1. Conceptual Foundation

Human-In-The-Loop (HITL)

Definition:
A governance approach in which human review, approval, or override is inserted at defined stages of AI system design, deployment, or operation.

Typical implementations:

- Manual approval before model deployment
- Escalation queues for flagged outputs
- Policy review committees
- Post-hoc audit validation
- Operator override buttons

HITL is primarily **procedural and organizational**.

SagaAI™ (Executable Governance Model)

Definition:
A deterministic runtime guard architecture that evaluates AI function invocation against structured governance objects (risk, controls, oversight, logging, classification) prior to state mutation, while preserving institutional authority .

SagaAI is:

- Infrastructure-based
- Object-graph anchored
- Provenance-aware
- Pre-commit validated
- Deterministic

It operationalizes governance frameworks (ISO 42001, NIST RMF, EU AI Act) as canonical class architecture. [1-3]

2. Governance Medium

Dimension	HITL	SagaAI™
Governance artifact	Documents, approvals, workflows	Canonical class objects with LOIDs
Evidence model	Reconstructed	Deterministic object lineage
Linkage to runtime	Indirect	Explicit, typed references
Validation timing	Pre-deployment or post-incident	Pre-commit at invocation boundary

Authority	Human	Human (preserved)
Enforcement type	Procedural	Structural + procedural

[4]

3. Structural Comparison

A. Identity Continuity

HITL

- Risk registers in governance platform
- Impact assessments in GRC system
- Model artifacts in MLOps system
- Logs in monitoring platform
- Human approval stored in ticketing tool

These are typically:

- Cross-system
- Versioned independently
- Reconciled during audits

There is no deterministic binding between:

- AI function invocation
- Risk object
- Control declaration
- Oversight classification

SagaAI™

Every governance artifact is:

- A canonical class instance
- Anchored to a Ledger Object ID (LOID)
- Immutable in provenance
- Cross-referenced via typed object linkage
  [4]

This satisfies the formal Referential Binding Condition:

AI function invocation must reference governance objects prior to state mutation

Result:
Governance is not reconstructed — it is structurally present.

4. Runtime Behavior

HITL Runtime Model

Common pattern:

1. AI runs
2. If threshold triggered → human review
3. Human approves or rejects
4. Execution continues

Limitations:

- Reactive
- Dependent on thresholds
- May bypass review if misconfigured
- Governance artifacts may not be structurally validated at invocation

HITL is event-driven escalation.

SagaAI™ Runtime Model

Deterministic pre-commit flow:

1. AI function invoked
2. SagaAI parser loads canonical class tree
3. Resolve governance object references
4. Evaluate guards:
    - Risk Guard
    - Data Governance Guard
    - Logging Guard
    - Oversight Guard
    - Least-Privilege Guard
5. Produce decision:
    - Commit
    - Abort
    - Escalate
    - Incident object creation
      [4]

Key distinction:

HITL inserts humans into workflow.
SagaAI binds governance state into runtime architecture.

Humans remain required for authorization, but structural sufficiency is machine-verified.

5. Authority Model

HITL

Human approval = execution permission.

However:

- Humans often rely on documents.
- Approval may not validate structural consistency across artifacts.
- Compliance posture may depend on interpretation.

SagaAI™

Authority Preservation Condition (formalized in paper):

$$Commit(F) \Rightarrow Authorized(InstitutionalActor) \land StructurallyValid$$

Important:

SagaAI does **not**:

- Determine legal compliance
- Grant certification
- Replace regulatory authority
  [4]

Humans remain final authority.

Difference:

- HITL = humans check governance.
- SagaAI = governance must exist structurally before humans can commit.

6. Standards Operationalization

HITL + Standards

Typical approach:

- ISO 42001 implemented via documentation
- NIST RMF maintained as risk workbook
- EU AI Act addressed via legal memos
- Audits validate documentation completeness

Governance is narrative and interpretive.

SagaAI™ + Standards

Standards encoded as:

- Canonical class domains
- Composable mixins
- Deterministic identity
- Typed references

ISO Clauses → mixins
NIST RMF → functional objects
EU AI Act → structural classification objects
Cyber overlay → composable cybersecurity mixins

Governance becomes:

Persistent governance state + runtime validation

Not document review.
[1-4]

7. Failure Modes

HITL Failure Modes

- Human fatigue
- Escalation overload
- Inconsistent interpretation
- Incomplete cross-system reconciliation
- Manual bypass
- Governance drift over time

SagaAI™ Failure Modes

- Incorrect schema modeling

- Incomplete object binding
- Guard misconfiguration
- Version evolution conflicts

However:

- Referential closure enforced
- No dangling references allowed
- Guard non-bypassability model-checked in TLA+ [5]

HITL risks are procedural.
SagaAI risks are architectural.

8. Comparison

HITL Audit

Auditor asks:

- Show risk assessment.
- Show control declaration.
- Show oversight structure.
- Show logs.

Organization reconstructs artifacts.

SagaAI Audit

Auditor queries:

- Governance graph for function X
- LOID lineage
- Risk object reference
- Control object linkage
- Guard certificate
- Oversight authorization

Evidence is deterministic object lineage.

This shifts audit from:

“Prove documents exist”

to:

“Verify structural referential integrity” [4]

9. Human Oversight: Replacement or Reinforcement?

Critical distinction:

SagaAI does **not** eliminate HITL.

Instead:

HITL becomes one governance object within SagaAI.

Human oversight (EU Article 14 modeling) is encoded structurally:

- Oversight flags
- Quorum requirements
- Institutional actor linkage
  [3]

SagaAI enforces:

AI cannot mutate governance state without human authorization. [4]

Thus:

HITL inside SagaAI is structurally enforced.
HITL alone is not structurally enforced.

10. Philosophical Difference

HITL Philosophy

"Humans ensure AI remains safe."

Governance = supervision.

SagaAI Philosophy

"Governance must exist as state before execution."

Governance = infrastructure.[4]

11. Formal Distinction

Let:

F = AI function invocation
H = Human approval
G(F) = Governance objects
V(F,G(F)) = Structural validation

HITL Model

Commit(F) ⇒ H

No structural requirement that G(F) exists or is coherent.

SagaAI Model

Commit(F) ⇒ H ∧ V(F,G(F)) = StructurallyValid

Human authority + deterministic structural binding.[4]

12. Where They Complement

SagaAI does not replace HITL.

It:

- Enforces existence of governance artifacts
- Ensures structural sufficiency
- Provides deterministic linkage
- Preserves institutional authority

HITL provides:

- Judgment
- Interpretation
- Context
- Ethical discretion

Together:

HITL = interpretive layer
SagaAI = structural substrate

13. Executive-Level Summary

Dimension	HITL	SagaAI™
Governance nature	Procedural	Structural + procedural
Evidence type	Documents	Object lineage
Runtime enforcement	Escalation-based	Pre-commit guard injection
Identity continuity	Weak	Deterministic LOID
Cross-framework interoperability	Manual	Typed object references
Compliance determination	Human interpretive	Human interpretive (preserved)
Structural sufficiency enforcement	No	Yes

[4]

14. Bottom Line

HITL answers:

Who approves AI behavior?

SagaAI answers:

Is governance structurally present and deterministically bound before execution?

HITL is necessary but insufficient.

SagaAI provides:

- Deterministic governance state
- Referential closure
- Runtime guard enforcement
- Provenance immutability
- Cross-framework composability

And critically:

It does so without displacing institutional authority.[4]

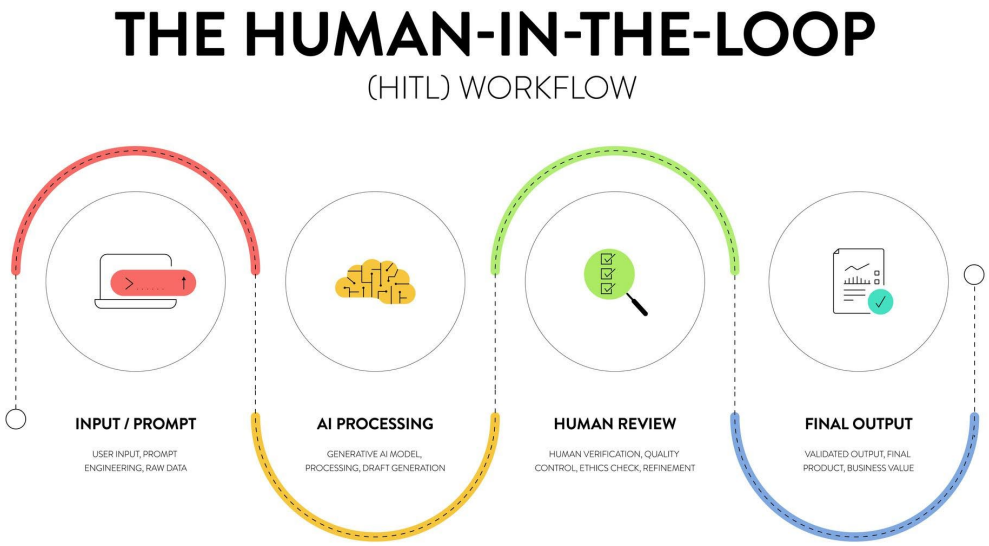
Below is a **formal architecture comparison diagram set** contrasting:

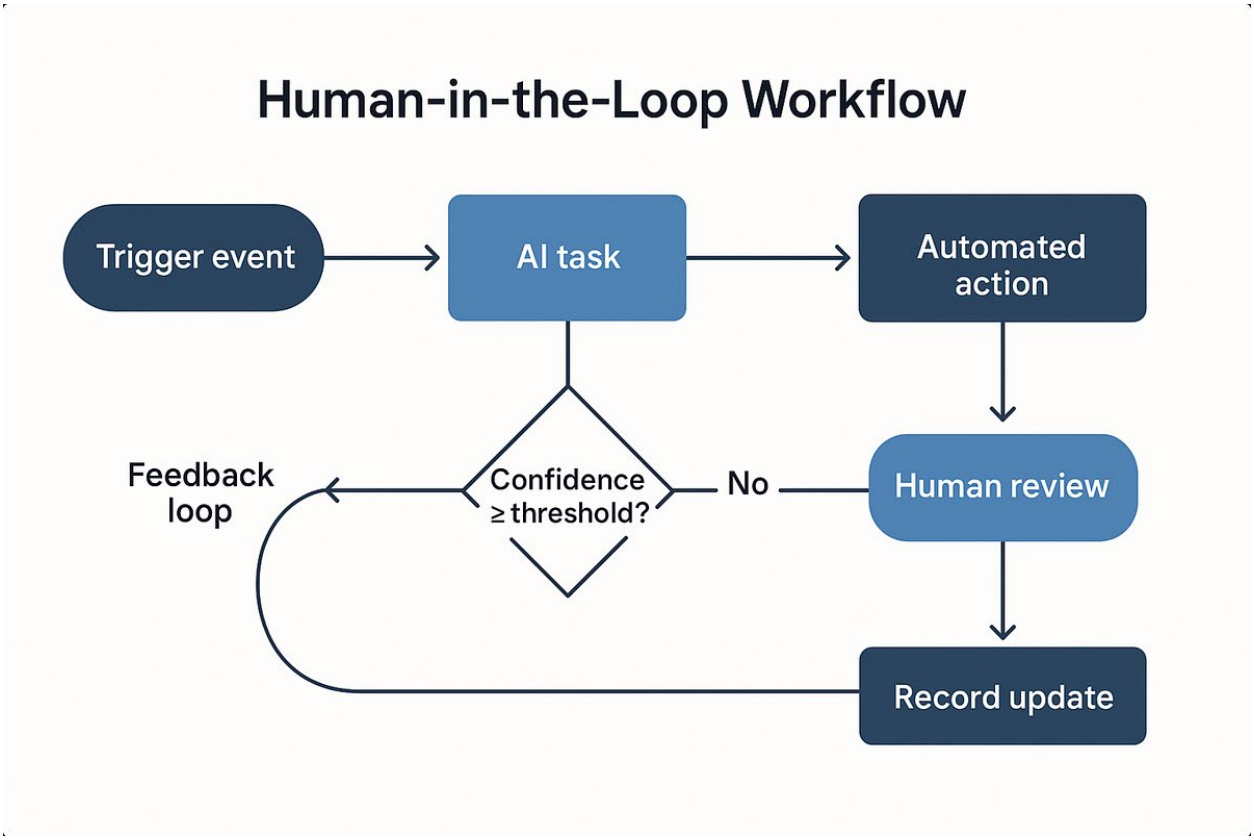
- **Traditional Human-In-The-Loop (HITL)** control paths
- **SagaAI Executable Governance** control paths

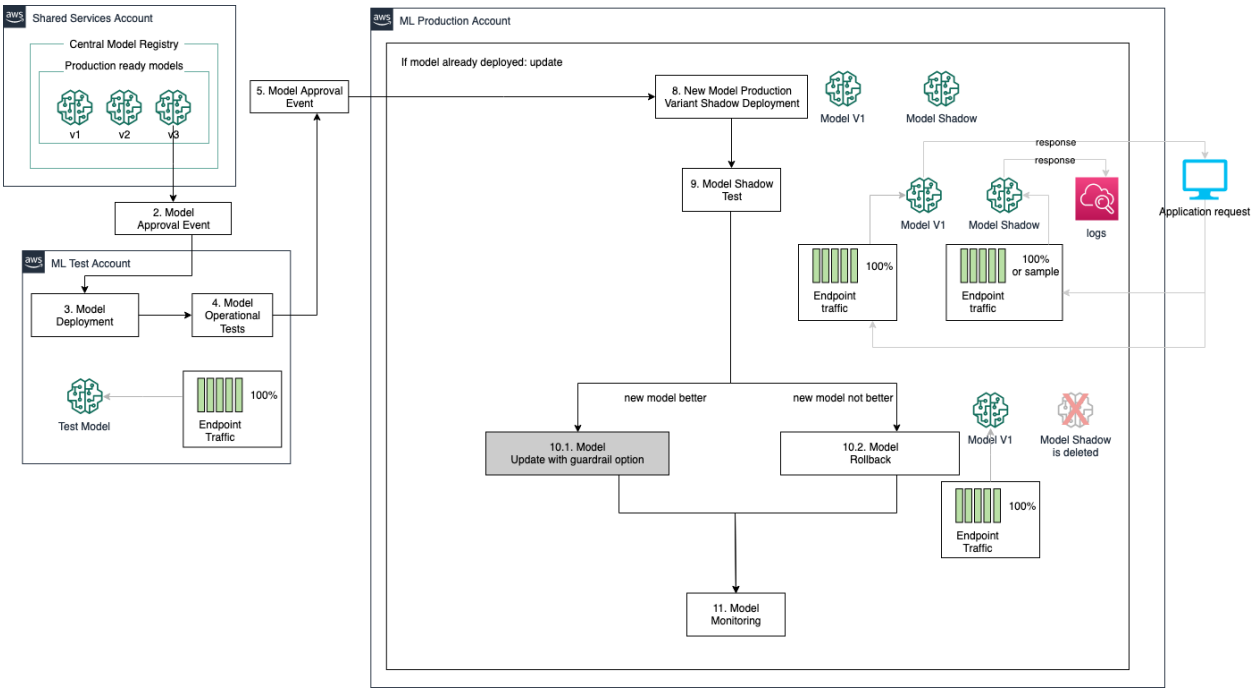
The diagrams are structured to match the formal runtime model and guard injection architecture defined in the Executable Governance paper .

Diagram 1 — HITL Control Path

(Procedural Governance Flow)




Human-in-the-Loop Workflow
Trigger event
AI task
Automated action
Feedback loop
Confidence ≥ threshold?
No
Human review
Record update


Shared Services Account
Central Model Registry
Production ready models
v1
v2
v3
2. Model Approval Event
ML Test Account
3. Model Deployment
4. Model Operational Tests
Test Model
100%
Endpoint Traffic
5. Model Approval Event
ML Production Account
If model already deployed: update
8. New Model Production Variant Shadow Deployment
Model V1
Model Shadow
response
response
9. Model Shadow Test
Model V1
Model Shadow
logs
Application request
100%
Endpoint traffic
100% or sample
Endpoint traffic
new model better
new model not better
10.1. Model Update with guardrail option
10.2. Model Rollback
Model V1
Model Shadow is deleted
100%
Endpoint Traffic
11. Model Monitoring

Structural Layout (Left → Right Flow)

```
AI Model Invocation
        ↓
Application Runtime
        ↓
Decision Threshold / Flag
        ↓
Escalation Queue
        ↓
Human Reviewer
        ↓
Approval / Reject
        ↓
System Commit
```

Architectural Characteristics

1. Governance Artifacts

- Risk register (separate system)
- Impact assessment document
- Control matrix
- Audit logs
- Policy repository

These are:

- Document-based
- Cross-system
- Linked by reference, not identity

2. Control Boundary

Validation occurs:

- After threshold
- After output generation
- Before final approval

Governance artifacts are:

- Consulted by humans
- Not structurally enforced at invocation boundary

3. Failure Surface

- Human fatigue

- Inconsistent interpretation
- Threshold misconfiguration
- Audit reconstruction burden
- Governance drift over time

