INTERIM ARCHITECTURE REPORT (TEXT PLACEHOLDER)

Executive summary

This interim report documents the current design and implementation progress of the **“Automaton Auditor” Digital Courtroom**. The codebase includes typed state models (`Evidence`, `JudicialOpinion`, `AgentState`), sandboxed git tooling, and a LangGraph `StateGraph` wiring the Detective layer in parallel with a synchronization node.

Architecture decisions so far (with trade-offs)

Pydantic + `TypedDict` over plain dicts

- **Decision:** Use Pydantic `BaseModel` for `Evidence` and `JudicialOpinion` to ensure strongly-typed, validated objects exchanged between nodes. Keep `AgentState` as a `TypedDict` with `Annotated` reducers (e.g., `operator.add`, `operator.ior`) so parallel branches can safely merge lists and dicts.
- **Trade-offs:** Pydantic adds runtime validation overhead and stronger coupling to the schema. In this governance/audit setting the clarity and safety provided by types outweigh the small performance cost.
- **Alternatives considered:**
    - Pure dict-based state — rejected (too brittle for parallel reducers, hard to validate at graph boundaries).
    - Single monolithic Pydantic model — rejected (would complicate LangGraph integration and create tightly coupled schemas).

AST-based analysis over regex scanning

- **Decision:** Use Python's `ast` module (and possibly `tree-sitter`) for repo forensics rather than regex-based searches to understand graph wiring and state structures.
- **Trade-offs:** AST parsing is more complex and heavier than simple string matching, but robust to formatting changes and avoids false positives (e.g., commented mentions of `StateGraph`).
- **Rejected alternative:** Regex scanning — considered for speed/simplicity but rejected because the rubric requires structural verification rather than mere token presence.

Sandboxed git tooling

- **Decision:** `src/tools/repo_tools.py` clones repositories into a `tempfile.TemporaryDirectory` and runs `git log --oneline --reverse` to build a commit narrative; it never touches the live working tree.
- **Trade-offs:** Temporary directories require careful lifetime management and slightly more code, but they reduce the blast radius of cloning untrusted repos.
- **Rejected alternative:** Cloning directly into the workspace (e.g., `os.system("git clone ...")`) — rejected as a security liability.

RAG-lite PDF forensics

- **Decision:** `src/tools/doc_tools.py` ingests PDFs once, chunks/indexes content, and supports targeted queries for concepts such as Dialectical Synthesis, Fan-In/Fan-Out, and Metacognition — rather than dumping the entire report into a single LLM prompt.
- **Trade-offs:** RAG-lite adds complexity (chunking, indexing, targeted queries) but keeps context windows small and allows citing specific sentences as Evidence.
- **Rejected alternative:** Naive single-prompt summarizer — rejected because it's hard to ground forensic claims to exact text spans.

StateGraph flow (Detective focus)

1. **Entry:** `context_builder` loads rubric dimensions into state.
2. **Fan-out Detectives (parallel):**
    - `repo_investigator` — Code Detective (git, AST, graph, state)
    - `doc_analyst` — Paperwork Detective (PDF and textual forensics)
    - `vision_inspector` — Diagram Detective (image/diagram analysis; currently a structural placeholder)
3. **Fan-in:** `evidence_aggregator` synchronizes inputs. Because `evidences` uses an `operator.ior` reducer, each Detective can contribute evidence without overwriting others.
4. **Judicial fan-out:** Evidence is reviewed by multiple judges (parallel), producing `JudicialOpinion` lists.
5. **Chief Justice:** Deterministic synthesis of opinions into an `AuditReport` (Markdown).

Known gaps, risks, and forward plan

RepoInvestigator

- **Planned:** Implement AST parsing to validate `AgentState` structure, reducers, and StateGraph fan-out/fan-in wiring. Capture the exact graph definition block as Evidence.
- **Risks:** AST visitors might miss dynamically constructed graphs or unusual `add_edge` patterns, causing false negatives.
- **Mitigation:** Start with a narrow set of supported patterns and document assumptions in the Evidence rationale.

DocAnalyst

- **Planned:** Implement robust PDF parsing and file-path extraction; cross-reference claims with repository contents to flag hallucinated paths.
- **Risks:** OCR/parse errors or unusual formatting can hide paths and phrases, causing under-reporting.
- **Mitigation:** Keep raw extracted text snippets in Evidence and enable manual inspection during debugging.

VisionInspector

- **Planned:** Add image extraction and multimodal analysis to classify diagrams and verify that parallel Detectives/Judges and Chief Justice synthesis are shown rather than a linear pipeline.
- **Risks:** Small, dense, or low-quality diagrams may be misclassified.
- **Mitigation:** Treat vision findings as complementary and explicitly record absence of diagrams.

Judicial layer & Chief Justice

- **Planned:** Replace placeholders with LLM-backed judge personas that output `JudicialOpinion` (structured). Implement deterministic ChiefJustice rules (security override, fact supremacy, dissent requirement) to synthesize scores into a Markdown `AuditReport`.
- **Risks:** Persona collusion (near-identical opinions) or hallucinated citations.
- **Mitigation:** Enforce `.with_structured_output(JudicialOpinion)` with strict validation and retries; design prompts to emphasize philosophical differences.

