The Automaton Auditor Interim Report

Overview

The challenge is to build a system that can autonomously evaluate peer code submissions with the same depth and nuance a senior engineer would distinguishing between code that looks correct and code that is correct, between effort and competence, between understanding an architecture and cargo-culting its syntax. A single LLM prompt cannot do this reliably. The solution is a hierarchical multi-agent swarm modelled on a courtroom: Detectives collect objective evidence, Judges interpret it through opposing lenses, and a Chief Justice resolves the conflict through deterministic rules.

This document covers the core architectural decisions driving the design, the known gaps in the judicial and synthesis layers, and the planned StateGraph topology.

1. Architecture Decisions

1.1 Why Pydantic Over Plain Dicts

The fundamental question in any multi-agent system is: how does one agent's output become another agent's reliable input? Plain Python dicts are the obvious answer and the wrong one.

The problem is that dicts are implicit contracts. When a Detective node returns {"found": True, "location": "src/state.py"} and a Judge node expects to read evidence.file_path, the system fails at runtime with a KeyError, inside a parallel branch, with no indication of which node produced malformed data or why. The failure is silent until it isn't, and when it surfaces it is opaque.

Pydantic BaseModel turns this implicit contract into an explicit one. The Evidence model defines exactly what a Detective must return: *goal, found, content, location, rationale, confidence* with enforced types on every field. If a node tries to return something that doesn't conform, the error is raised immediately at the source, not silently downstream. This is the difference between a compile-time error and a production mystery.

The same principle applies to the Judicial layer. LLMs produce freeform text by default. A Judge returning "this looks like a solid 4" will crash any node that tries to access opinion.score. Binding the model with .with_structured_output(JudicialOpinion) forces output to conform to score: int, argument: str, cited_evidence: List[str], or the call retries. The schema is the contract. The contract is the architecture.

There is also a deeper reason specific to LangGraph's parallel execution model. When multiple nodes write to the same state field concurrently, without a declared merge strategy, the last writer wins silently

destroying the other nodes' work. LangGraph solves this through **reducers** declared in Annotated type hints, which only work with typed state fields:

```python
# When RepoInvestigator and DocAnalyst both write to evidences in parallel,
# operator.ior merges their dicts by key union rather than overwriting.
# {"repo": [...]} | {"doc": [...]} → {"repo": [...], "doc": [...]}
evidences: Annotated[Dict[str, List[Evidence]], operator.ior]

# When Prosecutor, Defense, and TechLead all write opinions in parallel,
# operator.add appends their lists rather than replacing the whole list.
# [op1] + [op2] + [op3] → [op1, op2, op3]
opinions: Annotated[List[JudicialOpinion], operator.add]
```

Without operator.ior, the DocAnalyst's evidence overwrites the RepoInvestigator's, and the Chief Justice reasons from an incomplete picture. Without operator.add, the dialectical bench collapses to whichever judge finished last. These reducers are not implementation details they are what makes the parallel architecture semantically coherent rather than just visually parallel.

1.2 How AST Parsing Should Be Structured

The RepoInvestigator's job is forensic verification, not surface-level grep. The distinction matters enormously when the thing being verified is code structure.

Consider the simplest forensic task: confirm that a submitted repo uses Pydantic BaseModel for state management rather than plain dicts. A regex search for BaseModel returns True for:

- A comment: should use BaseModel here
- A string in a test: assert "BaseModel" in class_name
- An import that is never used: from pydantic import BaseModel with no subclass anywhere

All of these are false positives that a naive auditor would pass. Regex cannot distinguish a syntactic relationship from an incidental string occurrence. This is exactly the failure mode the rubric calls "Hallucination" , an agent that reports code exists when it only found the word.

AST parsing solves this by operating on the syntax tree rather than the raw text. Python's built-in ast module parses source code into a structured tree of nodes. The investigator walks this tree looking for semantically meaningful patterns:

- **Pydantic usage:** ClassDef nodes where the bases list contains Name(id="BaseModel") this is inheritance at the syntax level, not the word in a comment
- **Reducer usage:** Subscript nodes whose content is Annotated[..., operator.ior] or Annotated[..., operator.add] confirming that the reducer is actually wired into the type annotation, not just imported

- **Graph topology:** Call nodes where func.attr == "add_edge" with two positional arguments extracting the actual (from_node, to_node) pairs that define the graph's wiring
- **Fan-out detection:** if the same source node appears in multiple add_edge calls, the graph genuinely branches a structural property that is completely invisible to text search

The investigator's output is not a boolean. It is a structured Evidence object with location (the file and line), content (the specific code snippet confirming the finding), confidence (0.0–1.0), and rationale (the reasoning behind the confidence score). This is what separates forensic evidence from an opinion every claim is tied to a specific, citable artifact.

