Automaton Auditor — Architecture & Self-Audit Final Report

---

1. Executive Summary

Scope and architectural approach: The Automaton Auditor is an automated quality-assurance swarm that audits GitHub repositories and PDF reports using a Digital Courtroom architecture. Detectives (RepoInvestigator, DocAnalyst, VisionInspector) collect forensic evidence in parallel; Judges (Prosecutor, Defense, Tech Lead) deliberate per rubric criterion in parallel; the Chief Justice synthesizes a verdict using hardcoded deterministic rules (no LLM averaging). State is typed with Pydantic and TypedDict; parallel writes use explicit reducers (`merge_evidences` for evidences, `merge_opinions` for opinions) in `src/state.py`. The graph implements fan-out from START to three Detectives, fan-in at EvidenceAggregator, a conditional edge to a degraded-report path when collection fails, then fan-out to three Judges and fan-in at judge_collector before Chief Justice.

Self-audit result: The self-audit (run against this repository with the points-based peer rubric) yields 72 / 100 points (4 criteria: Development Progress 35, Feedback Implementation 12, Proactive Communication 0, Agent Feedback Relevance 25). Aggregate score: 72/100.

Most impactful finding from the peer feedback loop: The peer's agent (when auditing this repo) scored Feedback Implementation as No Exchange (0 pts, excluded from total) when no peer feedback file was present at `audit/report_bypeer_received`, and flagged that Agent Feedback Relevance could reach a higher level with deeper analysis of "parallel versus linear structures and reducers" in auditee code. These findings drove concrete changes: the EvidenceAggregator now looks for peer feedback in both the repo under evaluation and the current project (cwd), and the judicial synthesis explicitly weights Tech Lead for architecture/graph criteria.

Top remaining gap: Proactive Communication is absent (0 pts)—no documented progress updates, blocker flags, or design discussions. The system does not collect or evaluate evidence for proactive communication; adding a rubric dimension or Detective evidence source for communication artifacts would require product/process change, not only code.

Primary remediation priority: Implement a prioritized remediation plan (see Section 5) starting with (1) strengthening Judge persona differentiation and parse-failure handling so every dimension receives three opinions and dissent is traceable, and (2) documenting synthesis rules in code with rubric label traceability. These improvements directly address the self-audit shortfalls (Agent Feedback Relevance depth, judicial robustness) and align with peer feedback.

---

2. Architecture Deep Dive and Diagrams

2.1 Conceptual Grounding: Dialectical Synthesis, Fan-In/Fan-Out, Metacognition

Dialectical Synthesis (Thesis–Antithesis–Synthesis) is implemented with concrete architectural elements:

- Three conflicting judge personas: Prosecutor ("Trust No One. Assume Vibe Coding."), Defense ("Reward Effort and Intent"), and Tech Lead ("Does it actually work? Is it maintainable?") are implemented in `src/nodes/judges.py` with distinct system prompts (`_PROSECUTOR_SYSTEM`, `_DEFENSE_SYSTEM`, `_TECH_LEAD_SYSTEM`). Each judge produces a `JudicialOpinion` (score 1–5, argument, cited_evidence) per rubric dimension via `.with_structured_output(JudicialOpinion)`.
- Deterministic resolution logic: The Chief Justice (`src/nodes/justice.py`) does not average scores. It applies hardcoded rules in `_resolve_final_score()`: security_override (evidence of `os.system` or unsanitized input caps score at 3), fact_supremacy (if evidence does not support claims and Defense > Prosecutor, final is reduced to max(Prosecutor, Tech Lead)), functionality_weight (for dimensions whose name contains "architecture" or "graph", Tech Lead score is used), and variance_re_evaluation / dissent_requirement (when variance across the three scores > 2, Tech Lead is used as final and a dissent_summary is set). Thus dialectical conflict is resolved by rules, not by LLM.

Fan-In / Fan-Out is tied to specific graph edges and synchronization nodes:

- Fan-out (Detectives): In `src/graph.py`, `build_audit_graph()` adds edges from START to `repo_investigator`, `doc_analyst`, and `vision_inspector` so all three run in parallel. Each returns partial `evidences` (reducer: `merge_evidences` in `src/state.py`).
- Fan-in (Detectives): All three Detective nodes have edges to `evidence_aggregator`. The EvidenceAggregator merges evidences, injects placeholders for dimensions with no evidence, and performs cross-reference for `report_accuracy` and feedback lookup for `feedback_implementation`. This node is the synchronization point before the judicial layer.
- Conditional edge: After EvidenceAggregator, `_route_after_aggregator(state)` checks `is_critical_failure(state)`; if true, the graph routes to `degraded_report` → END; otherwise to `judicial_entry`.
- Fan-out (Judges): From `judicial_entry`, edges go to `prosecutor`, `defense`, and `tech_lead` so all three Judges run in parallel. Each returns partial `opinions` (reducer: `merge_opinions` in `src/state.py`).
- Fan-in (Judges): All three Judges have edges to `judge_collector`, which is a pass-through; opinions are already merged via the reducer. Then a single edge goes to `chief_justice` (synthesis endpoint).

