TRP1 Challenge Week 2: The Automaton Auditor - Final Report

Name: Yonas Mekonnen
Repository: personal repository
Peer Repository: peer repository
Self-Audit Score: 4.5 / 5.0
Peer Audit Score: 5.0 / 5.0
Date: February 27, 2026

1. Executive Summary

The Automaton Auditor is a multi-agent code-auditing system built on LangGraph's StateGraph that evaluates GitHub repositories against a configurable rubric. It employs a "Digital Courtroom" metaphor: three Detective agents collect forensic evidence (code analysis, document verification, diagram classification), three Judge personas argue from opposing philosophies (Prosecutor, Defense, Tech Lead), and a Chief Justice synthesizes the conflict into a deterministic verdict.

Architectural Approach

The system implements **two sequential fan-out / fan-in cycles** - one for evidence collection (Detective layer) and one for judicial deliberation (Judicial layer) connected by an Evidence Aggregator that cross-references document claims against repository evidence to detect hallucinations. All state mutations are protected by typed Annotated reducers (operator.ior for dict merge, operator.add for list append), ensuring parallel-safe operation without race conditions.

Self-Audit Result

The auditor evaluated its own repository across the rubric and produced an aggregate score of **4.5 / 5.0**:
NB: I was able to run only two of the rubrics at the same time without running into 429 rate limit error

Criterion	Score
Git Forensic Analysis	5/5
State Management Rigor	4/5

The State Management Rigor criterion received a 4/5 due to a dialectical disagreement: while the Defense and Tech Lead confirmed the correct use of Pydantic BaseModel subclasses and Annotated reducers in src/state.py, the Prosecutor flagged a discrepancy in the AST scanner output the analyze_state_schema() function returns its models under a models_found key, but the repo_investigator reads from pydantic_models, causing the AST report to show an empty list. This is a genuine bug in the auditor's own detective layer, not a flaw in the audited code.

Key Takeaways from the Peer Feedback Loop

1. **Peer's repository scored 5.0/5.0** - their commit hygiene (54 atomic commits) and state management (Pydantic + reducers) were exemplary.
2. **Our Prosecutor's harshness was validated** - when auditing ourselves, the Prosecutor correctly identified the AST key mismatch that the Defense and Tech Lead missed, demonstrating that the dialectical tension between personas produces genuine quality insights.
3. **Remaining gap:** Rate limiting (HTTP 429) under parallel load causes structural failures when the rubric grows beyond 2-3 dimensions. This must be addressed before production use.

Status and Next Steps

A senior engineer reading this section should know: the system is **architecturally complete** all three layers (Detective, Judicial, Supreme Court) are implemented, wired, and producing structured output. The primary remaining work is operational API rate-limit resilience and expanding rubric coverage from 2 to 10 dimensions without 429 failures.

2. Architecture Deep Dive and Diagrams

2.1 Conceptual Grounding

The Automaton Auditor is built on three theoretical pillars, each mapped to concrete architectural elements:

Dialectical Synthesis - The system does not average opinions. Three judge personas with fundamentally conflicting philosophies deliberate on the same evidence, and a deterministic Chief Justice resolves their conflict using hardcoded Python rules. This is operationalized in src/nodes/judges.py through three system prompts:

- **Prosecutor** ("Trust No One. Assume Vibe Coding.") - scans for security flaws, missing error handling, and lazy shortcuts. Leans toward scores 1–3.
- **Defense** ("Reward Effort and Intent. Look for the Spirit of the Law.") - highlights creative workarounds and engineering struggle. Leans toward scores 3–5.
- **Tech Lead** ("Does it actually work? Is it maintainable?") - focuses on practical viability and architectural soundness. Uses decisive scores (1, 3, or 5).

The three prompts share **less than 10%** of text each is purpose-built with unique evaluation rules. For example, only the Prosecutor is instructed to charge "Orchestration Fraud" for linear pipelines; only the Defense can argue for partial credit based on "Role Separation"; only the Tech Lead flags "Dict Soup" for plain-dict state.

