Compliance Aware Prompt Compiler & Evidence Generator

A Compliance Constrained Task Compilation and Execution with Verifiable Audit Evidence

Confidential / To be filed for Patent

Owner: ArqAI

Inventor: Habib Mehmoodi

August 15, 2025

Background and Problem

Enterprise AI agents and orchestration platforms often accept natural language requests that may involve regulated data (PII/PHI, financial records) or high risk actions (modifying customer records, triggering payments). Traditional approaches plan and execute these requests without deeply understanding the compliance implications, applying policies only after the fact or leaving them to ad hoc manual reviews. This leads to inconsistent governance, missed policy violations and incomplete audit trails. ArqAI's positioning emphasises governance by default and cross domain orchestration; this invention implements that principle in a compiler style approach.

Inventive Concept

The invention introduces a compiler like pipeline that transforms free form requests into a compliance constrained intermediate representation (IR) annotated with data sensitivity, required approvals and policy preconditions. The IR is validated statically and dynamically against a policy graph. Execution is orchestrated through a trust aware engine (e.g. ArqFlow) that enforces preconditions and uses real time lineage and drift signals from ArqSight. After execution, the system generates a verifiable evidence bundle, a machine readable packet containing the request, IR, policy decisions, lineage hashes, model/tool versions and outputs, for audit and regulatory review.

Architecture and Components

1. Natural Language Parser: Extracts intents, entities, sensitivity levels, and jurisdictional constraints from the user's request.
2. Policy Graph / Compliance Knowledge Base: Encodes role based permissions, data sensitivity classifications, data transfer restrictions, and model/tool allowances (managed via ArqGuard).
3. Compliance IR Compiler: Translates the parsed request into a DAG of opcodes, each tagged with a sensitivity class, required approval level, geographic restriction, and model/tool constraints.
4. Static IR Validator: Checks the IR against the policy graph, ensuring that each opcode is allowed for the agent's role and within sensitivity thresholds. Attaches runtime checks.
5. Runtime Orchestrator: Executes the IR step by step, invoking models or tools (via ArqFlow) and evaluating real time lineage, drift or anomaly signals from observability components (ArqSight).
6. Evidence Generator: Assembles a signed evidence packet containing the request, IR, policy version, preconditions, approvals, lineage hashes, model/tool versions, results, timestamps, and previous evidence hash.
7. Exception Router: Handles violations at compile time or runtime; aborts, rolls back or forwards to manual review.

Process Flow

8. Receive Request: The user issues a natural language command.
9. Parse & Extract: The parser identifies the intent, affected records, and data sensitivity.