4. Formal Representation

```
Commit(F) ⇒ HumanApproval
```

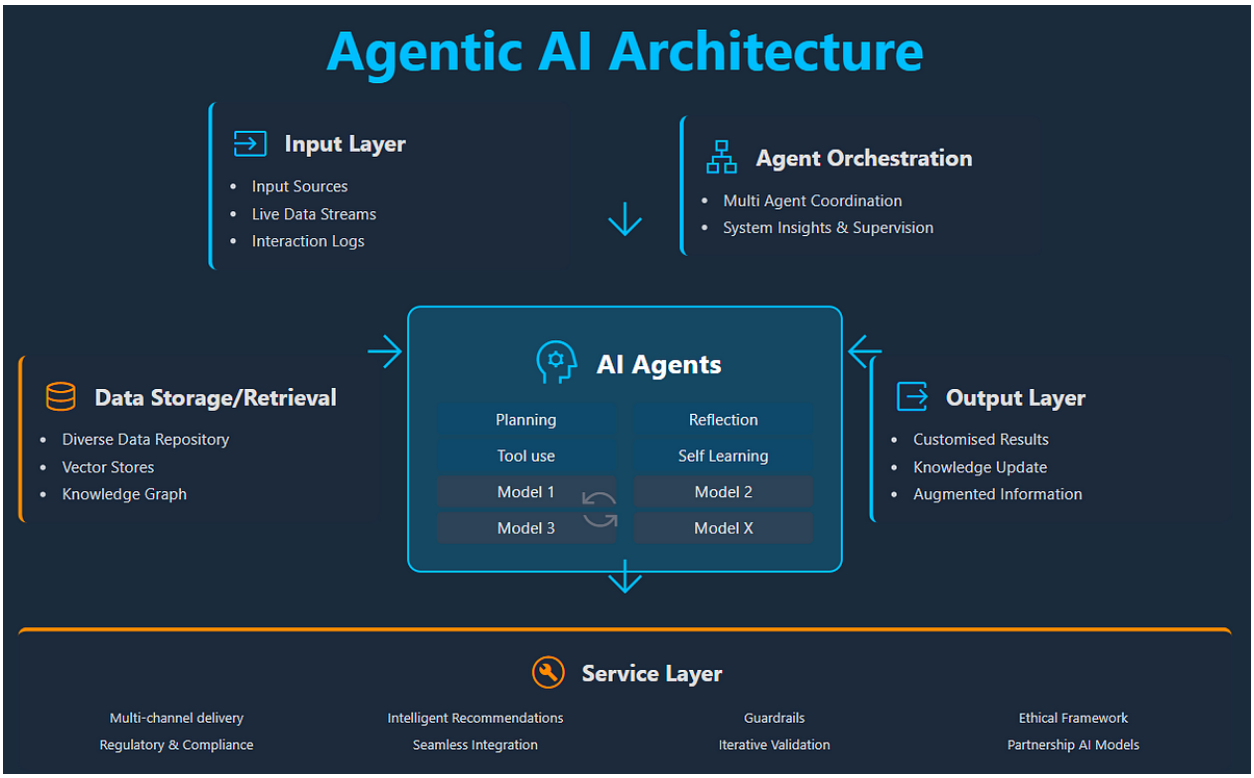
There is no deterministic requirement that:

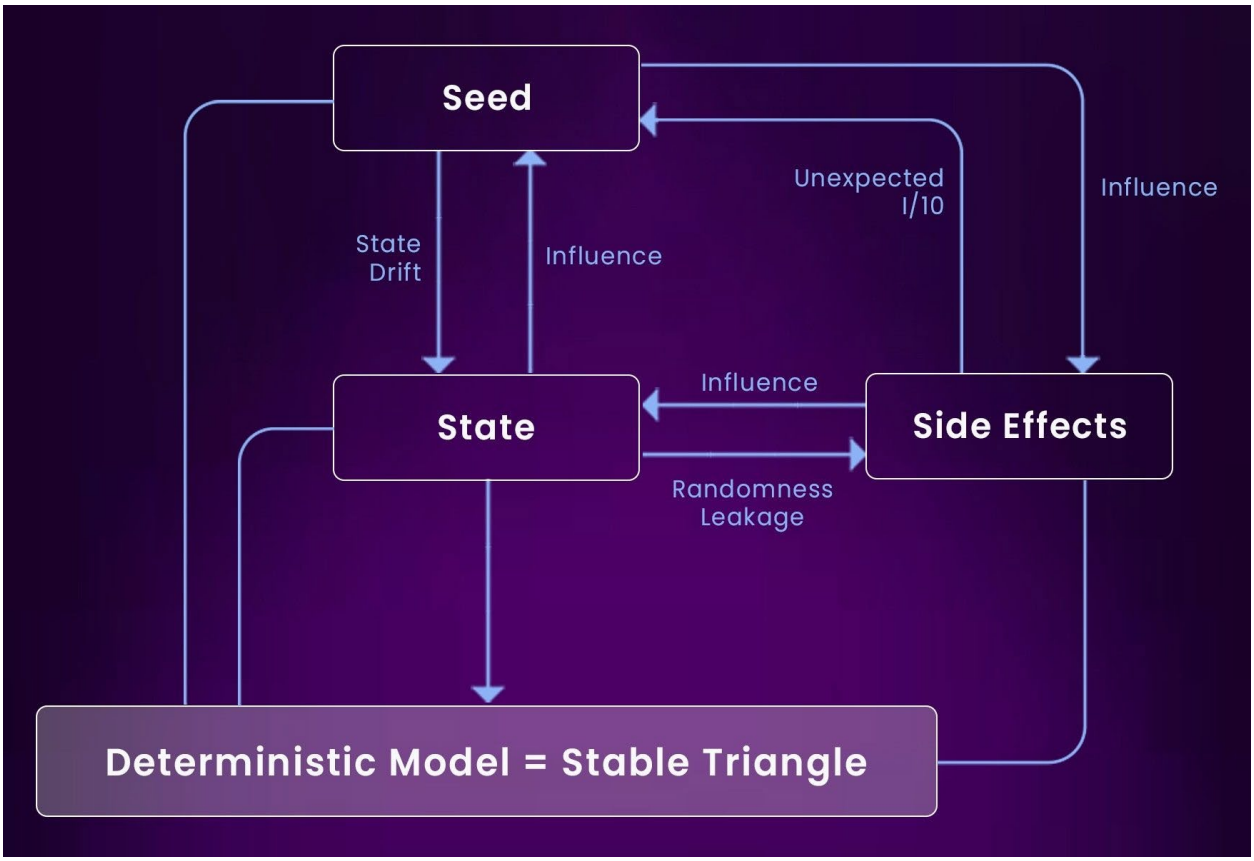
```
G(F) exists AND is structurally coherent
```

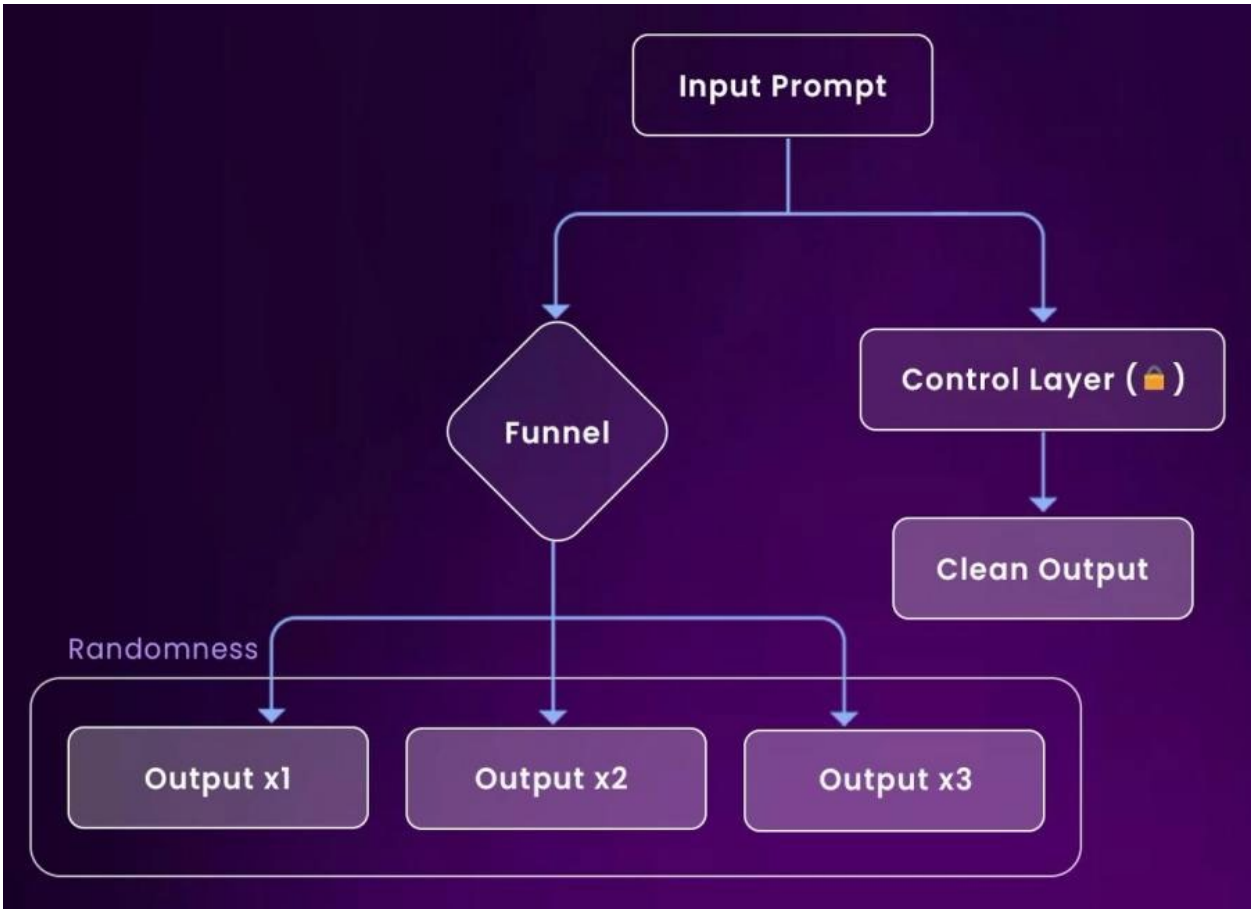
Governance = procedural.