1.3 Sandboxing Strategy

Cloning and analyzing unknown peer code is the riskiest operation in the system. The threat model is not necessarily malicious it is simply that peer repos are unpredictable. A setup.py might attempt to install packages. A post-checkout git hook might write to arbitrary paths. A broken import at the top level might crash the auditor process if the cloned code is ever imported rather than only read.

The design principle is: the auditor touches cloned code, but cloned code never touches the auditor.

- **Isolation via tempfile.mkdtemp() -** Every clone operation targets a fresh directory outside the project tree, managed by the OS. Cloned code never lands in src/, never appears alongside live auditor files, and the directory is cleaned up after analysis regardless of whether analysis succeeds or fails.
- **No shell via subprocess.run() -** The alternative os.system("git clone " + repo_url) passes the full command string to a shell interpreter. Any user-controlled content in repo_url is therefore shell code. A URL like https://github.com/user/repo; rm -rf / would execute. subprocess.run(["git", "clone", repo_url, tmp_dir]) passes each element as a literal token with no shell interpolation. No injection surface.
- **Timeout enforcement -** A repo requiring interactive authentication hangs indefinitely if unchecked. A 60-second timeout kills the process and surfaces the failure as a low-confidence Evidence object the graph continues, the failure is recorded, the pipeline does not block.
- **Read-only analysis -** After cloning, the investigator reads files using Path.read_text() and parses them with ast.parse(). No cloned code is ever executed, imported, or evaluated. The AST analysis is purely structural it reasons about what the code says, not what it does.

2. Known Gaps & Plan for the Judicial Layer and Synthesis Engine

2.1 The Judicial Layer Design

The judicial layer is the hardest part of the system to get right, and the easiest to fake. The obvious failure mode of three agents with slightly different names but nearly identical prompts producing nearly identical scores is what the rubric calls "Persona Collusion," and it defeats the entire purpose of dialectical reasoning.

The design goal is genuine adversarial tension: the Prosecutor and Defense should regularly disagree, and their disagreement should be substantive, not cosmetic. This requires the three personas to be built from fundamentally different axioms, not just different tones.

The Prosecutor operates from the axiom "Trust No One. Assume Vibe Coding." Its job is to find what's broken, missing, or deceptively shallow. It reads the rubric as a list of charges and looks for evidence to sustain each one. A graph that is visually parallel but has no state reducers is convicted of "Orchestration Fraud." A Judge node that returns freeform text is charged with "Hallucination Liability." The Prosecutor's default score is 1; it has to be argued up from there.

The Defense Attorney operates from the axiom "Reward Effort and Intent." Its job is to find what works, what shows genuine understanding, and what deserves credit even if the execution is imperfect. It reads the git history as a narrative: a repo with ten commits showing progressive refinement tells a story of engineering process that a bulk-upload repo cannot. A system that fails to compile but contains sophisticated AST logic gets credit for "deep code comprehension that tripped on framework syntax." The Defense's default score is 4; it has to be argued down.

The Tech Lead operates from the axiom "Does it actually work? Is it maintainable?" It ignores the vibe of the Prosecutor and the generosity of the Defense and asks the pragmatic question: would this pass a real code review? It is the tiebreaker by design when the Prosecutor says 1 and the Defense says 5, the Tech Lead's assessment of actual architectural soundness determines the outcome.

The key enforcement mechanism is .with_structured_output(JudicialOpinion) each judge must return a structured score, argument, and cited_evidence list. The cited evidence must reference specific Evidence.location values from the Detective layer, not general assertions. This prevents a Judge from fabricating claims the Detectives did not find.

2.2 The Chief Justice Synthesis Design

The synthesis layer is where the system either demonstrates genuine intelligence or degrades into a sophisticated averaging function. The design choice is deliberate: **the scoring decision must be deterministic Python logic, not an LLM prompt.**

The reason is constitutional integrity. If the Chief Justice is just another LLM call that reads three scores and decides a final one, there is no guarantee the security override will be applied when it should be, no guarantee the dissent will be explained when scores diverge, and no guarantee that facts from the Detective layer will outweigh creative arguments from the Defense. The Chief Justice needs to be a function that cannot be argued out of its rules.

The three constitutional rules are:

- **Rule of Security -** If the Prosecutor cites a confirmed security violation os.system() with unsanitized input, shell injection in the clone logic, credentials committed to the repo the

final_score is capped at 3, regardless of how strongly the Defense argues for effort and intent. Security is non-negotiable and deterministically overrides the judicial consensus.