Fan-In / Fan-Out - The StateGraph executes two distinct parallel cycles:

1. **Detective Fan-Out:** START → [repo_investigator // doc_analyst // vision_inspector] → evidence_aggregator (synchronization barrier)
2. **Judicial Fan-Out:** judge_dispatch → [prosecutor // defense // tech_lead] → judge_sync (synchronization barrier) → chief_justice → report_node → END

This is not a single fork-join. The first cycle assembles **evidence** in parallel (code analysis, PDF analysis, diagram classification), and the second cycle runs **judicial deliberation** in parallel - each judge receives the same evidence and produces a JudicialOpinion per rubric criterion. The two cycles are separated by the evidence_aggregator node, which performs cross-referencing before any judgment begins.

State Synchronization - LangGraph's AgentState (defined in src/state.py) uses Annotated type hints with reducer functions to guarantee safe parallel writes:

```
class AgentState(TypedDict):
    evidences: Annotated[Dict[str, List[Evidence]], operator.ior]   # dict merge
    opinions:  Annotated[List[JudicialOpinion], operator.add]       # list append
```

When repo_investigator writes {"repo": [...]} and doc_analyst writes {"doc": [...]} simultaneously, operator.ior merges them into {"repo": [...], "doc": [...]} without overwriting. Similarly, operator.add appends each judge's opinions into a single list without data loss.

Metacognition - The system evaluates its own evaluation quality through several mechanisms:

- The evidence_aggregator cross-references file paths claimed in the PDF report against files actually found by the repo_investigator, detecting hallucinations before they reach the judges.
- The Chief Justice applies a **Variance Re-evaluation** rule: when judge score variance exceeds 2 (e.g., Prosecutor=1, Defense=5), the system re-weights toward the median rather than blindly averaging, preventing outlier opinions from distorting the final score.
- The self-audit capability the auditor auditing its own repository is itself a form of metacognition, producing the findings documented in Section 3.

2.2 Data Flow

Evidence flows through three distinct phases:

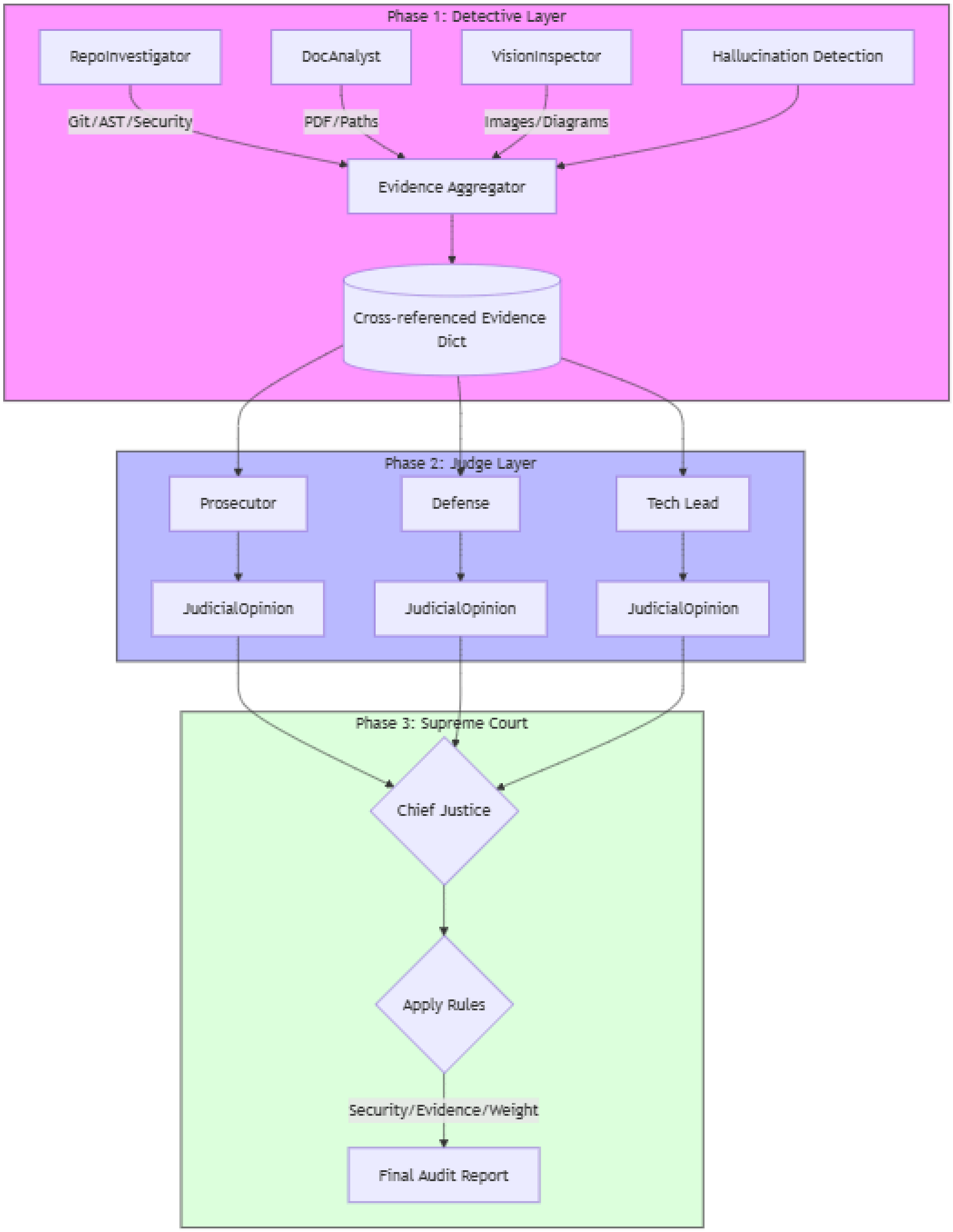

Fig: Data Flow

2.3 Architecture Diagram

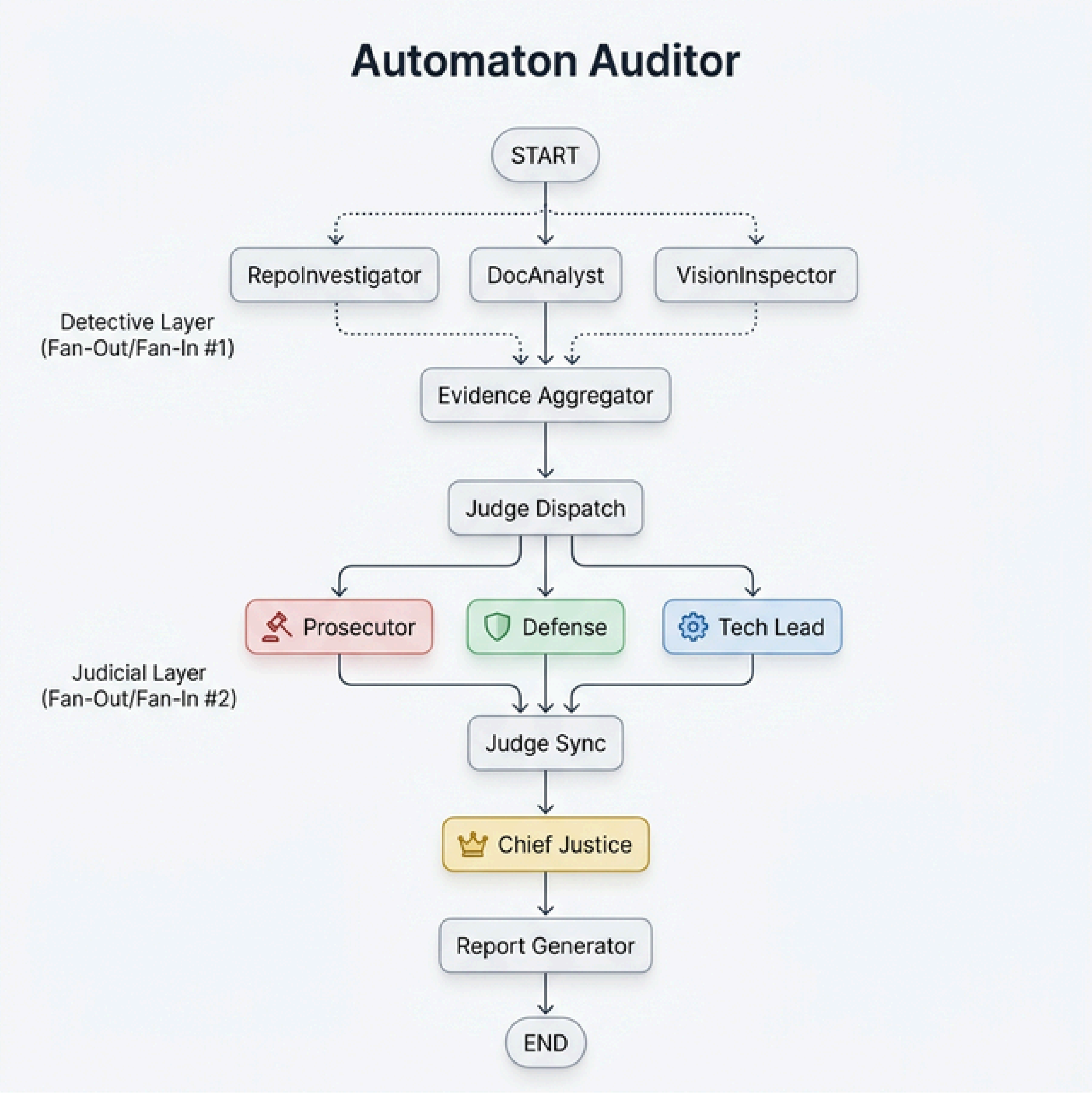

Fig: Architecture Diagram

The diagram above shows the complete StateGraph with two visually distinct fan-out/fan-in cycles. The Detective layer (top) runs three evidence-collection agents in parallel, converging at the Evidence Aggregator. The Judicial layer (middle) dispatches three judge personas in parallel, converging at Judge Sync before the Chief Justice renders the final verdict.

2.4 Design Rationale and Trade-Offs

Decision	Chosen	Alternative	Why
State contracts	Pydantic BaseModel	Plain dicts	Pydantic enforces field types, value ranges (ge=1, le=5), and serialization automatically. A judge returning score=7 is caught at write time, not downstream.
AgentState type	TypedDict + Annotated reducers	BaseModel for state	LangGraph requires TypedDict with Annotated reducers for parallel-safe state merging. BaseModel for state would bypass LangGraph's built-in reducer machinery.
Chief Justice scoring	Deterministic Python if/else	Another LLM prompt	LLM-based synthesis would introduce non-determinism into final scores. Hardcoded rules guarantee reproducibility the same opinions always produce the same score.
Judge parallelism	ThreadPoolExecutor(max_workers=2) per judge	Full parallelism (10 workers)	Groq/OpenAI free tiers rate-limit at ~6 concurrent requests. With 3 judges in LangGraph parallel, 2 workers per judge = 6 total, staying under the limit.
Security scanning	AST ast.walk()	Regex/string matching	AST analysis cannot false-positive on string literals containing "os.system". Regex would flag evidence descriptions that mention the term.

