complex implementation for the evaluation graph, involving a chain of distinct responsibilities:

1. A node to parse claims from the report.
2. A conditional node to check for tools.
3. A ToolNode to execute verification calls.
4. A final node to synthesize the results into a JSON object.

However, practical implementation revealed a critical flaw in this design: state passing in multi-node LLM graphs is inherently brittle. Errors frequently occurred when the output of one node (e.g., the ToolNode) was not perfectly formatted for the input of the next node (e.g., the finalize\_evaluation\_json node), leading to empty states and API errors, such as the previously observed InvalidArgument: 400 \* GenerateContentRequest.contents: contents is not specified.

The successful implementation in agent\_core\_v2.py demonstrates a more effective pattern. Instead of using a graph to force a rigid, step-by-step reasoning process, we leverage the advanced reasoning capabilities of modern LLMs (like Gemini 1.5 Pro) to perform this process internally.

## **3. The v19 Architecture: Single-Call, Tool-Enabled Evaluation**

The stable v2 architecture refactors the evaluation graph into a single, powerful node that is both simpler and more robust.

The generate\_evaluation\_node:

* Unified Responsibility: This single node is responsible for the entire evaluation task.
* Tool Binding: The LLM client passed to this node is pre-emptively bound with the necessary tools (i.e., citation\_retriever) [cite: agent\_core\_v2.py].
* Comprehensive Prompting: The node is guided by a single, detailed prompt (evaluation\_prompt\_v2 from prompts.yaml). This prompt instructs the LLM on the full chain of reasoning it must perform internally: parse claims, use the citation\_retriever tool for each claim, compare the results, assess consistency and goal alignment, and finally, structure all of its findings into a single JSON object.

This architectural shift offers several advantages:

* Robustness: It eliminates all intermediate state-passing failures within the graph. The only state passed is the initial report and the final evaluation.
* Efficiency: It reduces the number of LLM calls required, as the model can perform its full reasoning chain in one pass.
* Clarity and Maintainability: The logic for the entire evaluation process is now consolidated into a single, clear prompt (evaluation\_prompt\_v2), making it much easier to understand, debug, and refine than a complex chain of graph nodes.

This refined architecture, as implemented in agent\_core\_v2.py, represents the most effective and stable version of the system, successfully fulfilling the goals laid out in TRR v18 while overcoming the practical challenges of earlier designs.