- **Rule of Evidence (Fact Supremacy).** If the Defense claims an artifact exists "the student clearly understands state synchronization" but the RepoInvestigator found found=False for that artifact, the Defense is overruled. Forensic facts always outrank judicial opinion. An LLM cannot argue a file into existence.
- **Rule of Functionality.** On the graph_orchestration criterion specifically, if the Tech Lead confirms the architecture is modular and actually executable, this carries maximum weight. Architectural soundness as assessed by the pragmatic lens is the deciding factor for structural questions.

After these rules lock the score, an LLM generates the narrative the dissent summary, the remediation language, the executive summary prose. This is the right division of labor: deterministic logic for decisions, LLM for communication.

Variance re-evaluation handles the case where scores diverge by more than 2 (e.g., Prosecutor says 1, Defense says 5). Rather than averaging or defaulting, the Chief Justice triggers a second pass: it re-examines the specific cited_evidence from each Judge, checks whether the cited locations are verified in the Detective evidence, and applies the constitutional rules to the re-examined evidence before rendering a final score. The dissent summary is mandatory for any criterion where this occurs.3. Planned StateGraph Flow

3.1 Full Target Architecture

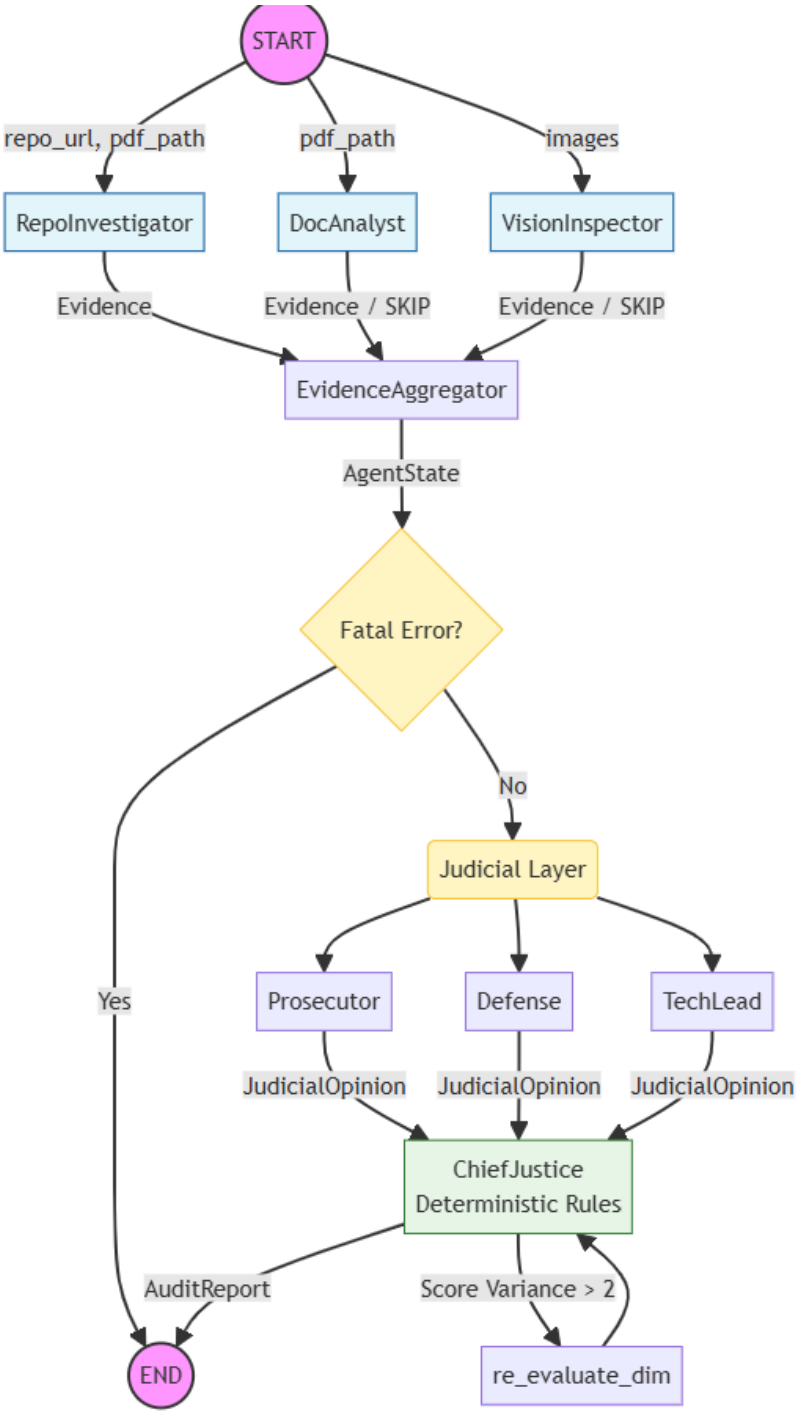

Fig: State graph flow

3.2 Why the Graph Is Structured This Way

Two fan-out / fan-in cycles, not one - The Detective fan-out runs first so that evidence is fully assembled before any judgment begins. The Judicial fan-out runs second so that all three judges analyze the same

complete evidence bundle. The two synchronization points EvidenceAggregator (fan-in after Detectives) and the implicit accumulation before ChiefJusticeNode (fan-in after Judges) enforce this ordering. Without the first sync, a Judge might reason from partial evidence. Without the second sync, the Chief Justice might receive only two of three opinions.