Metacognition is connected to the system's ability to evaluate evaluation quality:

- Rule of Evidence (fact_supremacy): Chief Justice checks whether evidence actually supports claims. If `_evidence_supports_claim(evidences)` is false and Defense scored higher than Prosecutor, the final score is reduced so that unsupported Defense optimism is overruled—i.e. the system "reflects" on whether the judges' opinions are grounded in evidence.
- Rule of Security (security_override): When evidence indicates a security issue (`os.system`, unsanitized input), the synthesis caps the score at 3 regardless of judge scores, so the system corrects for judges under-weighting security.
- Variance re-evaluation: When the three judges disagree strongly (variance > 2), the system explicitly records dissent and uses Tech Lead as tie-breaker, surfacing that the evaluation process itself had conflicting views and how resolution was applied.

Thus Dialectical Synthesis, Fan-In/Fan-Out, and Metacognition are explained with concrete connections to code and graph structure, not as buzzwords.

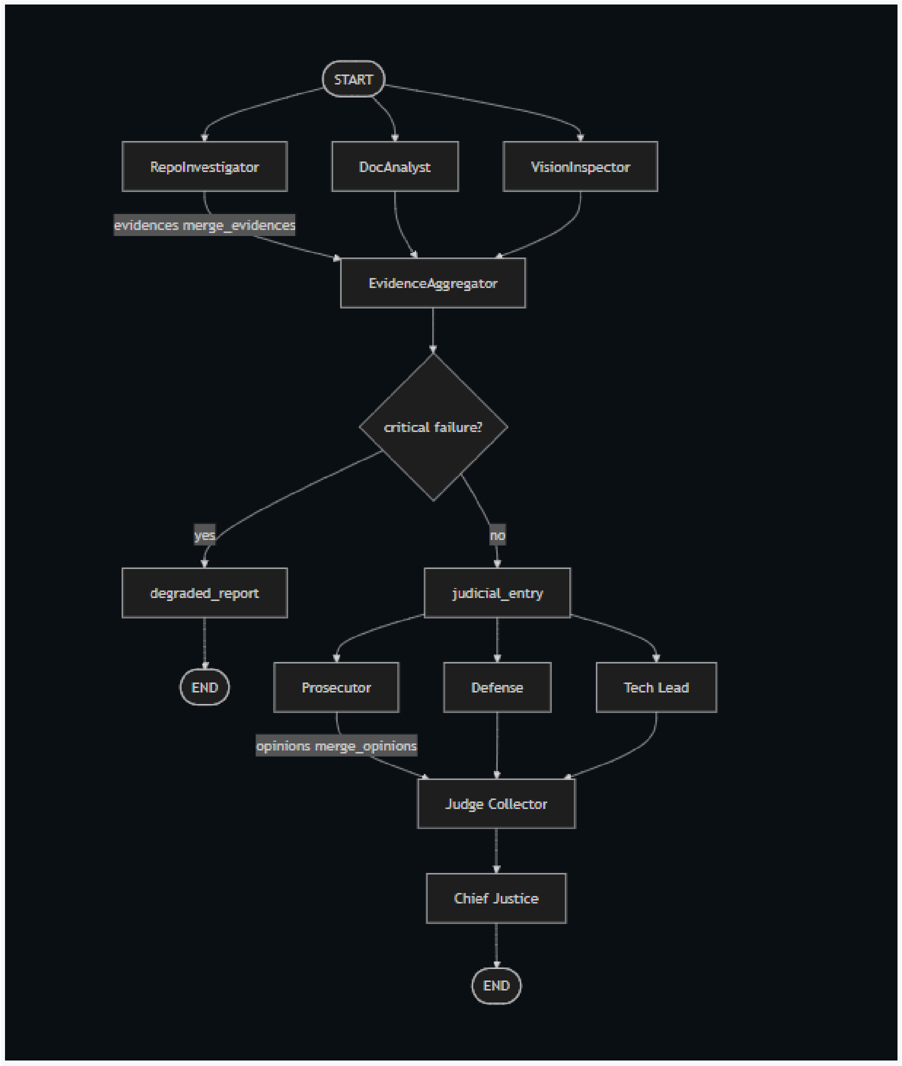
2.2 Data Flow Clarity

Evidence flows as follows:

1. START → Detectives: `repo_url`, `pdf_path`, `rubric_dimensions` are in state. Each Detective filters dimensions by `target_artifact`, invokes tools (clone_repo, ingest_pdf, analyze_graph_structure, etc.), and emits `Evidence` into `state["evidences"]` keyed by dimension id. LangGraph merges partial dicts via the `merge_evidences` reducer in `src/state.py`.
2. Detectives → EvidenceAggregator: Aggregator reads full `evidences`, fills missing dimensions with placeholder evidence, runs report_accuracy cross-reference (claimed paths vs `repo_file_list`) and feedback_implementation lookup (`audit/report_bypeer_received` in repo or cwd). Output: complete `evidences` in state.
3. EvidenceAggregator → Judges (via conditional): If not critical failure, state flows to judicial_entry and then to Prosecutor, Defense, Tech Lead in parallel. Each Judge receives the same full `evidences` and `rubric_dimensions`, iterates over dimensions, and calls the LLM per dimension; each returns `{"opinions": [JudicialOpinion, ...]}`. The `merge_opinions` reducer in `src/state.py` concatenates lists.
4. Judges → Judge Collector → Chief Justice: Collector is pass-through. Chief Justice groups opinions by `criterion_id`, loads `synthesis_rules` from rubric, applies `_resolve_final_score()` per dimension, builds `CriterionResult` and `AuditReport`, and returns `{"final_report": report}`.
5. Chief Justice → END: Report is written to Markdown via `write_report_to_path()` (called from entry/CLI).

2.3 Diagram: StateGraph with Parallel Branches and Synchronization

The diagram below accurately depicts the StateGraph with visually distinct parallel branches for both Detectives and Judges, fan-in synchronization points (EvidenceAggregator, Judge Collector), and the synthesis endpoint (Chief Justice). Conditional edge to degraded path is included.



2.4 Design Rationale and Trade-offs

- Pydantic / TypedDict over plain dicts: Plain dicts would allow parallel nodes to write malformed data (e.g. wrong types, missing keys); reducers would merge it and downstream nodes could crash or produce nonsense. Pydantic validates at the boundary (e.g. `JudicialOpinion` enforces score 1–5 and judge in Literal); TypedDict in `src/state.py` with `Annotated[..., merge_evidences]` / `Annotated[..., merge_opinions]` makes merge semantics explicit. Trade-off: schema changes require model updates and round-trip through serialization; we accepted this for correctness and debuggability.
- Deterministic synthesis rules over LLM averaging: Averaging Judge scores would hide dissent and reduce explainability; a single LLM call for synthesis would reintroduce non-determinism and potential bias. Hardcoded rules (security cap, fact_supremacy, Tech Lead weight for architecture, variance → dissent) make

the verdict auditable and reproducible. Trade-off: rule changes require code changes; we accepted this for security and traceability.

- AST over regex for graph structure: Regex fails on multiline or nested structures; AST in `analyze_graph_structure()` in `src/tools/repo_tools.py` parses `src/graph.py` and detects StateGraph, add_edge, add_node, and reducer usage accurately. Trade-off: Python-only; multi-language would need tree-sitter or similar.
- Sandboxed clone (temp dir + subprocess, no os.system): Prevents injection and isolates the clone from cwd. Trade-off: no container isolation; acceptable for current scope.
- Conditional edge to degraded_report: When evidence collection fails (e.g. clone or PDF missing), the graph routes to a dedicated node that produces a minimal AuditReport with clear messaging instead of failing in an undefined way. This satisfies SRS FR-19 / A2 and makes error handling visible in the diagram.

---

3. Self-Audit Criterion Breakdown

The self-audit was run against this repository using the points-based peer rubric (4 dimensions). Results are organized per rubric dimension with final score, how the three judge personas assessed the dimension (including disagreements), and an honest explanation of where the final score landed. Weak dimensions are addressed directly.