Diagram 2 — SagaAI™ Executable Governance Control Path

(Deterministic Guard-Bound Architecture)




Seed
State
Side Effects
Deterministic Model = Stable Triangle
State Drift
Influence
Unexpected I/10
Influence
Influence
Randomness Leakage



Structural Layout (Left → Right Flow)

```
AI Function Invocation
        ↓
SagaAI™ Parser
        ↓
Load Canonical Class Tree
        ↓
Resolve Governance Object References (LOIDs)
        ↓
Guard Evaluation Layer
    • Risk Guard
    • Data Governance Guard
    • Logging Guard
    • Oversight Guard
    • Least-Privilege Guard
        ↓
Decision Node
   ↙        ↓        ↘
Abort     Escalate   Structurally Valid
                            ↓
                  Institutional Authorization
                            ↓
                         Commit
```

This flow is explicitly defined in Section 8 of the paper .

Architectural Characteristics

1. Governance Artifacts

Governance is encoded as:

- Canonical class objects
- Ledger Object IDs (LOIDs)
- Immutable provenance metadata
- Typed cross-domain references

2. Pre-Commit Structural Enforcement

Before commit:

- Referential closure verified
- No dangling references allowed
- Guard non-bypassability enforced (TLA+ modeled)
- Oversight flags validated
- Risk objects confirmed
- Control declarations verified

Validation occurs:
At method boundary.
Before state mutation.

3. Authority Model

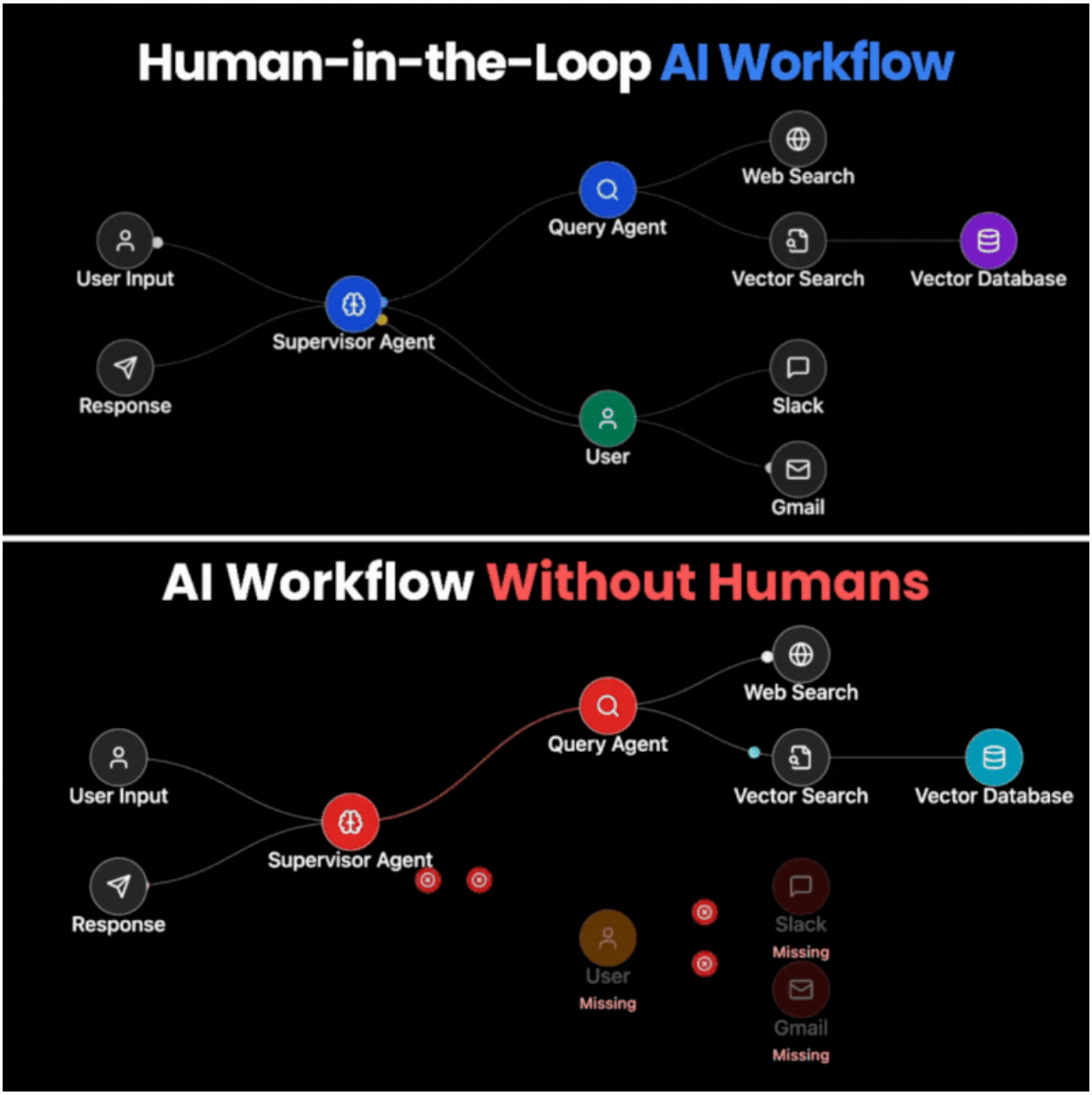
SagaAI™ does **not**:

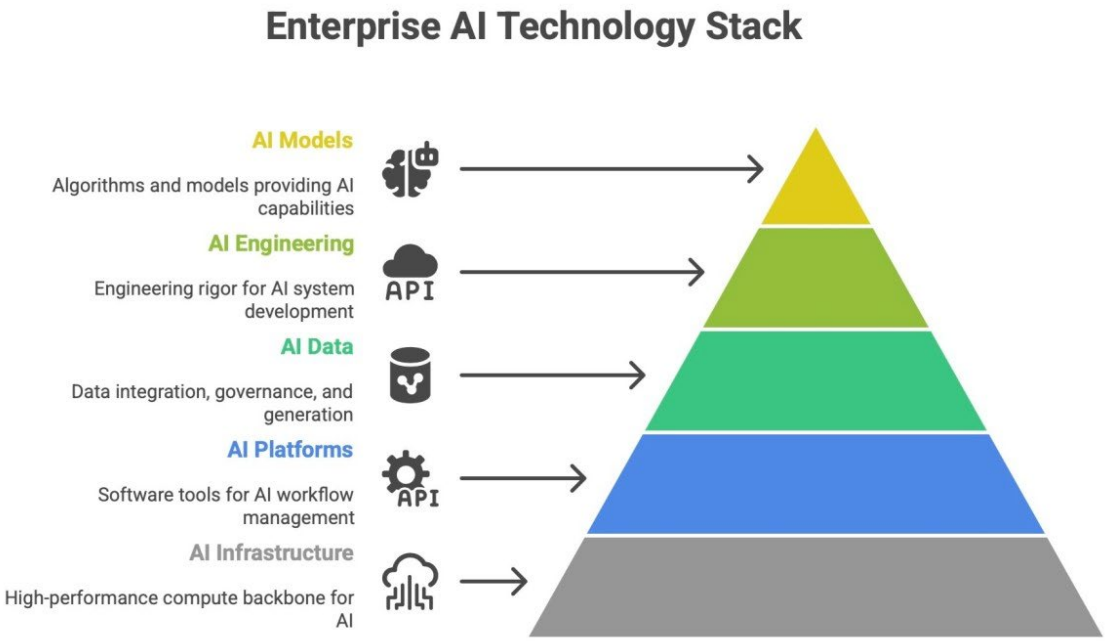
- Determine compliance
- Replace human authority
- Grant certification