Sequencing & prioritization

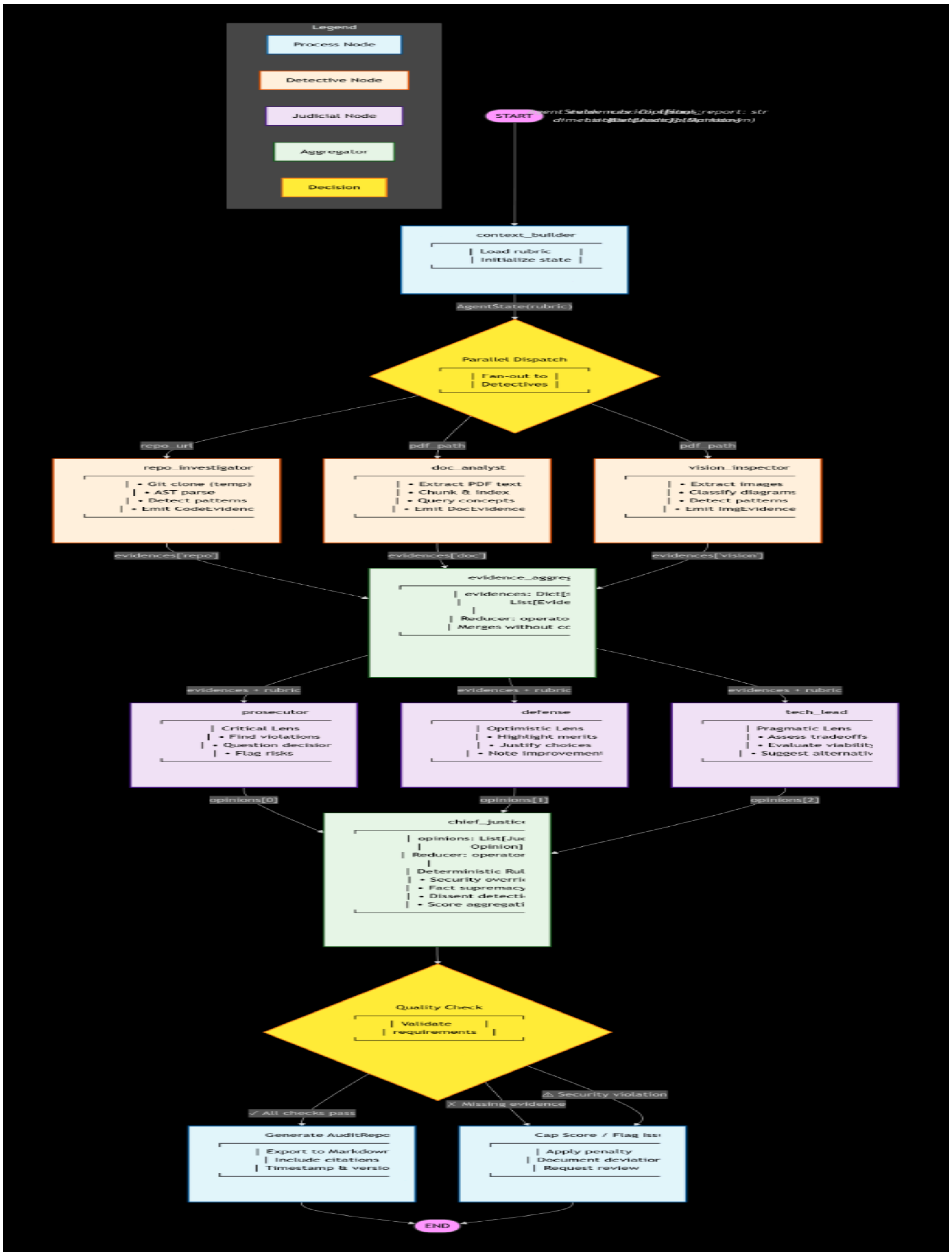
1. **Stabilize forensic foundation** — AST-based RepoInvestigator + robust PDF ingestion (foundation for all higher judgments).

2. **Harden judicial personas** — structured LLM judges, tune prompts to produce meaningfully divergent, grounded opinions.

3. **Implement Chief Justice rules** — encode overrides, fact supremacy and dissent checks as deterministic Python logic; wire score-variance checks.

4. **Enhance diagrams & vision** — add StateGraph diagrams and VisionInspector after textual evidence and judicial reasoning are reliable.

StateGraph architecture diagram (Mermaid)


Legend
Process Node
Detective Node
Judicial Node
Aggregator
Decision
START
context_builder
Load rubric
Initialize state
AgentState(rubric)
Parallel Dispatch
Fan-out to Detectives
repo_url
pdf_path
pdf_path
repo_investigator
• Git clone (temp)
• AST parse
• Detect patterns
• Emit CodeEvidenc
doc_analyst
• Extract PDF text
• Chunk & index
• Query concepts
• Emit DocEvidence
vision_inspector
• Extract images
• Classify diagrams
• Detect patterns
• Emit ImgEvidence
evidences['repo']
evidences['doc']
evidences['vision']
evidence_aggreg
evidences: Dict[s
List[Evide
Reducer: operato
Merges without cc
evidences + rubric
evidences + rubric
evidences + rubric
prosecutor
Critical Lens
• Find violations
• Question decisior
• Flag risks
defense
Optimistic Lens
• Highlight merits
• Justify choices
• Note improvement
tech_lead
Pragmatic Lens
• Assess tradeoffs
• Evaluate viability
• Suggest alternativ
opinions[0]
opinions[1]
opinions[2]
chief_justice
opinions: List[Juc
Opinion]
Reducer: operator
Deterministic Rul
• Security overri
• Fact supremacy
• Dissent detecti
• Score aggregati
Quality Check
Validate requirements
✓ All checks pass
✕ Missing evidence
⚠ Security violation
Generate AuditRepo
Export to Markdowr
Include citations
Timestamp & versio
Cap Score / Flag Issu
Apply penalty
Document deviatior
Request review
END