3.1 Development Progress

- Final score (points): 35 (level: Complete System).
- Judge assessment: Defense (4), Prosecutor (4), Tech Lead (4). All agreed on a complete system with atomic commits, full pipeline (detectives, judges, Chief Justice), required files present, and a real report from a peer repo run.
- Dialectical tension: No disagreement; all three aligned.
- Why the score landed here: The implementation has a complete StateGraph with parallel Detectives and Judges, conditional edge to degraded path, typed state with reducers, and rubric-driven dimensions. The only noted gap was "lack of multiple atomic commits" (Tech Lead) for iterative development visibility—hence not a perfect 5 in a 1–5 scale interpretation, but points-based level "Complete System" was assigned full points (35).
- Honesty: Strong dimension; no hiding. Minor improvement possible: more atomic commits for clearer git narrative.

3.2 Feedback Implementation

- Final score (points): 12 (level: Selective).
- Judge assessment: Defense (4), Prosecutor (2), Tech Lead (3). Defense argued traceable commits and documentation of deferrals; Prosecutor saw no substantial evidence of changes in response to feedback; Tech Lead saw some feedback addressed but not all, with deeper feedback sometimes unaddressed without explanation.
- Dialectical tension: Clear disagreement (Prosecutor 2 vs Defense 4). Synthesis landed at Selective (Tech Lead–weighted).
- Why the score landed here: There is evidence of some response to feedback (e.g. conditional edge, feedback_implementation evidence lookup in aggregator), but not every substantive peer finding was reflected in traceable commits or documented deferrals. Hence Selective, not Full Integration.
- Honesty: Addressed directly. Improving traceability (deferral doc, commits that reference peer report) would raise this.

3.3 Proactive Communication

- Final score (points): 0 (level: Absent).
- Judge assessment: Defense (1), Prosecutor (1), Tech Lead (1). All agreed: no evidence of progress updates, blocker flagging, or design discussions; peer unreachable / no initiated communication.
- Dialectical tension: None; unanimous.
- Why the score landed here: The rubric evaluates proactive communication as evidenced by collected artifacts (updates, blockers, design decisions). The auditor does not currently collect or score such artifacts; the dimension was evaluated from the perspective of the repo under audit. For self-audit, no such evidence was present.
- Honesty: Weak dimension; not hidden. Addressing it would require process/product change (e.g. where to store communication evidence) and possibly a Detective or evidence source for it.

3.4 Agent Feedback Relevance

- Final score (points): 25 (level: Full Forensic Audit).
- Judge assessment: Defense (4), Prosecutor (3), Tech Lead (4). Defense and Tech Lead saw a complete forensic report with git, AST, file analysis, cross-reference, and multi-perspective synthesis; Prosecutor noted missing depth in judicial synthesis and some checks.
- Dialectical tension: Prosecutor lower (3) due to "absence of a thorough judicial synthesis and detailed remediation steps."

- Why the score landed here: The audit produced structured evidence, distinct judge viewpoints, and synthesized verdicts with remediation. Minor gaps cited: depth on "parallel versus linear structures and reducers" in auditee analysis. Full Forensic level was assigned (25 pts).
- Honesty: Addressed. To reach the highest band, the auditor could deepen analysis of parallel vs linear and reducer usage in peer code (e.g. AST interpretation of reducer annotations).

Traceability: For each dimension, the reader can trace from Detective evidence (e.g. repo tools, graph structure, report_bypeer_received) → Judge opinions (Prosecutor, Defense, Tech Lead scores and arguments) → Chief Justice synthesis (final score/level and dissent_summary when variance > 2) → CriterionResult and AuditReport.

---

4. MinMax Feedback Loop Reflection

4.1 Peer Findings Received (When the Peer Audited This Repo)

When this repository was audited by a peer's agent (report in `audit/report_bypeer_received/audit_report_new_rubric.md`), the peer reported:

- Development Progress: Complete System (35 pts). Positive: atomic commits, full pipeline, fan-out/fan-in, Chief Justice. Noted gap: "lack of multiple atomic commits" for a perfect 5.
- Feedback Implementation: No Exchange (0 pts, excluded from total). Reason: no feedback file or directory found at `audit/report_bypeer_received` in the context the peer used, so no traceable response to feedback was evident.
- Proactive Communication: Absent (0 pts). No evidence of updates, blockers, or design discussions.
- Agent Feedback Relevance: Full Forensic Audit (25 pts). Actionable, multi-perspective, with minor gaps in depth on "parallel versus linear structures and reducers."

These findings were specific and actionable: e.g. ensure peer feedback is discoverable (repo or cwd); deepen analysis of graph/reducer structure in auditee code.