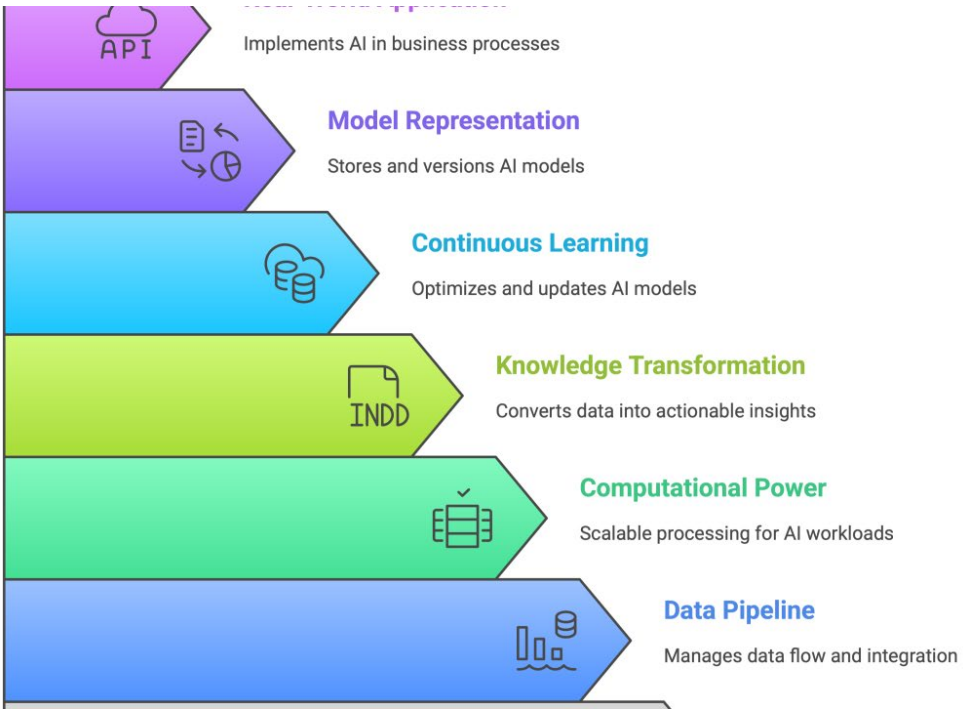
Formal boundary condition:

```
Commit(F) ⇒ Authorized(InstitutionalActor)
            ∧ StructurallyValid
```

Diagram 3 — Side-By-Side Control Boundary Comparison


Human-in-the-Loop AI Workflow
Web Search
Query Agent
User Input
Vector Search
Vector Database
Supervisor Agent
Response
Slack
User
Gmail
AI Workflow Without Humans
Web Search
Query Agent
User Input
Vector Search
Vector Database
Supervisor Agent
Response
Slack
Missing
User
Missing
Gmail
Missing





Control Boundary Location

HITL

```
Runtime → Output → Human Review → Commit
```

Governance evaluated after execution logic.

SagaAI™

```
Invocation → Guard Validation → Authorization → Commit
```

Governance validated before execution mutation.

Key Architectural Differences

Dimension	HITL	SagaAI™
Governance representation	Documents	Canonical objects
Identity continuity	Weak	Deterministic LOID
Validation timing	Post-threshold	Pre-commit
Guard non-bypassable	No	Yes
Referential closure enforced	No	Yes
Human authority preserved	Yes	Yes
Cross-framework binding	Manual	Typed object graph

Architectural Summary

HITL

- Procedural control
- Human-centric enforcement
- Episodic governance validation
- Document reconstruction model

SagaAI™

- Infrastructural control
- Deterministic structural binding
- Runtime guard injection
- Object lineage verification model

Core Distinction

HITL asks:

“Did a human approve this?”

SagaAI asks:

“Is governance structurally present, referentially coherent, and guard-validated before any commit occurs?”

And then still requires: Human Authorization

Appendix: Formal Control-Theoretic Model of Governance Architectures

The formal notation below corresponds to the informal model in Section 11 as follows: $\Psi(x_k, u_k)$denotes the structural validity function $V(F, G(F))$; $g_k$denotes the commit gate; $\mathcal{S}$denotes the set of governance-admissible states.

A.1 System Model

A.1.1 Plant

Let the operational system evolve in discrete time:

[
x_{k+1} = f(x_k, u_k, w_k)
]

where (x_k \in \mathcal{X}) is system state, (u_k \in \mathcal{U}) is a control input (any action that may mutate governance-relevant state), and (w_k \in \mathcal{W}) is a disturbance.

A.1.2 Governance Admissibility

Define a governance-admissible set of states:

[
\mathcal{S} \subseteq \mathcal{X}
]

where (\mathcal{S}) represents states consistent with required governance conditions (e.g., risk linkage present, logging active, oversight designated, authorization recorded, least privilege enforced).

To distinguish *unsafe and silent* from *unsafe but governed*, define a governance mode (m_k \in \mathcal{M}) with:

$$
\mathcal{M} := {\text{Normal}, \text{Escalate}, \text{Incident}, \text{Abort}}
$$

and joint state:

$$
z_k := (x_k, m_k) \in \mathcal{X} \times \mathcal{M}.
$$

Define a joint admissible set:

$$
\mathcal{S}_z \subseteq \mathcal{X} \times \mathcal{M}
$$

with the intended semantics:

- $((x,\text{Normal}) \in \mathcal{S}_z \iff x \in \mathcal{S})$
- $((x,\text{Escalate}) \in \mathcal{S}_z \iff)$ commit was blocked and escalation was recorded
- $((x,\text{Incident}) \in \mathcal{S}_z \iff)$ failure was logged and contained
- $((x,\text{Normal}) \notin \mathcal{S}_z)$ if an unsafe mutation occurred silently

This encodes the architectural distinction between *unsafe and silent* versus *unsafe but governed*.

A.1.3 Commit as a Gated Actuator

Separate committed persistent state (x_k) from pre-commit working state ($\tilde{x}_k$). The system computes a candidate next working state:

$$
\tilde{x}_{k+1} = \tilde{f}(x_k, u_k, w_k).
$$

A commit gate determines what persists:

$$
x_{k+1} =
\begin{cases}
\Pi(\tilde{x}_{k+1}) & \text{if } g_k = 1 \\
x_k & \text{if } g_k = 0
\end{cases}
$$

where (\Pi) projects working state into persistent state, and (g_k \in {0,1}) is the commit-enable signal.

A.2 Atomicity Preconditions

A.2.1 Atomic Commit Boundary

The “hard-gate” interpretation of (g_k) requires that **no externally visible side effects** occur when a commit is blocked. Let (\mathrm{Trace}(x_k,u_k,w_k)) denote the full execution trace of processing ((x_k,u_k,w_k)) up to the point the commit decision is finalized, including guard evaluation. Let (\mathrm{SE}(\cdot)) map a trace to its externally visible side effects (network calls, distributed lock acquisition, side-effecting reads, external service queries, durable writes, etc.).

Atomicity precondition (hard-gate):
[
g_k = 0 ;\Rightarrow; \mathrm{SE}(\mathrm{Trace}(x_k,u_k,w_k)) = 0.
]

This must include the guard phase: the guard cannot rely on external calls whose execution is itself a side effect.

A.2.2 Guard Locality / Purity Axiom (Implementation Constraint)

Let (\mathcal{C}(x_k)) be a **locally cached snapshot** containing all governance-relevant artifacts required for evaluation (e.g., LOID-resolved governance objects, local proofs/receipts, and local authorization artifacts).

Guard locality axiom:
[
\Psi(x_k,u_k) \equiv \Psi(\mathcal{C}(x_k),u_k)
]
and (\Psi) is evaluated without external calls and without side-effecting reads during evaluation.

Institutional authorization is represented as a **pre-attested local artifact** (e.g., cryptographic signature/receipt), not a live call to an authorization service, to preserve guard purity.

Remark. This axiom is an implementation property that must be verified. It is architecturally satisfiable in environments where governance artifacts referenced by LOID are locally resolvable inside the transaction boundary.

A.3 Observability Models

A.3.1 HITL Observation Channel

Let $y_k$denote observed governance evidence. Under HITL, missing evidence is modeled as structured absence:

$$y_k = h(x_k)\ \mathbf{1}\{A_k = 1\},$$

where $A_k \in \{0,1\}$indicates whether required governance evidence was generated and transmitted in time to be observable.

This is not additive noise: governance failures are systematic absences and omissions, not symmetric perturbations.

Load-Dependent Observability Degradation

Let $L_k$denote operational load (e.g., queue depth, incident volume, staffing constraint, time pressure). We model HITL governance channels as exhibiting load-dependent observability degradation, meaning:

$$\mathbb{P}(A_k = 0 \mid L_k = \ell) \text{ is nondecreasing in } \ell.$$

This assumption is consistent with empirical findings in human-factors and safety engineering literature, where error rates and omission rates increase under operational stress and workload [6], [7].

Decomposed Missed-Trigger Probability

Let the trigger be:

$$E_k := \mathbf{1}\{\phi(y_k) \geq \theta\},$$

where $\phi$is a detection/flagging function and $\theta$is a threshold. Then, by the law of total probability over the partition $\{A_k = 0, A_k = 1\}$:

$$\mathbb{P}(E_k = 0 \mid x_k \notin \mathcal{S}) = \mathbb{P}(A_k = 0 \mid x_k \notin \mathcal{S}) + \mathbb{P}(A_k = 1 \mid x_k \notin \mathcal{S}) \cdot \mathbb{P}(\phi(h(x_k)) < \theta \mid A_k = 1, x_k \notin \mathcal{S}).$$

HITL failure therefore arises from:

1. Evidence non-generation (often load-amplified), and
2. Trigger miss conditional on evidence presence.

These terms are generally dependent under high load.