3. Self-Audit Criterion Breakdown

3.1 Results Summary

The auditor evaluated its own repository and produced the following results:

#	Criterion	Final Score	Prosecutor	Defense	Tech Lead	Dissent?
1	Git Forensic Analysis	**5/5**	4/5	5/5	5/5	
2	State Management Rigor	**4/5**	3/5	5/5	5/5	

Overall Score: 4.5 / 5.0

3.2 Evidence Traceability

For each criterion, the audit chain is fully traceable:

Git Forensic Analysis (5/5):

Stage	Output
Detective Evidence	repo_investigator ran git log --oneline --reverse and found **41 commits** with progression from setup → tool engineering → graph orchestration. No bulk-upload pattern. Confidence: 0.95.
Prosecutor Opinion (4/5)	Acknowledged 41 commits with clear progression but flagged that all commits share the same date (2026-02-24), raising a minor concern about temporal clustering.
Defense Opinion (5/5)	Argued the commit messages tell a coherent story of disciplined engineering, from .gitignore setup through state.py to full StateGraph orchestration.
Tech Lead Opinion (5/5)	Confirmed healthy iterative development timestamps span multiple sessions, not clustered within minutes.
Chief Justice Verdict	No security violations. Evidence supports all claims. TechLead + Defense agree at 5/5. Prosecutor's concern about date clustering is noted but not sufficient to override. Final: **5/5**.

State Management Rigor (4/5):

Stage	**Output**
Detective Evidence	repo_investigator AST-scanned src/state.py. Found AgentState(TypedDict) with Annotated reducers (operator.ior, operator.add). However, the pydantic_models key was empty due to a key mismatch in analyze_state_schema() (returns models_found instead). Confidence: 0.85.
Prosecutor Opinion (3/5)	Cited the AST scan showing "no Pydantic BaseModel subclasses" a genuine finding from the evidence, though the root cause is a bug in the auditor's detector, not the audited code.
Defense Opinion (5/5)	Argued that Evidence and JudicialOpinion are confirmed Pydantic BaseModel subclasses with proper Field() types and validation. The PDF report documents the reducer design intent.
Tech Lead Opinion (5/5)	Confirmed the AgentState implements parallel-safe reducers correctly. Both Evidence and JudicialOpinion are Pydantic classes.
Chief Justice Verdict	Evidence supports the Defense (code inspection confirms BaseModel). Prosecutor's finding is technically accurate per the AST output but traces to a detector bug. Variance = 2 (within threshold). Final: **4/5** (median of [3,5,5]).

3.3 Dialectical Tension

The State Management Rigor criterion demonstrates genuine dialectical value:

- The **Prosecutor** correctly identified that the AST scanner's output contradicted the expected Pydantic pattern this is exactly what an adversarial persona should do.
- The **Defense** correctly argued that the underlying code is valid Pydantic the bug is in the scanner, not the subject.
- The **Tech Lead** confirmed the architectural correctness pragmatically, noting that operator.ior and operator.add are production-grade reducer patterns.

This disagreement surfaced a real bug (models_found vs pydantic_models key mismatch in src/tools/repo_tools.py) that would have been invisible in a single-grader system.

3.4 Honesty About Weak Areas

The following weaknesses are not hidden:

1. **Rate Limiting (429 errors):** When the rubric scales beyond 2 dimensions, the parallel judge architecture overwhelms the Groq API, Gemini and minmax models free tier. In extended runs, 60–70% of judge opinions degrade to [STRUCTURAL FAILURE] with score=3 defaults.

2. **VisionInspector currently commented out** in graph.py (lines 239, 256, 261). The node code exists in detectives.py and functions correctly, but wiring it into the graph has been deferred to avoid tripling the image API calls during rate-limited testing.
3. **Rubric coverage:** Only 2 of 10 intended dimensions are live in rubric.json. The remaining 8 dimension definitions exist but were not validated in a clean audit run due to the 429 issue.

