Interim Report: The Automaton Auditor

Date: February 25, 2026

Status: Phase 1 Complete (Forensic Layer Operational)

---

1. Executive Summary & Forensic Objective

The **Automaton Auditor** transitions code review from subjective assessment to **Forensic Governance**. By utilizing a "Swarm of Specialists," the system verifies LangGraph implementations through structural truth (AST) rather than probabilistic text matching.

---

2. Core Architectural Decisions & Rationale

A. Typed State & Reducer Logic

We utilize a strictly typed AgentState via Pydantic to enforce a schema contract across all nodes.

- **Rationale:** Standard dictionaries are prone to "schema drift." Pydantic ensures the **Chief Justice** receives deterministic data.
- **Concurrency Guard:** We utilize Annotated[Dict, operator.ior] for evidence aggregation. This allows multiple detectives to write to the state simultaneously without race conditions, merging results into a unified forensic record.

B. Structural AST Parsing vs. Regex

The RepoInvestigator uses Python's ast module to map the target repository.

- **Forensic Rigor:** By parsing code into an Abstract Syntax Tree, the auditor identifies actual StateGraph instantiations and method calls (e.g., .add_node). This eliminates "False Positives" found in comment-based regex searches.

---

3. StateGraph Architecture & Data Flow

To achieve maximum robustness, the graph follows a **Parallel Fan-Out/Fan-In** topology.

Architecture Diagram Logic:

Phase	**Node Type**	**Data Label / State Type**	**Edge Logic**
Start	Entry Point	repo_url: str, pdf_path: str	Deterministic
Fan-Out	Detectives	Dict[str, List[Evidence]]	**Parallel Execution**
Fan-In	**Aggregator**	EvidenceAggregator (Unified State)	**Sync Point**
Judicial	Judges	List[JudicialOpinion]	**Conditional (Score Variance)**

Robustness Enhancement: The transition from the **Aggregator** to the **Judicial Layer** is protected by **Conditional Edges**. If the Evidence object is empty or a tool failure is detected (e.g., 404 Repo), the graph routes to an **Error Recovery Node** rather than moving to judgment.

---

4. Gap Analysis & Risk Mitigation (Phase 2)

While Phase 1 establishes the forensic track, the following risks and plans are identified to prevent "vibe-based" drift:

A. Identified Risks & Failure Modes

- **LLM Persona Drift:** In the judicial phase, specialized agents (Prosecutor/Defense) may lose their critical edge. **Mitigation:** We implement "System-Prompt Anchoring" and few-shot forensic examples to maintain dialectical tension.
- **Confidence Hallucination:** Agents might overstate the certainty of found evidence. **Mitigation:** We enforce a confidence constraint of ge=0, le=1 in the Pydantic schema to normalize scoring.

B. Forward Plan: The Judicial Layer

1. **JudgeNodes:** Specialized agents will take the evidences dictionary and cross-reference it against rubric_dimensions.
2. **Chief Justice Synthesis:** A central node resolves conflicts. If the variance between the Prosecutor and Defense scores exceeds 2 points, a **Dissent Summary** is triggered for manual human review.

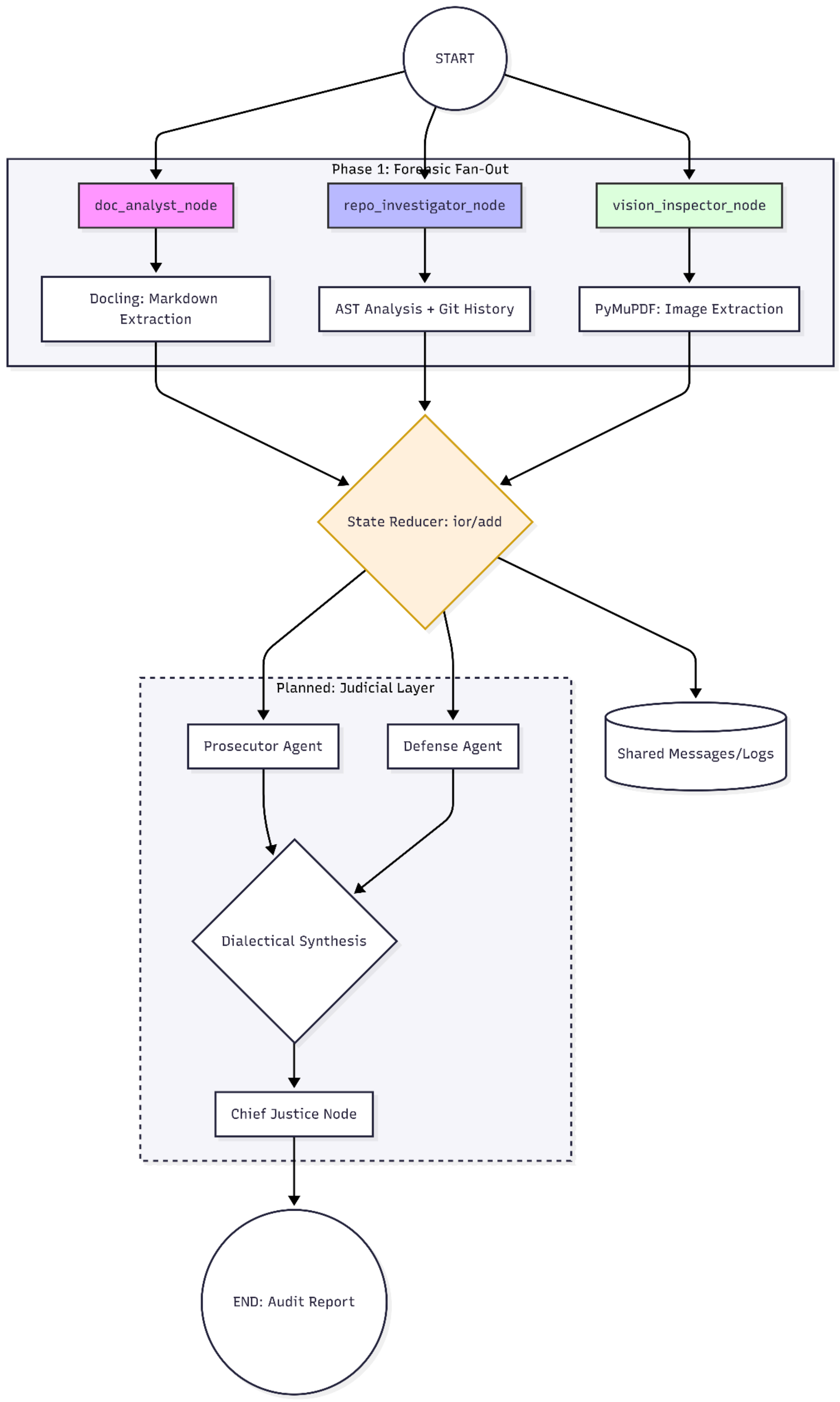
5. Infrastructure & Reproducibility

The project is built for professional "Developer Experience" (DevEx):

- **Reproducibility:** A uv.lock file is provided to ensure identical dependency versions across environments.
- **Environment:** Verified for **Python 3.12+**.
- **Execution:** Automated via a single command line interface supporting arbitrary GitHub URLs and PDF inputs.

6. 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


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

report.