A.3.2 SagaAI Observation at the Commit Boundary

Under the Guard Locality / Purity Axiom, governance-relevant artifacts are locally observable at guard evaluation time. In particular, the probability of missing required evidence due to cross-system absence satisfies the design objective:

$$
\mathbb{P}(A_k = 0) \to 0,
$$

conditional on correct implementation of governance object instantiation, local caching, and LOID-resolved retrieval. This observability advantage does not, by itself, imply guard soundness (Section A.5).

A.4 Governance Gates: Fail-Open vs. Fail-Closed

A.4.1 HITL Gate (Fail-Open / Default-Enable)

HITL inserts a delayed review on flagged events. Let $\pi(\cdot)$ be a human decision policy and τ be the review delay (time-varying and typically load-dependent). Then:

$$
g_k =
\begin{cases}
\pi(y_{k-\tau}) & \text{if } E_k = 1 \\
1 & \text{if } E_k = 0
\end{cases}
$$

This is a **default-enable** (fail-open) gate: commits proceed unless a trigger fires.

Key vulnerabilities (control interpretation):

- Delay τ creates a window in which the uncontrolled plant may evolve into unsafe states.
- When $E_k=0$ but $x_k\notin\mathcal{S}$, the gate defaults open.

A.4.2 SagaAI Gate (Fail-Closed / Default-Disable)

SagaAI-style governance evaluates structural sufficiency and institutional authorization prior to commit:

$$
g_k = \mathbf{1}{\Psi(x_k,u_k)=1}\cdot \mathbf{1}{\mathrm{Auth}(x_k,u_k)=1}.
$$

Explicit default:
$$
\Psi(x_k,u_k)=0 \Rightarrow g_k = 0.
$$

This is a **default-disable** (fail-closed) gate, evaluated prior to any persistent state mutation and under the Guard Locality / Purity Axiom. [13]

A.5 Forward Invariance and Guard Soundness

A.5.1 Design Axiom (Guard Soundness Under Bounded Disturbances)

Let $\mathcal{W}_B \subseteq \mathcal{W}$ denote bounded disturbances under which a soundness claim is made.

Guard soundness axiom:
$$
\Psi(x,u)=1 \Rightarrow f(x,u,w)\in \mathcal{S} \quad \forall w\in\mathcal{W}_B.
$$

This axiom is not derivable from guard structure alone when Ψ checks *structural proxies* (e.g., "risk object linked") rather than the full semantic safety condition. Soundness must be established via domain-specific calibration: either (a) a formal argument in a restricted domain where proxies imply outcomes, or (b) empirical validation against enumerated failure scenarios.

A.5.2 Two-Layer Disturbance Model (Transaction-Bound Tractability)

Under atomicity and guard purity, disturbances can be decomposed as:

$$
w_k = (w_k^{in}, w_k^{out}),
$$

where w_k^{in} are disturbances already present in the invocation payload or pre-state snapshot, and w_k^{out} are external disturbances occurring outside the commit boundary.

Atomicity implies w_k^{out} does not affect (f) **during the guarded transaction step**. Robust safety inside the commit boundary therefore reduces to:

[
\Psi(x,u)=1 \Rightarrow f(x,u,w^{in})\in\mathcal{S}.
]

This restriction makes robust reasoning tractable in transactionally isolated execution environments.

Proposition A.1 (Conditional Forward Invariance)

If (i) the Atomicity Preconditions hold, and (ii) the Guard Soundness Axiom holds for (\mathcal{W}_B), then:

[
x_0 \in \mathcal{S} \Rightarrow \forall k:\ x_k \in \mathcal{S}.
]

Proof sketch. By induction. Base case holds by assumption. For the step: if (g_k=0), atomicity implies (x_{k+1}=x_k\in\mathcal{S}). If (g_k=1), then (\Psi(x_k,u_k)=1), and guard soundness implies (f(x_k,u_k,w_k)\in\mathcal{S}) for all (w_k\in\mathcal{W}*B), hence (x*{k+1}\in\mathcal{S}). (\square)[8]

A.6 Specification Gap

A.6.1 Baseline and Robust Gaps

Baseline specification gap (zero disturbance):
[
\mathcal{G}={(x,u)\mid \Psi(x,u)=1 \ \wedge\ f(x,u,0)\notin\mathcal{S}}.
]

Robust specification gap:
[
\mathcal{G}_{rob}={(x,u)\mid \Psi(x,u)=1 \ \wedge\ \exists w\in\mathcal{W}_B:\ f(x,u,w)\notin\mathcal{S}}.
]

Note (\mathcal{G}\subseteq \mathcal{G}_{rob}). The baseline gap is a lower bound on true residual risk.

A.6.2 Specification Completeness and Residual Risk

Specification completeness: the guard is complete iff
[
\mathcal{G}_{rob}=\emptyset.
]

Residual risk under SagaAI:
[
P_{spec}=\mathbb{P}((x_k,u_k)\in\mathcal{G}_{rob}).
]

Interpretation: the dominant risk term shifts from HITL operational stochasticity and observability degradation to guard/specification completeness. Minimizing (P_{spec}) requires reducing (\mathcal{G}_{rob}) via guard refinement and/or shrinking (\mathcal{W}_B) via architectural constraints (e.g., strengthened atomicity and narrowed authorization scope).

A.7 Ramadge–Wonham Formalization

A.7.1 Event Alphabet and Controllability Partition

Let (\Sigma) be the event alphabet and partition:

[
\Sigma = \Sigma_c \cup \Sigma_{uc},
\quad
\Sigma_c \cap \Sigma_{uc}=\emptyset.
]

Example controllable events:
[
\Sigma_c := {\textsf{commit},\ \textsf{abort},\ \textsf{escalate},\ \textsf{incident_record}}.
]

Example uncontrollable events:
[
\Sigma_{uc} := {\textsf{disturb},\ \textsf{partition},\ \textsf{evidence_loss},\
\textsf{auth_compromise}}.
]

Including (\textsf{auth_compromise}\in\Sigma_{uc}) captures the policy-critical fact that compromised institutional authorization cannot be ruled out by technical supervision alone.[10]

A.7.2 Plant and Specification Languages

Let (G) be the plant automaton generating (L(G)\subseteq \Sigma^*). Let (K\subseteq L(G)) be the governance-admissible specification language.

A supervisor (S) restricts behavior to the closed-loop language (L(S/G)).

Controllability condition: (K) is controllable w.r.t. (L(G)) and (\Sigma_{uc}) if:

[
\overline{K}\ \cap\ L(G)\cdot \Sigma_{uc} \ \subseteq\ \overline{K},
]

where (\overline{K}) is the prefix closure of (K). If (K) is not controllable, define the supremal controllable sublanguage (\sup\mathcal{C}(K)) and implement a supervisor achieving:

[
L(S/G) = \sup\mathcal{C}(K).
]

The set (K \setminus \sup\mathcal{C}(K)) represents governance goals that cannot be enforced given the uncontrollable events—an explicit impossibility characterization.[13]

A.7.3 HITL as Partial-Observation Supervision

Let (P:\Sigma^*\to\Sigma_o^*) be an observation projection with observable subalphabet (\Sigma_o\subseteq\Sigma). Under HITL, evidence loss corresponds to events that are (i) uncontrollable and (ii) unobservable, inducing ambiguity in trace equivalence classes under (P).

A partial-observation supervisor (S_{po}) must choose control actions based only on (P(s)) for observed traces (s\in\Sigma^*), which generally requires conservative disabling to ensure safety under ambiguity.[11] , [12]

A.7.4 SagaAI as Full-Observation Supervision at the Commit Boundary

Under the Guard Locality / Purity Axiom, governance-relevant information required at commit time is locally observable for decision, corresponding to effectively full observation on commit-relevant events.

Proposition A.2 (Observability Advantage Under Fixed Plant and Specification)

Let Gbe a plant automaton, $K \subseteq L(G)$a specification language, and let sup $\mathcal{C}(K)$denote the supremal controllable sublanguage of $K$with respect to $L(G)$and Σ_{uc}.

Let:

- S_{full}be a full-observation supervisor achieving

$$L(S_{full}/G) = \text{sup } \mathcal{C}(K),$$

- S_{po}be a partial-observation supervisor under projection P.

Then:

$$L(S_{po}/G) \subseteq \sup\ \mathcal{C}(K),$$

with strict containment whenever partial observation induces ambiguity between safe and unsafe continuations.

Updated proof sketch

Under full observation, standard supervisory control synthesis yields the maximally permissive safe supervisor achieving $\sup\ \mathcal{C}(K)$. Under partial observation, control decisions must be uniform over observational equivalence classes induced by P. If an equivalence class contains both (i) traces after which enabling a controllable event preserves $\sup\ \mathcal{C}(K)$and (ii) traces after which enabling the same event would violate safety, the supervisor must disable the event for the entire class. This strictly reduces permissiveness whenever such ambiguity exists. □

A.7.5 Worked Finite-State Example (Observability Gap Under Partial Observation)

This example instantiates the abstract supervisory control discussion with a finite-state plant and computes the closed-loop language under full and partial observation.