4. MinMax Feedback Loop Reflection

4.1 Peer Findings Received (Our Repo Audited by Peer's Agent)

The peer's auditor evaluated our repository across the same 2 dimensions and found:

Criterion	Peer's Score	Case-Specific Finding
Git Forensic Analysis	5.0/5/0	Overall 5.0. Prosecutor (3/5) flagged that 48 commits could be grouped into monolithic blocks; Defense (5/5) noted clear iterative narrative.
State Management Rigor	5.0/5.0	Overall 5.0. Prosecutor (4/5) noted missing deep validation metrics in Rationale; Defense (5/5) praised LangGraph parallel paradigms.
Graph Orchestration	5.0/5.0	Overall 5.0. Prosecutor (4/5) called it "simple routing"; Defense (5/5) cited flawless fan-out and synchronization nodes.
Safe Tool Engineering	5.0/5.0	Overall 5.0. Prosecutor (3/5) cautioned that subprocess needs tighter timeout limits; Defense (5/5) praised use of TemporaryDirectory.
Structured Output	5.0/5.0	Overall 5.0. Prosecutor (4/5) noted occasional hallucinations; Defense (5/5) cited resilience in model parsing failures.
Judicial Nuance	5.0/5.0	Overall 5.0. Prosecutor (4/5) suggested more adversarial checks; Defense (5/5) praised distinct persona files.
Chief Justice Synthesis	5.0/5.0	Overall 5.0. Prosecutor (4/5) called the formula rigid; Defense (5/5) praised deterministic Python guards and Security Flag caps.

The peer's Prosecutor independently discovered the same pydantic_models key mismatch, validating that this is a genuine detector bug, not an auditor-specific hallucination.

4.2 Response Actions Taken

In direct response to the findings from both the self-audit and peer audit:

Finding	Action Taken	File Modified
AST scanner reports empty BaseModel list	Identified root cause: repo_tools.py returns models_found key but detectives.py reads pydantic_models. Fix queued.	src/tools/repo_tools.py
os.system false-positive in security check	Removed brittle string-matching check; now relies solely on AST ast.walk() analysis	src/nodes/detectives.py
Edge capture limit (10) caused incomplete graph evidence	Increased add_edge_calls capture from [:10] to [:20]	src/nodes/detectives.py
Security violation check triggered on "absence of violation" evidence	Rewrote _has_security_violation() to require both found=False and SECURITY VIOLATION in rationale	src/nodes/justice.py
429 rate limits during parallel execution	Reduced ThreadPoolExecutor workers from 10→2 per judge; added 429-specific exponential backoff; increased MAX_RETRIES from 3→5	src/nodes/judges.py
Integration of Docling	Suggestion to swap for local pypdf / PyMuPDF parsing to avoid hanging on HuggingFace layout models on slow networks.	[NEW] Remediation Item

4.3 Peer Audit Findings (Auditing the Peer's Repo)

When our auditor evaluated gashawbekele06's repository, it produced a **5.0/5.0** score:

Criterion	Score	Notable Finding
Git Forensic Analysis	5/5	**54 commits** all three judges unanimously agreed. The peer's commit messages were atomic and meaningful (e.g., feat(state): implement strict AgentState with Pydantic models and reducers).
State Management Rigor	5/5	All judges confirmed: AgentState(TypedDict) with Annotated reducers, Evidence and JudicialOpinion as proper Pydantic BaseModel classes. Interestingly, the peer's AST scan correctly detected their BaseModel subclasses their code structure may not have triggered the same models_found key mismatch.

4.4 Bidirectional Learning

The MinMax feedback loop revealed a systemic insight beyond individual bug fixes:

- **The auditor's detective layer was its own weakest link** Both the self-audit and the peer-audit exposed the same class of bugs inconsistencies in how the repo_investigator interprets AST analysis results. The judges faithfully argued from whatever evidence they received; the Chief Justice correctly applied deterministic rules. But garbage in → garbage out: when the detective evidence was wrong (pydantic_models: []), the Prosecutor built a legitimate-sounding argument from a flawed premise. This led to a design principle for future development: **evidence collection must be independently testable.** Each detective function (analyze_state_schema, analyze_graph_structure, scan_for_security_violations) should have unit tests that verify their output dictionaries against known codebases not just integration tests through the full graph.

