Interim Architecture Report: The Automaton Auditor

Submission Date: Wednesday (Pre-21hr UTC)

Status: Phase 1 Complete (Forensic Layer Operational)

1. Objectives & Deliverables

The primary objective of this system is to transition from "Vibe-based" code review to **Forensic Governance**. The system is designed to audit LangGraph implementations by verifying structural compliance rather than just text presence.

2. Core Architectural Decisions

A. Pydantic State vs. Standard Dicts

We implemented the AgentState using strictly typed **Pydantic models** (Evidence, JudicialOpinion, AuditReport).

- **Why:** Standard dictionaries allow for "schema drift" where agents might hallucinate field names. Pydantic enforces a contract, ensuring the Chief Justice node receives data in a predictable format for deterministic scoring.
- **Reducers:** We utilized Annotated[dict, operator.ior] for evidence and Annotated[list, operator.add] for opinions to enable **Parallel Fan-Out**. This prevents agents from overwriting each other's findings during concurrent execution.

B. Structural AST Parsing (The RepoInvestigator)

Instead of brittle Regex searches, our RepoInvestigator utilizes Python's ast (Abstract Syntax Tree) module.

- **Forensic Rigor:** The system parses the code into a tree to identify actual StateGraph instantiations and .add_node method calls. This ensures that we don't return a "False Positive" if a developer simply mentions "StateGraph" in a comment.
- **Safety & Sandboxing:** All cloning occurs within a tempfile.TemporaryDirectory. This ensures the host environment remains clean and prevents the execution of potentially malicious untrusted code during the audit.

C. Docling & Structural Document Analysis

We integrated **Docling** to convert architectural PDFs into structured Markdown.

- **Contextual Retrieval:** By converting to Markdown, the DocAnalyst can identify requirements based on header hierarchy (e.g., finding text specifically under the

"Constraints" header), providing higher accuracy than simple keyword search.

3. Implementation Status (Phase 1)

- **State Schema:** Fully implemented with official Pydantic models.
- **Parallel Swarm:** Graph configured for concurrent Detective execution.
- **AST Detective:** Capable of mapping graph structure.
- **Vision Slot:** Image extraction from PDF is functional; vision-model inference is ready for Phase 2 integration.

4. Constraints & Known Gaps

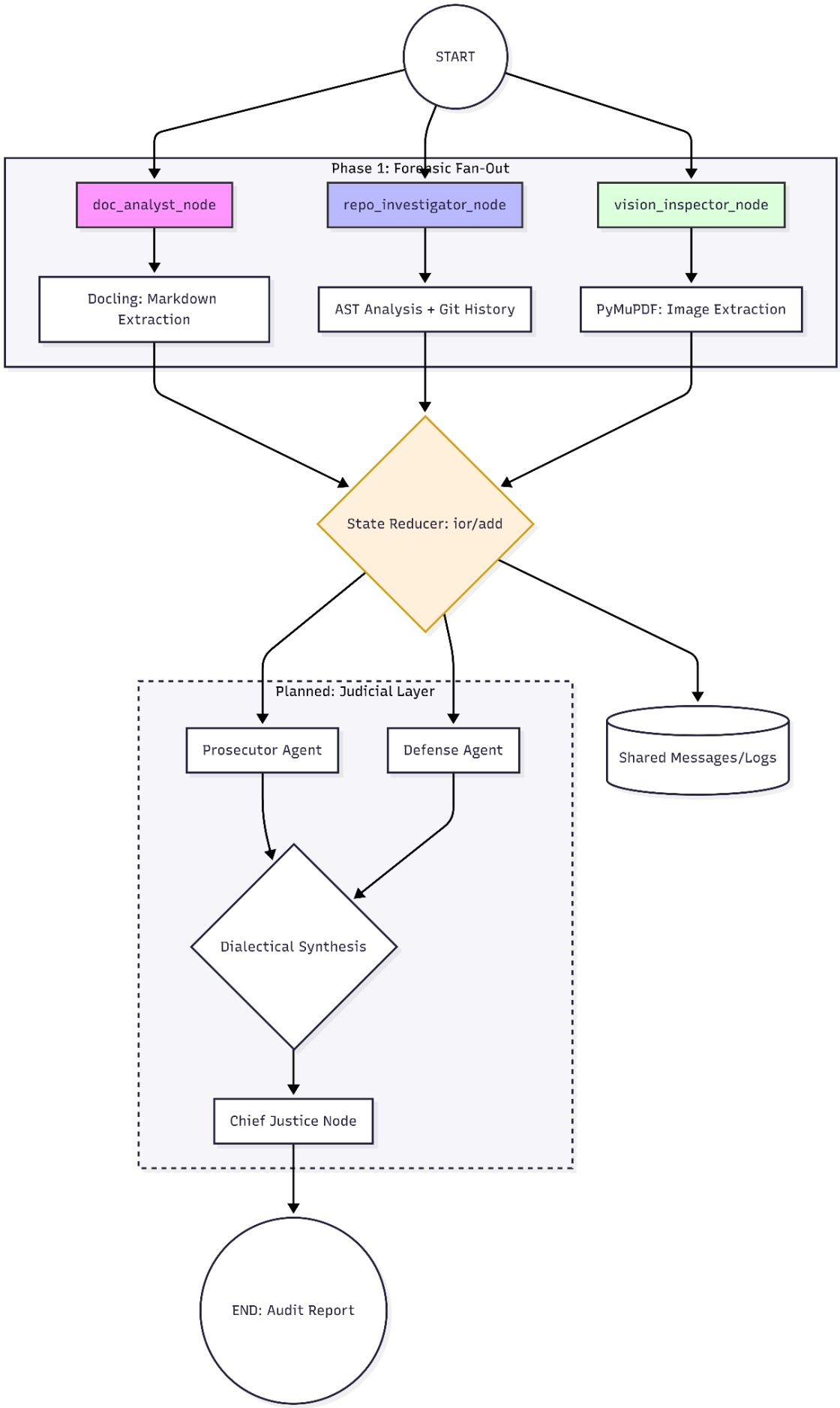
- **Inference Latency:** Parallel execution helps, but deep AST analysis of large repos can be time-intensive.
- **Judicial Layer:** Currently, the graph ends after the detective phase. The synthesis engine (Prosecutor/Defense debate) is the immediate next priority.

5. Concrete Plan for Phase 2

1. **Judicial Logic:** Implement a JudgeNode that takes the evidences dictionary and generates a JudicialOpinion based on the extracted rubric_dimensions.
2. **Dialectical Synthesis:** Configure the Chief Justice node to resolve conflicts when the Prosecutor and Defense scores vary by more than 2 points (as required by the schema).
3. **Final Formatting:** Automate the export of the AuditReport into a professional Markdown file in the /reports directory.

Architectural Workflow

The diagram below illustrates the current Phase 1: Forensic Fan-Out implementation and the planned Phase 2: Judicial Layer, highlighting how evidence is synthesized into a final audit report.


START
Phase 1: Forensic Fan-Out
doc_analyst_node
repo_investigator_node
vision_inspector_node
Docling: Markdown Extraction
AST Analysis + Git History
PyMuPDF: Image Extraction
State Reducer: ior/add
Planned: Judicial Layer
Prosecutor Agent
Defense Agent
Shared Messages/Logs
Dialectical Synthesis
Chief Justice Node
END: Audit Report