A.7.5.1 Plant Automaton (G)

Let

[
G = (Q,\Sigma,\delta,q_0)
]

with states:

[
Q = {q_0,q_1,q_2,q_3,q_4,q_5,q_6}
]

interpreted as:

- (q_0): idle
- (q_1): invocation started
- (q_2): governance evidence complete
- (q_3): governance evidence missing
- (q_4): authorization artifact present
- (q_5): committed

- q_6): incident recorded / abort

Event Alphabet

$$
\Sigma = \{\textsf{invoke},\ \textsf{evidence\_ok},\ \textsf{evidence\_loss},\ \textsf{auth},\ \textsf{commit},\ \textsf{abort},\ \textsf{incident\_record}\}
$$

Controllability Partition

$$
\Sigma_c = \{\textsf{commit},\ \textsf{abort},\ \textsf{incident\_record}\},
\quad
\Sigma_{uc} = \{\textsf{invoke},\ \textsf{evidence\_ok},\ \textsf{evidence\_loss},\ \textsf{auth}\}.
$$

Transition Function

$$
\delta(q_0,\textsf{invoke}) = q_1
$$

$$
\delta(q_1,\textsf{evidence\_ok}) = q_2,
\quad
\delta(q_1,\textsf{evidence\_loss}) = q_3
$$

$$
\delta(q_2,\textsf{auth}) = q_4,
\quad
\delta(q_3,\textsf{auth}) = q_4
$$

$$
\delta(q_4,\textsf{commit}) = q_5
$$

$$
\delta(q_4,\textsf{abort}) = q_6,
\quad
$$

$$
\delta(q_4,\textsf{incident\_record}) = q_6
$$

Modeling Note (Justification of $(q_3 \to q_4)$)

Authorization is modeled as arriving independently of evidence completeness:

$$
\delta(q_3,\textsf{auth}) = q_4.
$$

This reflects the realistic governance scenario in which an institutional signatory may authorize an action without visibility into the internal evidence pipeline state. The plant therefore permits authorization to occur even when evidence is missing; it is the supervisor's role to prevent unsafe commits.

A.7.5.2 Probabilistic Interpretation (Connection to Section A.3)

The transition

$$
q_1 \xrightarrow{\textsf{evidence\_loss}} q_3
$$

instantiates the structured missingness model of Section A.3.1.

Let L_k denote operational load. We annotate:

$$
\mathbb{P}(q_1 \to q_3 \mid L_k=\ell)

=

\mathbb{P}(A_k = 0 \mid L_k=\ell),
$$

where A_k is the evidence-generation indicator defined in Section A.3.

Under the load-dependent observability degradation assumption:

$$
\mathbb{P}(A_k = 0 \mid L_k=\ell)
\text{ is nondecreasing in } \ell.
$$

Thus, increasing load increases the probability of entering state (q_3), directly connecting the probabilistic observation model (A.3) to this deterministic DES abstraction.

A.7.5.3 Governance Specification (K)

Specification requirement:

- Commit is allowed only if evidence was complete.
- If evidence was missing, the system must abort or record an incident.

Formally:

[

S_{safe}:=invoke evidence_ok auth commit∈K

[

[
S_{unsafe}:=invoke evidence_loss auth commit∉K

]

Governed alternatives:

[
\textsf{invoke}\ \textsf{evidence_loss}\ \textsf{auth}\ \textsf{abort}
\in K,
]

[
\textsf{invoke}\ \textsf{evidence_loss}\ \textsf{auth}\ \textsf{incident_record}
\in K.
]

A.7.5.4 Verification of Controllability

We verify that (K) is controllable with respect to (L(G)) and (\Sigma_{uc}).

Controllability requires:

[
\overline{K} \cap L(G)\cdot \Sigma_{uc}
\subseteq
\overline{K}.
]

The uncontrollable events are

[
\Sigma_{uc} =
{\textsf{invoke},\textsf{evidence_ok},\textsf{evidence_loss},\textsf{auth}}.
]

None of these events constitutes a commit.

Therefore:

- Any trace in (\overline{K}) extended by an uncontrollable event remains a prefix of a trace that has not yet committed.
- The only violation occurs through (\textsf{commit}), which is controllable.

Hence the controllability condition holds, and:

[
\sup\mathcal{C}(K) = K.
]
[13]

A.7.5.5 Full-Observation Supervisor

Under full observation, the supervisor can distinguish whether state (q_4) was reached via:

- (q_1 \to q_2 \to q_4) (evidence complete), or
- (q_1 \to q_3 \to q_4) (evidence missing).

Thus:

- Enable (\textsf{commit}) only if the prior transition was (\textsf{evidence_ok}).
- Otherwise disable (\textsf{commit}) and enable (\textsf{abort}) or (\textsf{incident_record}).

Since (K) is controllable:

[
L(S_{full}/G) = \sup\mathcal{C}(K) = K.
]

A.7.5.6 Partial-Observation Supervisor

Let observable alphabet:

$$
\Sigma_o = {\textsf{invoke},\textsf{auth},\textsf{commit},\textsf{abort},\textsf{incident_record}}.
$$

Unobservable:

$$
\textsf{evidence_ok},\ \textsf{evidence_loss}.
$$

Then:

$$
P(s_{safe})

\textsf{invoke}\ \textsf{auth}\ \textsf{commit},
$$

$$
P(s_{unsafe})

\textsf{invoke}\ \textsf{auth}\ \textsf{commit}.
$$

The two traces are observationally indistinguishable.

To prevent (s_{unsafe}), the supervisor must disable (\textsf{commit}) after observing:

$$
\textsf{invoke}\ \textsf{auth}.
$$

This disables both safe and unsafe continuations.

Thus:

$$
L(S_{po}/G)
\subset
K

L(S_{full}/G).
$$

The containment is strict.

A.7.5.7 Interpretation

This example concretely demonstrates Proposition A.2:

- Under full observation (SagaAI-style commit-boundary evaluation), the maximally permissive safe language (K) is achievable.
- Under partial observation (HITL-style governance), ambiguity forces conservative disabling of safe actions.

The strict gap:

[
K \setminus L(S_{po}/G)
]

is the formal cost of observability degradation.

Additionally, the transition probability into (q_3) increases with operational load, directly instantiating the structured missingness model of Section A.3 within the DES framework.

Section A.7.6 extends the finite-state example by introducing an uncontrollable auth_compromiseevent. This extension computes sup $\mathcal{C}(K')$and formally identifies the boundary between technical supervisory guarantees and institutional controls. To keep the example tractable, one additional controllable event proceedis introduced, representing the system's decision to enter the authorization-critical phase. This is architecturally realistic: a fail-closed runtime can always refuse to proceed before entering a region where compromised authorization could force an unsafe commit.[10–13]

A.7.6.1 Extended Plant G'

Let

$$G' = (Q', \Sigma', \delta', q_0)$$

with states:

$$Q' = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6\}$$

with the same interpretations as in Section A.7.5:

- q_0: idle
- q_1: invocation started

- q_2: evidence complete
- q_3: evidence missing
- q_4: authorization artifact present
- q_5: committed
- q_6: incident recorded / abort

Event Alphabet

Σ'
$=$ {invoke, proceed, evidence_ok, evidence_loss, auth, auth_compromise, commit, abort, incident_record}.

Controllability Partition

$$\Sigma_c = \{\text{proceed, commit, abort, incident\_record}\}, \Sigma_{uc} = \{\text{invoke, evidence\_ok, evidence\_loss, auth, auth\_compromise}\}.$$

Here, auth_compromiseis explicitly modeled as uncontrollable.

A.7.6.2 Transition Function

Invocation:

$$\delta'(q_0,\text{invoke}) = q_1.$$

Proceed into evidence evaluation (controllable gate):

$$\delta'(q_1,\text{proceed}) = q_1.$$

Evidence outcomes (uncontrollable):

$$\delta'(q_1,\text{evidence\_ok}) = q_2, \delta'(q_1,\text{evidence\_loss}) = q_3.$$

Authorization arrival (uncontrollable):

$$\delta'(q_2,\text{auth}) = q_4, \delta'(q_3,\text{auth}) = q_4.$$

Authorization compromise (uncontrollable):

$$\delta'(q_4,\text{auth\_compromise}) = q_5.$$

Ordinary commit and governed handling (controllable):

$$\delta'(q_4,\text{commit}) = q_5,$$
$$\delta'(q_4,\text{abort}) = q_6, \delta'(q_4,\text{incident\_record}) = q_6.$$

Modeling Note (Authorization Compromise)

The transition

$$q_4 \xrightarrow{\text{auth\_compromise}} q_5$$

models the possibility that an institutional authorization channel is compromised (e.g., coerced approval, fraudulent signature, malware on a signing endpoint). This event is uncontrollable by the supervisor and may force a committed outcome independent of the supervisor's intent.