The ContextBuilder node at the start - Rather than hardcoding forensic instructions inside each detective, the ContextBuilder loads rubric.json at runtime and distributes instructions by target_artifact key. This means the RepoInvestigator only receives instructions relevant to the GitHub repo, the DocAnalyst only receives instructions relevant to the PDF, and the VisionInspector only receives instructions relevant to images. Updating the rubric requires editing one JSON file, not touching any node code.

EvidenceAggregator as a pure Python node - The aggregator does not call an LLM. It is a synchronization checkpoint that validates evidence completeness and logs any missing keys before the Judicial layer begins. This is important because the vision evidence is optional if the VisionInspector fails or is not executed, the aggregator flags it as low-confidence rather than crashing the pipeline.

ChiefJusticeNode with locked scores. The deliberate separation between the Python scoring logic and the LLM narrative generation ensures that the constitutional rules cannot be argued away. The LLM's job is to explain the ruling in natural language, not to make the ruling.

3.3 Graph Orchestration (Fan-Out / Fan-In)

The current implementation includes:
Detective Parallelism

- RepoInvestigator
- DocAnalyst
- VisionInspector (optional execution)

These run in **parallel fan-out** from START.
Then synchronize into:

- EvidenceAggregator (fan-in barrier)

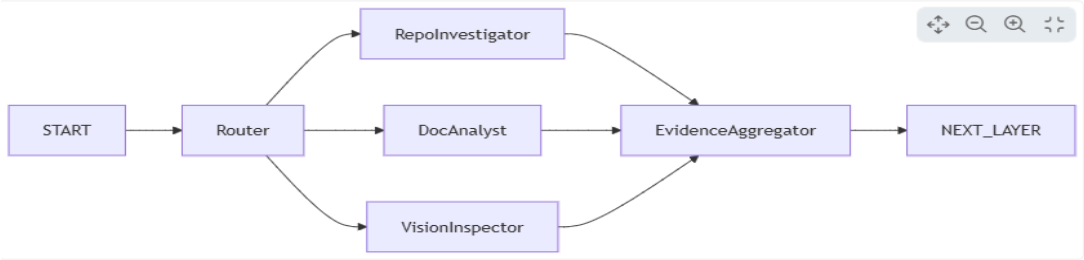

Fig.Graph Orchestration (Fan-Out / Fan-In)

4. Known Gaps

The Judicial Layer

The judicial layer is fully designed but not yet wired. The three personas Prosecutor, Defense, Tech Lead have distinct philosophical axioms (documented in Section 2.1) and their system prompts are drafted to share less than 30% text, avoiding Persona Collusion. The JudicialOpinion Pydantic schema is defined. What remains is building the three node functions in src/nodes/judges.py with .with_structured_output() enforcement and retry logic, then wiring the judicial fan-out from EvidenceAggregator into the graph.

The retry logic is a critical detail: if a Judge returns freeform text instead of a valid JudicialOpinion, the node retries up to two times before emitting an error opinion with score=1 and a cited_evidence entry flagging the parse failure. The Chief Justice then has the evidence it needs to apply the "Hallucination Liability" charge if appropriate.

The Chief Justice Synthesis Engine

The ChiefJusticeNode is designed (Section 2.2) but not yet implemented. The deterministic rule functions security override, fact supremacy, functionality weight, variance re-evaluation need to be written as pure Python before any LLM narrative synthesis is layered on top. The Markdown serialization of AuditReport into the structured output format (Executive Summary → Criterion Breakdown → Remediation Plan) is also pending.

rubric.json Externalization

The rubric is currently embedded in the task specification. Externalizing it into a rubric.json at the repo root and building the ContextBuilder node to load and distribute it is the prerequisite for the entire judicial layer judges need live rubric dimensions injected into their system prompts at runtime rather than hardcoded instructions. This will be the first thing built, as it unblocks everything else.

VisionInspector Execution

The VisionInspector's image extraction logic is in place, but the multimodal LLM classification call which determines whether a diagram shows genuine parallel fan-out or a misleading linear flowchart has not been connected. This is treated as optional for the interim but required for the final submission.