FDE Challenge Week 2: The Automaton Auditor Interim Report

Zemzem Hibet

February 25, 2026

Executive Overview

This interim submission presents the foundational architecture for the **Automaton Auditor**, a Deep LangGraph-based multi-agent system designed to perform structured forensic analysis of a GitHub repository and its accompanying PDF report. The system is not designed as a simple grading script but as scalable governance infrastructure capable of evaluating autonomous code generation systems.

The core objective of this architecture is to shift from generation to governance. Instead of relying on a monolithic language model to interpret and evaluate submissions, the system decomposes evaluation into structured, layered responsibilities. The architecture implements a Digital Courtroom model in which Detectives collect objective evidence, Judges (to be implemented in the final phase) interpret that evidence through distinct philosophical lenses, and a Chief Justice node synthesizes the final verdict using deterministic rules. The interim submission focuses specifically on typed state rigor, safe tool engineering, and parallel orchestration.

Architectural Philosophy

The Automaton Auditor is designed under the principle that facts must precede interpretation. Many AI systems fail because they merge evidence collection with evaluation, leading to hallucination and unverifiable claims. To prevent this, the architecture enforces a strict separation between forensic evidence collection and judicial reasoning.

The Detective layer is responsible only for collecting verifiable facts from artifacts such as Git history and PDF documentation. No scoring or subjective interpretation occurs at this stage. This separation establishes a structured reasoning pipeline in which downstream evaluation is grounded exclusively in collected evidence.

State Management Rigor

Robust state management is foundational to multi-agent orchestration. The system uses Pydantic BaseModel classes to define structured outputs such as `Evidence` and `JudicialOpinion`. The global graph state is defined using a TypedDict schema that explicitly declares all state fields.

Plain Python dictionaries were intentionally avoided because they introduce schema drift, silent corruption, and unpredictable behavior during parallel execution. By enforcing typed state definitions, the system guarantees structural consistency and validation of outputs at every stage of execution.

Additionally, reducers are implemented using `operator.add` and `operator.ior` within Annotated type hints. These reducers ensure that parallel agents append or merge state safely rather than overwriting one another's outputs. This design decision prepares the architecture for true fan-out/fan-in swarm orchestration in later phases.

Safe Tool Engineering

Security and isolation are critical when executing system-level operations such as cloning external repositories. The repository cloning mechanism uses `tempfile.TemporaryDirectory()` to sandbox all cloned content. Git operations are executed using `subprocess.run()` with explicit error handling and captured output.

Raw `os.system()` calls were deliberately avoided due to the risk of shell injection and uncontrolled execution environments. By isolating repository cloning in temporary directories and checking return codes, the system prevents both security vulnerabilities and silent failures.

Git Forensic Analysis

The interim implementation extracts commit history using the command:

```
git log --oneline --reverse
```

This allows the system to analyze development progression, count commits, and capture timestamps. The intention is to detect iterative engineering practices versus monolithic code uploads. In the final implementation, this data will be classified into progression patterns such as atomic development, clustered commits, or bulk uploads, forming part of the forensic evaluation layer.

PDF Document Analysis

The system includes structured PDF ingestion using the `pypdf` library. The current implementation extracts full document text and performs keyword-based scanning for architectural concepts such as “Dialectical Synthesis,” “Fan-In / Fan-Out,” and “Metacognition.”

While this interim version uses direct text search, the final phase will implement chunked retrieval and targeted querying to validate conceptual depth more rigorously. The objective is to distinguish between genuine architectural explanation and superficial keyword usage.

LangGraph Orchestration Architecture

The orchestration layer is implemented using LangGraph’s `StateGraph`. The interim architecture establishes a parallel fan-out structure at the Detective layer. From the START node, execution branches into `RepoInvestigator` and `DocAnalyst` nodes concurrently.

This parallel design avoids the bottleneck and fragility of linear pipelines. Instead of chaining agents sequentially, evidence collection occurs independently and concurrently. This prepares the system for the future addition of an Evidence Aggregator node and a parallel Judicial layer.

The architecture intentionally avoids a purely linear flow such as `RepoInvestigator → DocAnalyst → Judge`. Linear execution undermines scalability and reduces the system to a simple script rather than a swarm-based governance engine.

Planned Phase 2 Implementation

The interim submission intentionally excludes the Judicial and Synthesis layers. In the final phase, three judicial personas will be introduced: Prosecutor, Defense, and Tech Lead. Each will analyze identical evidence through distinct philosophical lenses and return structured `JudicialOpinion` objects enforced through schema validation.