This models the real-world fact that technical enforcement cannot prevent all failures of institutional integrity.

The self-loop reflects that proceed is a permission event rather than a state-advancing action; its role is solely as a controllable event the supervisor can disable to prevent entry into the authorization-critical region.

A.7.6.3 Specification K'

We define:

$$K' = \{s \in L(G') \mid s \text{ contains no occurrence of auth\_compromise}\}.$$

This encodes the requirement:

No committed trace may arise from compromised authorization.

A.7.6.4 Verification That K' Is Not Controllable

Controllability requires:

$$\bar{K'} \cap L(G') \cdot \Sigma_{uc} \subseteq \bar{K'}.$$

Consider the trace:

$$s = \text{invoke proceed evidence\_ok auth.}$$

This trace:

- Is feasible in G',
- Contains no auth_compromise,
- Therefore $s \in \bar{K'} \cap L(G')$.

From state $q_4$reached by s, the uncontrollable event auth_compromisemay occur:

$$s \cdot \text{auth\_compromise} \in L(G'),$$

but:

$$s \cdot \text{auth\_compromise} \notin \bar{K'}.$$

Therefore:

$$\bar{K'} \cap L(G') \cdot \Sigma_{uc} \nsubseteq \bar{K'},$$

and $K'$is not controllable.

A.7.6.5 Computation of sup $\mathcal{C}(K')$

We compute the supremal controllable sublanguage.

Key Observation

Any trace that reaches state $q_4$is unsafe with respect to controllability, because from $q_4$the uncontrollable event auth_compromisecan force a transition violating K'.

Therefore, any controllable sublanguage must exclude all traces reaching q_4.

Reaching $q_4$requires enabling proceed. Since proceedis controllable, the supervisor can prevent entry into this region by disabling proceed.

Supremal Controllable Sublanguage

Define:

$$L_{\neg\text{proceed}} = \{s \in L(G') \mid \text{proceed} \notin s\}.$$

Under this restriction:

- The plant remains in $q_1$after invoke,
- Evidence and authorization transitions never occur,
- auth_compromisecannot occur.

Allowing governed handling at q_1:

$$\delta'(q_1,\text{abort}) = q_6, \delta'(q_1,\text{incident\_record}) = q_6,$$

the supremal controllable sublanguage is:

$$\sup\ \mathcal{C}(K') = \{\epsilon\} \cup \{\text{invoke}\} \cup \{\text{invoke abort,}\ \text{invoke incident\_record}\}.$$

Maximality Argument

Any language $H \subseteq K'$containing a trace that reaches $q_4$is not controllable, since such traces admit an uncontrollable extension via auth_compromisethat violates K'.

Thus, every controllable sublanguage of K'must exclude all traces reaching q_4. Disabling proceedremoves exactly those traces and no others. Therefore the language above is maximal and equals $\sup\ \mathcal{C}(K')$.

A.7.6.6 Interpretation

This example formally proves:

- When authorization compromise is modeled as an uncontrollable event capable of forcing a commit,
- The specification “no compromised authorization commits” is not controllable.
- The only enforceable sublanguage is fail-closed refusal to proceed (plus governed abort/incident handling).

This makes precise the boundary identified in Section A.9:

Technical supervision cannot guarantee safety against compromised institutional authorization; such risks must be mitigated by institutional controls (audit, separation of duties, legal accountability, cryptographic deterrence).

A.7.6.7 Structural Insight

Compare Sections A.7.5 and A.7.6:

- In A.7.5, safety violations arise through controllable events; maximal safe permissiveness is achievable under full observation.
- In A.7.6, safety violations arise through uncontrollable events; maximal permissiveness collapses to fail-closed minimal behavior.

The distinction between “unobservable” and “uncontrollable” events is therefore not merely technical — it determines whether maximal safe enforcement is achievable at all.

A.7.6.8 Architectural Implication

The result of Section A.7.6 clarifies the precise boundary between the SagaAI commit-boundary supervisor and institutional authorization mechanisms discussed in the main architecture. The SagaAI guard enforces structural preconditions (risk linkage, oversight designation, logging activation, least-privilege constraints) at the commit boundary and, under full observation, can achieve maximal safe permissiveness when violations arise only through controllable events (Section A.7.5). However, when authorization compromise is modeled as an uncontrollable event capable of forcing a committed transition (Section A.7.6), the enforceable language collapses to the supremal controllable sublanguage (\sup\mathcal{C}(K')), which excludes all behaviors entering the authorization-critical region. This formally demonstrates that technical supervision guarantees safety only up to the boundary of uncontrollable institutional integrity failures. Beyond that boundary, safety depends on external institutional controls — auditability, separation of duties, cryptographic accountability, and legal enforcement — rather than supervisory control alone. [10–13]

A.8 Risk Decomposition Summary

Architecture	Dominant risk term	Default	Observation
HITL	(\mathbb{P}(\text{missed trigger}) + \mathbb{P}(\text{delay breach}))	Fail-open	Partial, load-degraded
SagaAI	(P_{spec}=\mathbb{P}((x_k,u_k)\in\mathcal{G}_{rob}))	Fail-closed	Full (commit-boundary), conditional on purity/locality

This architecture does not claim to eliminate risk. It claims to shift the dominant risk source from operational stochasticity and observability degradation to specification completeness—a target that is enumerable, testable, and reducible through systematic guard refinement and architectural constraints.

A.9 Authorization Boundary (Impossibility Result)

The SagaAI gate includes the factor

$$\mathbf{1}\{\mathrm{Auth}(x_k, u_k) = 1\}$$

(pre-attested local artifact).

If authorization compromise is modeled as an uncontrollable event

$$\text{auth\_compromise} \in \Sigma_{uc},$$

then there exist governance specifications K for which controllability fails: no purely technical supervisor can guarantee safety against compromised institutional authorization.

In such cases, the supremal controllable sublanguage

$$\sup\ \mathcal{C}(K)$$

characterizes the enforceable portion of the policy, and the complement

$$K \setminus \sup\ \mathcal{C}(K)$$

must be addressed by external institutional controls (e.g., separation of duties, audit, legal accountability, recourse mechanisms, and/or cryptographic deterrence mechanisms).[10] , [13]

Constructive Note

A concrete example arises whenever the specification requires

$$\mathrm{Auth}(x_k, u_k) = 1$$

as a necessary condition for commit, and

$$\text{auth\_compromise} \in \Sigma_{uc}$$

permits a transition that produces a committed trace despite invalid authorization.

In such a plant, the uncontrollable event auth_compromisecan generate traces violating K, rendering Kuncontrollable.

The worked finite-state example in Section A.7.6 explicitly demonstrates this construction by adding auth_compromisetransitions and computing the resulting sup $\mathcal{C}(K)$.

End of Appendix

References

[1] ISO/IEC. *ISO/IEC 42001:2023 — Information technology — Artificial intelligence — Management system.* International Organization for Standardization, 2023. (ISO)

[2] Tabassi, E., et al. *Artificial Intelligence Risk Management Framework (AI RMF 1.0).* NIST AI 100-1, National Institute of Standards and Technology, January 2023. (NIST Publications)

[3] European Parliament and Council. *Regulation (EU) 2024/1689 (Artificial Intelligence Act), Official Journal version (13 June 2024).* European Union, 2024. (EUR-Lex)

[4] PraSaga Foundation. *Executable Governance: Operationalizing ISO/IEC 42001:2023, NIST AI RMF, NIST Cyber AI Profile and the EU AI Act as Interoperable Class Architecture, Implemented on SagaChain.* Governance White Paper, February 19, 2026. (Internal / project publication.)

[5] Lamport, L. *Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers.* Addison-Wesley, 2003. (Microsoft)

[6] Reason, J. *Human Error.* Cambridge University Press, 1990.

[7] Leveson, N. *Engineering a Safer World: Systems Thinking Applied to Safety.* MIT Press, 2011.

[8] Blanchini, F. "Set invariance in control." *Automatica*, 35(11):1747–1767, 1999. (ScienceDirect)

[9] Sipahi, R., Niculescu, S.-I., Abdallah, C. T., Michiels, W., and Gu, K. "Stability and Stabilization of Systems with Time Delay: Limitations and Opportunities." *IEEE Control Systems Magazine*, 31(1):38–65, 2011. (ResearchGate)

[10] Ramadge, P. J., and Wonham, W. M. "Supervisory control of a class of discrete event processes." *SIAM Journal on Control and Optimization*, 25(1):206–230, 1987. (SIAM E-Books)

[11] Cieslak, R., Desclaux, C., Fawaz, A. S., and Varaiya, P. "Supervisory control of discrete-event processes with partial observations." *IEEE Transactions on Automatic Control*, 33(3):249–260, 1988. (Academia)

[12] Lin, F., and Wonham, W. M. “On observability of discrete-event systems.” *Information Sciences*, 44(3):173–198, 1988. (ScienceDirect)

[13] Cassandras, C. G., and Lafortune, S. *Introduction to Discrete Event Systems.* 2nd ed., Springer, 2008. (Google Books)