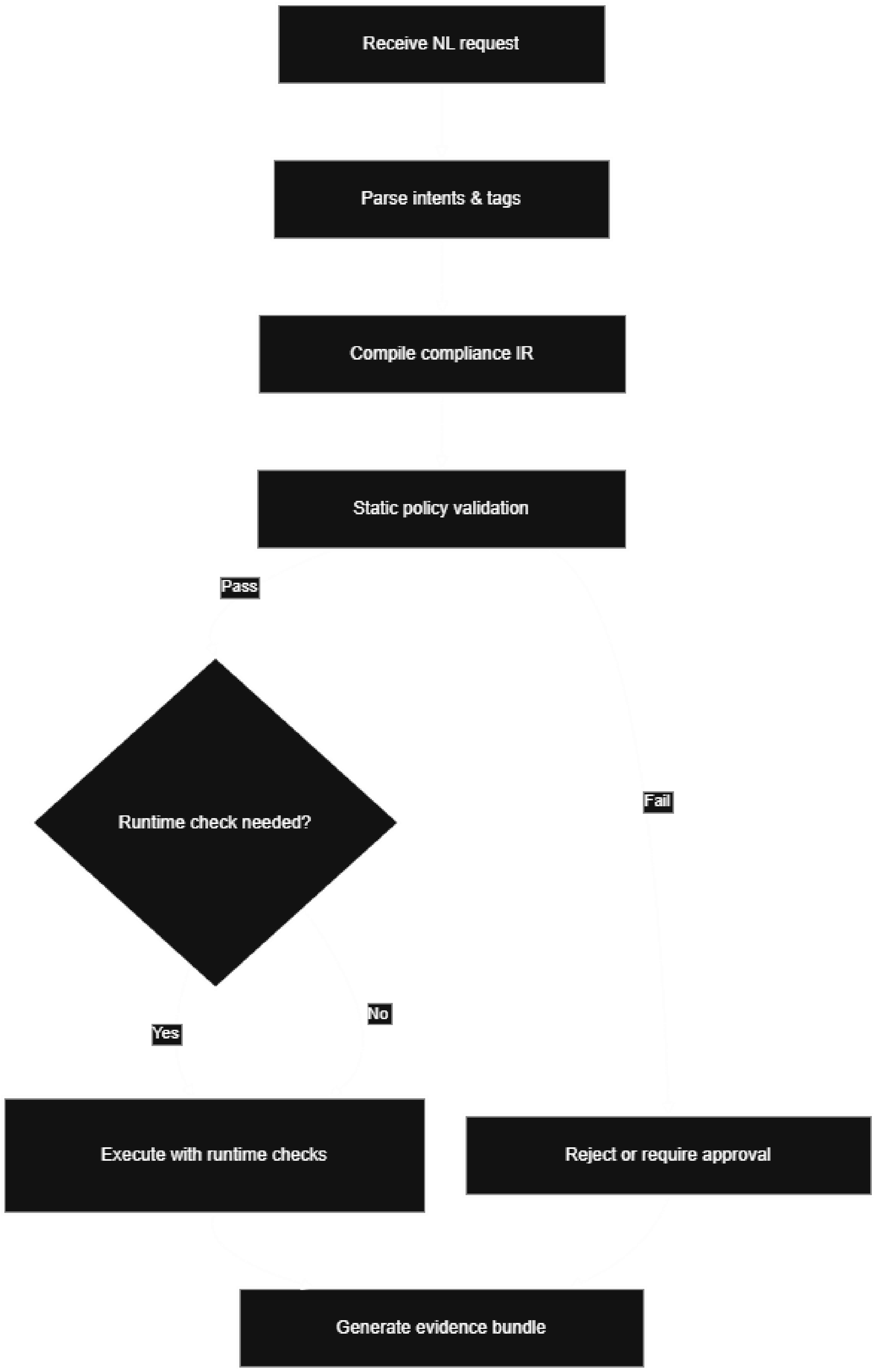
10. Compile IR: The compiler creates a DAG of operations with compliance attributes.
11. Static Validate: Each operation is checked against the policy graph.
12. Execute: The orchestrator executes operations sequentially, evaluating runtime conditions.
13. Generate Evidence: After execution or failure, an evidence packet is generated and stored.
14. Exception Handling: Violations at runtime are aborted or routed to human review.

Example Claims (illustrative, non exhaustive)

- Independent Method Claim: A method for compliance constrained task execution, comprising: (a) receiving a natural language request; (b) compiling the request into an intermediate representation comprising operations annotated with compliance attributes; (c) validating the intermediate representation against a policy graph; (d) executing permitted operations under runtime checks; and (e) generating a verifiable evidence artifact linking the request, the intermediate representation, policy decisions and outputs.
- Dependent Claim: The policy graph encodes data sensitivity classes, jurisdictional rules and role based allowances.
- Dependent Claim: Runtime checks evaluate data lineage, anomaly scores and drift signals; non conforming operations are sandboxed or aborted.
- Dependent Claim: The evidence artifact includes cryptographic hashes of inputs, outputs and policy versions.
- System Claim: A system comprising a processor and memory configured to perform the method above, plus a storage medium for audit receipts.

System Flow Diagram

The following diagram illustrates the compliance aware prompt compilation and execution process:


Receive NL request
Parse intents & tags
Compile compliance IR
Static policy validation
Pass
Runtime check needed?
Fail
Yes
No
Execute with runtime checks
Reject or require approval
Generate evidence bundle

Embodiments and Variations

- Jurisdiction overlays: Attach region predicates (EU/US/Asia) to operations to block cross border data movement.
- Model/tool neutrality: The IR references model/tool classes, not specific vendors.
- Adaptive thresholds: Risk thresholds and policy weights are tuned using past evidence packets and feedback loops.
- Deployment: The system can operate in cloud, on premises or air gapped environments; policies may be pulled from ArqGuard's governance layer.