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, structured parsing decisions, 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, source code, 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.

Structured Code Parsing Strategy (AST vs Regex Trade-off)

A critical architectural decision concerns how repository source code is analyzed. The system adopts Abstract Syntax Tree (AST) parsing rather than regular expression (regex) scanning for structural code inspection.

Why regex was avoided:

- Regex operates purely at the textual level and cannot reliably capture nested structures such as classes, decorators, async functions, or inheritance hierarchies.
- It is brittle against formatting variations and prone to false positives (e.g., matching commented code).
- It fails to guarantee syntactic validity, meaning malformed files may produce misleading matches.

Why AST parsing was selected:

- AST parsing provides syntactically validated structural representations of code.
- It enables reliable extraction of functions, classes, imports, decorators, and call graphs.
- It reduces hallucination risk by grounding analysis in compiler-level structure.

Trade-off Considerations:

- AST parsing is language-specific and requires structured error handling for malformed files.
- It introduces slightly higher computational cost compared to regex scanning.
- It may fail on incomplete or partially generated files.

Despite these trade-offs, AST parsing was selected because the Automaton Auditor prioritizes structural rigor and forensic reliability over speed or superficial pattern matching. Regex may still be used for lightweight heuristics, but never as the primary structural analysis mechanism.

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:

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

This allows analysis of development progression patterns and detection of monolithic uploads versus iterative engineering. Future classification logic will categorize commits into atomic, clustered, or bulk progression patterns as part of structured forensic scoring.

PDF Document Analysis

The system includes structured PDF ingestion using the `pypdf` library. The interim implementation performs keyword scanning for architectural concepts.

While this version uses direct text search, the final phase will implement chunk-based retrieval and targeted semantic querying to detect conceptual depth and reduce superficial keyword stuffing.

LangGraph Orchestration Architecture

The orchestration layer uses LangGraph's `StateGraph`. The interim architecture establishes a parallel fan-out structure at the Detective layer, branching from START into independent investigative nodes.

This avoids linear bottlenecks and prepares the system for swarm scalability. The design enforces independence of evidence streams prior to aggregation.

Planned Phase 2 Implementation with Risk Analysis

The Judicial and Synthesis layers remain to be implemented. Three judicial personas will analyze identical evidence and return structured `JudicialOpinion` outputs.

Anticipated Risks and Failure Modes:

- **Persona Convergence Risk:** Judges may produce overly similar opinions if prompts are insufficiently differentiated.
- **Score Variance Instability:** Large score divergence may indicate unclear evidence grounding.
- **Evidence Misinterpretation:** Judges may hallucinate facts not present in collected evidence.

- **Deterministic Rule Conflict:** Hard-coded override rules may unintentionally suppress legitimate dissent.
- **Aggregation Bias:** The Evidence Aggregator may unintentionally prioritize certain evidence types.

Mitigation Strategies:

- Strict schema validation of JudicialOpinion objects.
- Explicit evidence citation requirement within judicial outputs.
- Variance threshold detection and forced re-evaluation.
- Deterministic override rules limited to security and structural violations only.
- Logging and traceability for every decision path.

Including these safeguards ensures that the final governance layer remains transparent, auditable, and resistant to systemic bias.

Planned Full StateGraph Architecture

The final architecture explicitly models state transitions and error handling paths. Edges are labeled with the type of state being passed between nodes, and conditional edges manage failure or missing evidence scenarios.