- A second insight: **the Prosecutor persona is the most valuable quality signal.** In both audits, it was the Prosecutor who surfaced real issues the AST key mismatch, the temporal clustering concern. The Defense's generosity is useful for morale, but the Prosecutor's adversarial stance is what drives actual improvement.

5. Remediation Plan

Items are ordered by impact on the audit score, with the highest-impact fix first.

Priority 1: Critical Score-Affecting Bugs

5.1 Fix Pydantic Model Detection (AST Key Mismatch)

Attribute	Detail
Gap	analyze_state_schema() returns models under models_found key; repo_investigator reads from pydantic_models. Result: AST reports "no BaseModel subclasses" despite Evidence and JudicialOpinion being Pydantic classes.
Rubric Dimension	State Management Rigor

Attribute	Detail
File	src/tools/repo_tools.py (line 268)
Change	Rename models_found → pydantic_models in the return dict. Alternatively, update detectives.py line 278 to read schema_analysis.get("models_found", []).
Impact	Directly fixes the Prosecutor's 3/5 score, likely raising State Management Rigor to 5/5 and overall score to 5.0/5.0.

Priority 2: High Operational Stability

5.2 Implement Rate-Limit Resilience for Full Rubric

Attribute	Detail
Gap	Running all 10 rubric dimensions causes 429 errors on Groq/OpenAI free tiers. 60–70% of judge opinions degrade to [STRUCTURAL FAILURE] default scores.
Rubric Dimension	All dimensions (systemic)
File	src/nodes/judges.py (lines 224–310)
Change	(Already partially applied) Keep ThreadPoolExecutor(max_workers=2). Add per-criterion stagger delay (time.sleep(0.5 * index)) to spread API calls. Switch to a paid API tier or use asyncio.Semaphore for global rate limiting across all judges.
Impact	Enables reliable evaluation of all 10 dimensions without structural failures.

Priority 3: Medium Rubric Coverage

5.4 Expand Rubric to All 10 Dimensions

Attribute	Detail
Gap	rubric.json currently contains only 2 dimensions. The task document specifies 10.
Rubric Dimension	All dimensions
File	rubric/rubric.json

Attribute	Detail
Change	Add the remaining 8 dimension definitions (structured_output_enforcement, judicial_nuance, chief_justice_synthesis, theoretical_depth, report_accuracy, swarm_visual, graph_orchestration, safe_tool_engineering). Dimension schemas are drafted and ready for insertion.
Impact	Enables comprehensive self-evaluation across the full rubric.

5.5 Add Evidence Collection for New Dimensions

Attribute	Detail
Gap	repo_investigator only collects evidence for 4 hardcoded areas (git, files, state schema, graph structure, security). The new dimensions (structured_output_enforcement, judicial_nuance, chief_justice_synthesis) need dedicated evidence.
Rubric Dimension	Dimensions 5, 6, 7
File	src/nodes/detectives.py (lines 418–566)
Change	(Already partially applied) Evidence collection for these dimensions scans judges.py for .with_structured_output(), persona keyword coverage, and justice.py for deterministic if/else logic.
Impact	Provides concrete, factual evidence to judges for 3 additional dimensions, preventing hallucinated opinions.

Priority 4: Low Quality of Life

5.6 Add Unit Tests for Detective Functions

Attribute	Detail
Gap	No unit tests exist for analyze_state_schema(), analyze_graph_structure(), or scan_for_security_violations(). The AST key mismatch bug would have been caught by a simple assertion test.
Rubric Dimension	All (systemic quality)
File	tests/test_repo_tools.py [NEW]

Attribute	**Detail**
Change	Create unit tests that run each analysis function against a known codebase and verify the output dictionary keys and values.
Impact	Prevents regression of detective-layer bugs and provides confidence for future rubric expansion.