FDE Challenge Week 2: The Automaton Auditor

Technical Report

Architecture Decisions in Langraph AgentState

1. **Pydantic over Dicts**

    - **Type Safety**: Pydantic provides strong type checking, allowing for clearer contracts regard
    - ing data structures. This reduces runtime errors by catching them at the data validation stage.
    - **Validation**: Automatic validation of data ensures that only valid structures are processed, simplifying error handling.
    - **Serialization/Deserialization**: Pydantic simplifies converting data between different formats (like JSON), making it easier to persist state or transfer data across networks.
    - **Nested Models**: Pydantic allows for the creation of nested models, which is tedious to implement with plain dictionaries. This leads to more organized and maintainable code.
2. **AST Parsing Structure**

    - **Modularity**: The AST (Abstract Syntax Tree) parsing is designed to be modular, allowing for easy extension and modification (e.g., adding new language constructs) without significant refactoring.
    - **Visitor Pattern**: A visitor pattern is employed to traverse and process the AST, enabling separation of concerns by dealing with different node types in a structured manner.
    - **Error Handling**: Specific error reporting mechanisms are integrated within the parsing structure, allowing for comprehensive feedback on syntax and semantic issues encountered during parsing.
3. **Sandboxing Strategy**

    - **Isolated Execution**: A sandboxing framework is implemented to execute code in an isolated environment, minimizing risks associated with executing untrusted code.
    - **Resource Limitation**: The sandbox enforces restrictions on CPU and memory usage, ensuring that any executed code cannot exhaust host system resources.
    - **Allowlist**: A strict allowlist of permissible libraries and functions is maintained to prevent access to sensitive system functionalities.

    - **Logging and Monitoring**: The sandboxing strategy incorporates thorough logging and monitoring of executed processes to detect and respond to potentially harmful activities in real-time.

Summary

These architectural decisions contribute to a robust, maintainable, and secure environment for the Langraph AgentState, facilitating effective handling of complex data structures and interactions while ensuring the system's integrity and reliability.

Graph

✔ 1. Parallel Evidence Collection

The first fan-out allows:

- Documentation analysis
- Code investigation
- Visual inspection

This is proper multi-modal ingestion.

Good separation of concerns.

✔ 2. Aggregation Layer

You introduced:

`EvidenceAggregator`

This is very important because it:

- Normalizes evidence format
- Removes duplication
- Creates unified context for judges

Many systems skip this layer — you did not.

✔ 3. Judicial Independence

Defense, Prosecutor, TechLead are parallel.

This is correct:

- No judge is influenced by another.
- Opinions are independent.

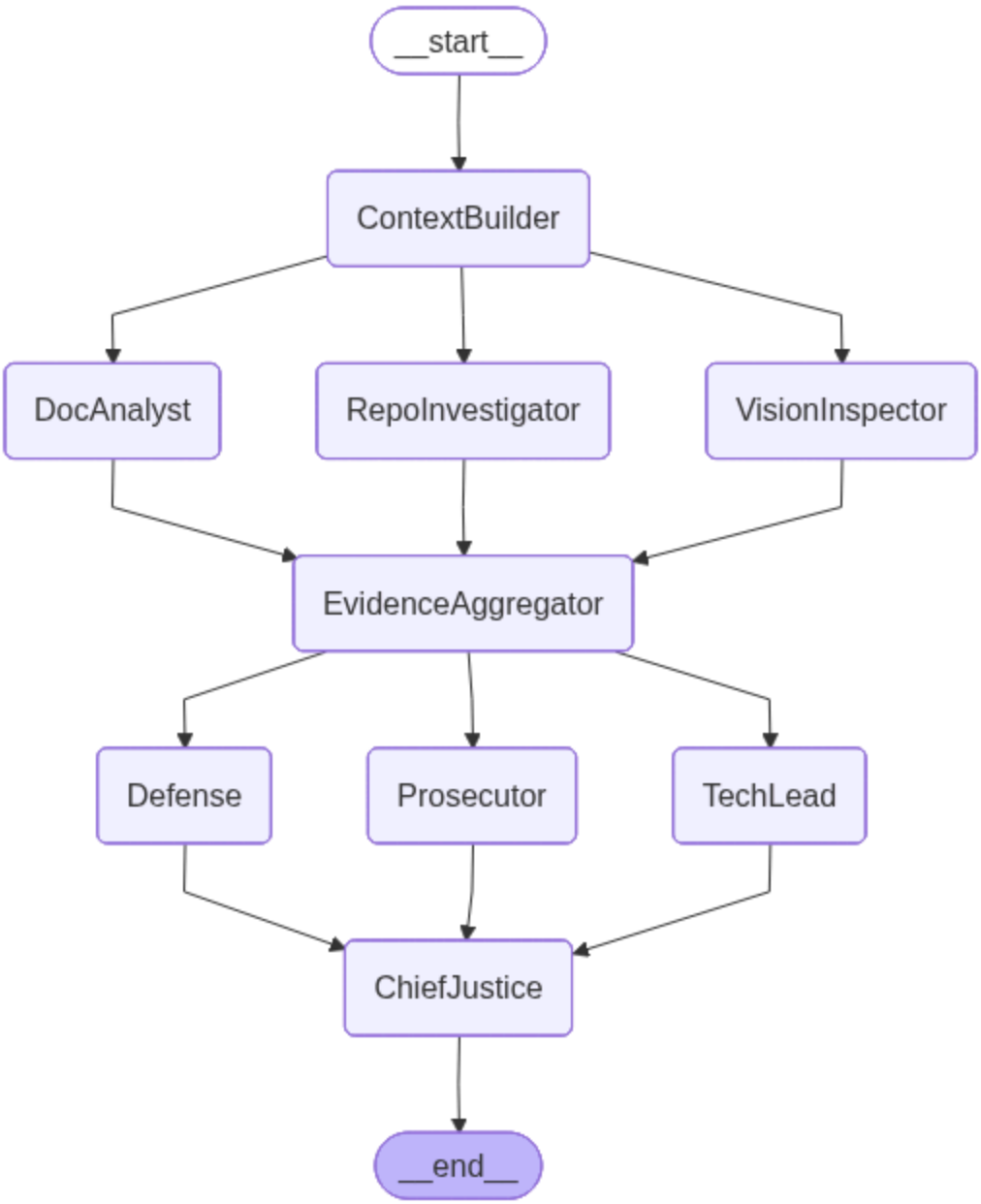
This is superior to sequential judge chaining.

✔ 4. Two Explicit Convergence Points

You have two merge layers:

1. EvidenceAggregator
2. ChiefJustice

This makes the reasoning pipeline interpretable.



Known Gaps in Current Judicial Layer

Right now judges:

- Don’t actually read evidence deeply
- Don’t use rubric scoring semantics

- Don’t handle multiple dimensions properly
- Don’t detect disagreement
- Don’t compute weighted aggregation
- Don’t generate structured reasoning chains
- Don’t detect insufficient evidence
- Don’t adapt to rubric dynamically

And Chief Justice:

- Averages scores blindly
- Doesn’t use dimension weights
- Doesn’t handle missing evidence
- Doesn’t produce dimension-level remediation
- Doesn’t detect variance between judges

So currently your system is:

- Architecture-level correct
- Reasoning-level shallow