Forensic Audit Report: The Automaton Auditor

Author: Yakob

Date: February 2026

Repository: https://github.com/yakobd/automaton_auditor_project_tenx.git

---

1. Executive Summary

The **Automaton Auditor** is a multi-agent LangGraph system designed for high-fidelity repository forensics. Unlike static linters, this system employs a **Diamond Architecture** that simulates a digital courtroom. It analyzes source code via AST (Abstract Syntax Tree) parsing, evaluates documentation using RAG-lite retrieval (via Docling), and resolves scoring through a three-judge dialectical panel. The system is hardened with subprocess sandboxing to ensure secure execution during the cloning and analysis phases.

---

2. Architecture Deep Dive

Dialectical Synthesis

The core of our judicial logic is not a simple average of scores. Instead, we use **Dialectical Synthesis**. The Prosecutor node is prompted to find "Critical Fails" (e.g., shell injections), while the Defense node is prompted to find "Pragmatic Strengths" (e.g., Pydantic schema enforcement). The Chief Justice then synthesizes these conflicting arguments into a final verdict, ensuring the score reflects both the risks and the engineering merits.

Fan-In / Fan-Out Architecture

To achieve high-efficiency auditing, the system utilizes a **Fan-Out** pattern. After the repository is cloned and indexed, the graph triggers multiple independent "Detective" nodes (RepoInvestigator, DocAnalyst) and "Judge" nodes (Prosecutor, Defense) simultaneously. This parallel execution is then consolidated in a **Fan-In** stage at the Justice node, where all evidence is aggregated before the final synthesis.

Metacognition

The system implements **Metacognition** (thinking about thinking) within the Chief Justice and

Tech Lead nodes. These nodes do not just look at code; they evaluate the *quality of the other agents' arguments*. If a Judge provides a score without citing specific line numbers or commit hashes, the Synthesis engine identifies this as a "low-confidence" argument and adjusts the final weight accordingly.

---

3. Architectural Diagram (StateGraph)

The system follows a strict state-based flow:

1. **Entry**: clone_repo (Secure subprocess sandboxing)
2. **Analysis (Fan-Out)**: repo_investigator + doc_analyst
3. **Litigation**: prosecutor + defense + tech_lead
4. **Synthesis (Fan-In)**: chief_justice
5. **Exit**: justice (Serialization to .md and .pdf)

---

4. Self-Audit Results (Criterion Breakdown)

Criterion	Score	Key Finding
Git Forensics	3.0/5	Transitioned from "Big Bang" commits to atomic, security-focused history.
State Management	4.0/5	Robust LangGraph State utilizing Pydantic for schema enforcement.
Safe Tool Engineering	5.0/5	Eliminated os.system in favor of subprocess.run(shell=False) .
Theoretical Depth	4.0/5	RAG-lite successfully identifies complex synthesis indicators.

---

5. Reflection: The MinMax Feedback Loop

What the Peer Agent Caught (Simulated)

Early iterations of my agent were flagged for **"Process Risk"** because files were saved in the root directory. My peer's agent also identified a potential **Shell Injection** vulnerability where repository URLs were passed directly to the shell.

How I Updated My Agent

1. **Directory Compliance:** Updated main.py to automatically generate the audit/report_onself_generated/ hierarchy to match official standards.
2. **Model Diversification:** Switched to a mix of llama-3.1-8b-instant (for speed/cost) and llama-3.3-70b-versatile (for reasoning) to prevent "Unavailable" judge outputs.
3. **Security Hardening:** Refactored all shell calls to use strict list-based arguments in subprocess.run to prevent command hijacking.

---

6. Remediation Plan

- **Short Term:** Integrate a **Reasoning Agent** (Gemini 2.0) to perform deeper AST logic verification beyond simple existence checks.
- **Medium Term:** Implement **Dynamic Simulation** nodes that attempt to run the code in a Docker container to verify functional correctness.
- **Long Term:** Expand the **Synthesis Engine** to generate visual SVG graphs of the audited project's architecture automatically.