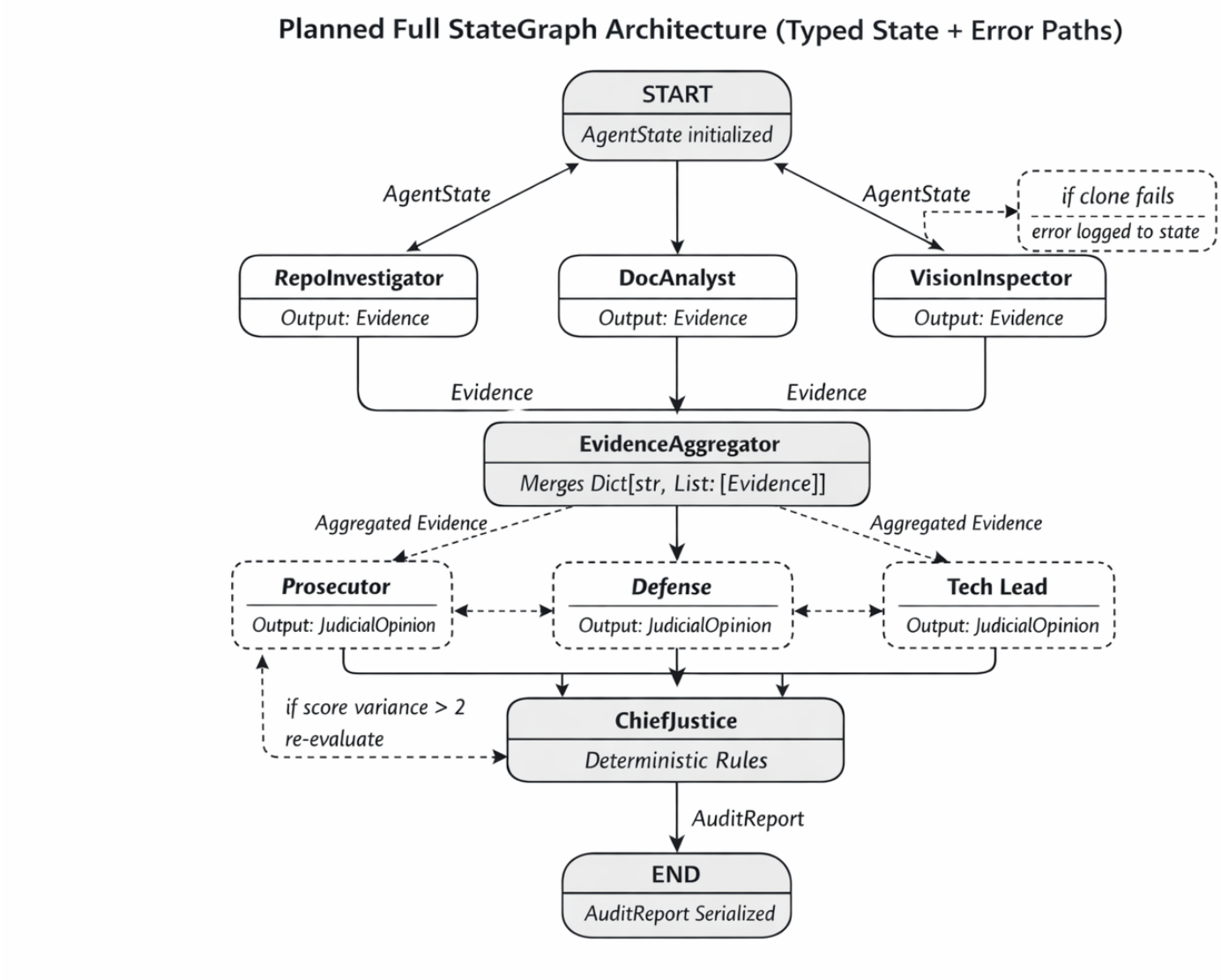

Figure 1: Planned Full StateGraph Architecture with Typed State and Error Paths.

Reflection

Building an evaluator requires greater architectural discipline than building a generator. Evaluation systems must enforce structure, prevent hallucination, anticipate failure modes, and guarantee traceability.

This interim phase strengthened not only implementation but also architectural foresight by incorporating parsing trade-offs and forward-looking risk analysis.

Conclusion

This interim submission strengthens architectural justification by incorporating structured trade-off analysis for code parsing and explicit risk modeling for future judicial layers.

The Automaton Auditor now reflects not only functional orchestration but principled governance design, positioning it as scalable AI-native evaluation infrastructure.