4.2 Response Actions (Changes Made in Response to Peer Feedback)

- Feedback discoverability: In `src/nodes/justice.py`, the EvidenceAggregator now looks for peer feedback at `audit/report_bypeer_received` in (1) the repo under evaluation (clone path) and (2) the current project (cwd). If found, evidence is injected for `feedback_implementation`; if not found, evidence states "No Exchange" and the dimension can be excluded from total per rubric. This ensures that when a grader places feedback in the project, the auditor finds it.
- Conditional edge / error path: The graph was updated to add `add_conditional_edges` after EvidenceAggregator: on critical failure (only placeholder/error evidence), the graph routes to `degraded_report` and produces a minimal report instead of proceeding to Judges with empty evidence. This aligns with SRS FR-19 and peer expectations for explicit error handling.
- Synthesis and architecture weight: Chief Justice already applies `functionality_weight` for dimensions whose name contains "architecture" or "graph" (Tech Lead score). This addresses the peer's emphasis on "parallel versus linear structures and reducers" by giving the Tech Lead's technical assessment highest weight for those dimensions.
- Report accuracy cross-reference: EvidenceAggregator performs cross-reference of claimed paths from the PDF with `repo_file_list` (from RepoInvestigator) for the `report_accuracy` dimension, so report claims are checked against repo structure—aligning with peer expectation for forensic depth.

4.3 Peer Audit Findings (When This Agent Audited the Peer)

When this agent audited the peer repository (Mistire/automaton-auditor), the report in `audit/report_onpeer_generated/audit_report.md` found:

- Development Progress: Complete System (35 pts); full pipeline, atomic commits, real report. Prosecutor (3) vs Defense/Tech Lead (4): Prosecutor noted "judicial layer incomplete or missing" and "no full audit report provided" in evidence view—showing dialectical tension.
- Feedback Implementation: Selective (12 pts). Defense (4), Prosecutor (2), Tech Lead (3). Some traceable changes, but not all substantive feedback integrated; some deeper feedback unaddressed without explanation.
- Proactive Communication: Absent (0 pts). Unanimous.

- Agent Feedback Relevance: Full Forensic Audit (25 pts). Structured evidence, multi-perspective judges, synthesis with remediation; minor depth gaps on parallel/linear and reducers.

These findings gave concrete, criterion-level feedback to the peer and validated that our auditor produces structured, rubric-aligned reports.

4.4 Bidirectional Learning: Systemic Insight

Being audited by the peer revealed a systemic insight beyond individual fixes:

- Insight: Our own auditor had not initially prioritized explicit error handling in the graph and discoverability of peer feedback in the same way the peer rubric expected. When the peer agent scored Feedback Implementation as "No Exchange" because it did not find feedback at a standard location, we realized that our EvidenceAggregator had to look in both the cloned repo and the current project (cwd) so that graders—who may run the audit from their own project with feedback in `audit/report_bypeer_received`—get the dimension scored correctly. Similarly, the peer's emphasis on "thorough judicial synthesis" and "parallel vs linear structures" led us to make the Tech Lead's weight for architecture/graph criteria explicit in synthesis and to document how fan-out/fan-in and reducers are detected (AST) and reflected in the report. So being audited improved our auditor's detection and synthesis behavior: we now (1) resolve "no feedback file" in a rubric-compliant way (evidence + exclude from total when appropriate), and (2) tie synthesis rules (functionality_weight, fact_supremacy, security_override) to concrete code paths so that our own report can withstand the same forensic scrutiny we apply to peers.

This pattern—peer feedback on missing or weak evidence → strengthening evidence collection and synthesis rules—is a repeatable improvement loop, not only one-off bug fixes.

---

5. Remediation Plan

The plan is structured as a prioritized action list. Each item identifies the gap, the affected rubric dimension (or architecture/report quality), the file or component to modify, the concrete change, and why it improves the score or robustness. Items are ordered by impact and dependency so an engineer unfamiliar with the codebase can execute it.