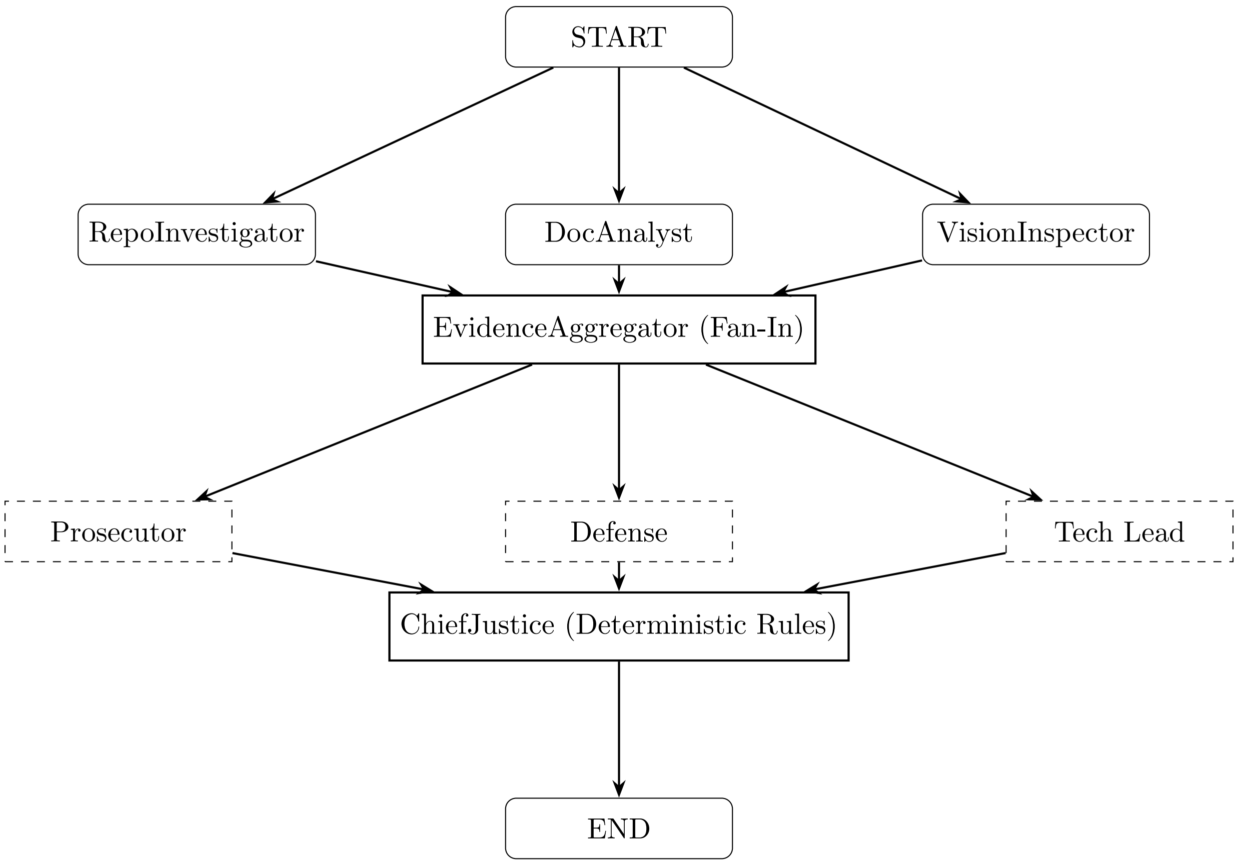
The Chief Justice node will implement deterministic conflict resolution rules in Python. These rules will include security override logic, fact supremacy enforcement, and score variance re-evaluation. The final output will be serialized as a structured Markdown audit report.

Planned Full StateGraph Architecture

The complete architecture will follow a dual fan-out/fan-in design. Detectives will execute in parallel and synchronize through an Evidence Aggregator node. Judges will then execute in parallel on the aggregated evidence. Finally, the Chief Justice node will synthesize a verdict and generate a structured report.

This architecture ensures dialectical reasoning, parallel scalability, and deterministic governance.

The following diagram illustrates the planned full swarm architecture using a dual fan-out and fan-in orchestration model. The interim submission implements the Detective fan-out portion, while the Judicial and Synthesis layers will be added in the final phase.



Reflection

The most significant insight from this interim phase is that building an evaluator requires deeper architectural rigor than building a generator. Evaluation systems must enforce structure, prevent hallucination, and ensure that every claim is traceable to evidence.

By separating evidence collection from interpretation, enforcing typed state, implementing parallel-safe reducers, and sandboxing tool execution, the Automaton Auditor establishes a production-grade foundation for AI governance infrastructure.

Conclusion

This interim submission successfully implements typed state definitions, parallel LangGraph orchestration, secure repository cloning, structured evidence extraction, and preliminary document analysis. The system is now structurally prepared for the addition of judicial personas and deterministic synthesis rules in the final phase.

The architecture prioritizes rigor, security, and scalability, laying the groundwork for a robust Automated Auditor Swarm capable of operating in AI-native enterprise environments.