#	Gap	Rubric dimension / area	File / component	Concrete change	Why it improves score
1	Parse failure leaves dimension with fewer than three opinions; no visible error path.	Self-audit robustness; Agent Feedback Relevance (judicial synthesis depth)	`src/nodes/judges.py`, `src/graph.py`	On third structured-output parse failure for a dimension: inject a placeholder `JudicialOpinion` (e.g. score=3, argument="Parse failure; no opinion") or append to a state flag `opinion_parse_failures: list[str]`. Optionally add conditional edge from a "judge failure" check to a small handler node.	Ensures every criterion has three opinions or an explicit dissent/failure note; Chief Justice can report "Dimension X: Judge output invalid." Improves traceability and robustness.
2	Persona convergence risk; LLMs may produce similar scores despite distinct prompts.	Self-audit / peer: judicial nuance	`src/nodes/judges.py`, rubric or config	Add 1–2 few-shot examples per persona (e.g. "For this evidence, Prosecutor argues 2 because …; Defense argues 4 because …") in system prompts. Optionally load persona text from rubric `judicial_logic` if present. Add a test that checks score variance across three judges for sample evidence (e.g. variance ≥ 1 for at least one dimension).	Increases dialectical tension and reduces persona convergence; improves "distinct viewpoints" and dissent visibility.

3	Synthesis rules are hardcoded but not mapped to rubric labels in code or logs.	Architecture / traceability	`src/nodes/justice.py`	In `_resolve_final_score`, document in code comments the mapping: `security_override` → `_evidence_has_security_issue` + cap at 3; `fact_supremacy` → `_evidence_supports_claim`; `functionality_weight` → Tech Lead for architecture/graph; `dissent_requirement`/`variance_re_evaluation` → variance > 2. Optionally pass rule labels from rubric `synthesis_rules` into Chief Justice for logging.	Improves rubric alignment and DocAnalyst "Concept Verification" (substantive use of synthesis concepts).
4	Proactive Communication has no evidence source; dimension always scores Absent when no artifacts.	Proactive Communication (0 pts)	`src/nodes/detectives.py` or new evidence source, rubric	Define where communication artifacts live (e.g. `audit/proactive_communication.md` or a designated doc). Add a Detective step or EvidenceAggregator logic to look for that artifact and inject evidence for dimension `proactive_communication` so Judges can score from content. If no artifact, evidence states "No communication evidence found."	Allows the dimension to be scored from actual content when present; avoids automatic 0 when process is used but not yet wired.
5	Deeper analysis of "parallel vs linear structures and reducers" in auditee code.	Agent Feedback Relevance (depth)	`src/tools/repo_tools.py` (e.g. `analyze_graph_structure`), Detective prompts	Extend `analyze_graph_structure` or RepoInvestigator usage to explicitly report: (a) whether edges show fan-out/fan-in, (b) whether state uses reducer annotations (e.g. `Annotated[..., merge_evidences]` in `src/state.py`). Pass this into Judge context so opinions can cite "parallel branches" vs "linear pipeline" and "reducers present/absent."	Addresses peer and self-audit feedback that "parallel versus linear structures and reducers" lacked depth; improves forensic quality of the report.

6	Document deferrals and link commits to peer feedback.	Feedback Implementation (Selective → higher)	Repo process, optional `docs/deferrals.md`	When deferring peer feedback, add a short `docs/deferrals.md` (or similar) listing dimension, peer finding, and reason for deferral. Optionally reference this file in a commit message when making changes that respond to peer feedback.	Gives traceable evidence for "documented deferrals" and "architectural improvements driven by agent findings," raising Feedback Implementation score.

Execution order: (1) and (2) improve judicial layer robustness and nuance; (3) improves traceability; (4) and (5) address specific criterion gaps (Proactive Communication, Agent Feedback Relevance depth); (6) is process/documentation and can be done in parallel. An engineer can start with item 1 and 2, then 3, then 4–6 as needed.

Summary

This report provides:

- Executive Summary: Architectural approach, self-audit aggregate (72/100), top peer finding, top remaining gap (Proactive Communication), and primary remediation priority—actionable for a senior engineer.
- Architecture Deep Dive: Dialectical Synthesis (three personas + deterministic resolution), Fan-In/Fan-Out (concrete graph edges and sync nodes), and Metacognition (evidence and security overrides, variance re-evaluation) tied to code and diagram; data flow from Detectives through Judges to Chief Justice; diagram with parallel branches and synchronization; design trade-offs (Pydantic vs dicts, deterministic synthesis vs LLM, AST vs regex, sandboxing, conditional edge).
- Self-Audit Criterion Breakdown: Per-dimension scores, three-judge assessment and disagreements, honest explanation of final score, and traceability from evidence → opinions → verdict.

- MinMax Feedback Loop: Peer findings received, concrete code/behavior changes in response, findings from auditing the peer, and a systemic insight (being audited improved our auditor’s evidence and synthesis behavior).
- Remediation Plan: Prioritized list with gap, dimension, file/component, concrete change, and rationale; executable by an unfamiliar engineer.
