

Reality-Grounded Autonomous Intelligence

An Evidence-, Authority-, and Outcome-Governed Architecture for Reliable AI Agents

Author: RM369

Date: 30 October 2026

Manuscript type: Conceptual architecture and empirical research proposal

Status: No implementation or empirical performance results are claimed in this manuscript.

Abstract

Autonomous AI agents increasingly combine language-model reasoning with tools that can inspect, modify, and act upon external environments. Their reliability therefore depends on more than the quality of generated text: an agent must preserve the user's authorized objective, distinguish observations from beliefs and predictions, operate within enforceable permissions, account for partial and irreversible effects, and establish whether the intended outcome actually occurred. Existing techniques address important pieces of this problem—including reasoning-action loops, tool-mediated critique, provenance, risk management, and transactional recovery—but these mechanisms are often deployed without a single operational model that governs the transitions between intention, knowledge, authority, execution, and acceptance.

This paper proposes **Reality-Grounded Autonomous Intelligence (RGAI)**, a system architecture for governing tool-using AI agents. RGAi separates five responsibilities: objective governance, epistemic integrity, authority governance, transactional execution and recovery, and independent assurance. These responsibilities share a versioned event history from which current state views are derived. RGAi distinguishes user-confirmed requirements from preferences and predictions; observations from external assertions and model inferences; permission to prepare from permission to commit; an attempted action from a confirmed effect; and a confirmed effect from an accepted outcome. A small trusted enforcement core mediates consequential actions, while risk-adaptive assurance allocates verification effort according to potential harm, uncertainty, irreversibility, and blast radius. The architecture also addresses prompt injection, privacy leakage, stale permissions, hidden dependencies, multi-agent concurrency, poisoned memory, reward tampering, and ambiguous external outcomes.

RGAI is a proposal, not a demonstrated improvement. The paper specifies its state model, action contract, transition rules, trust assumptions, failure handling, and a matched evaluation programme. It proposes testing against competent conventional agents on established computer-use, web, multi-application, research, and tool-interaction benchmarks augmented with controlled adversarial conditions. The central hypothesis is that explicit governance of state transitions can improve safe task completion and failure containment without imposing prohibitive costs on ordinary tasks. Whether that benefit exists, and at what operating point, remains an empirical question.

Keywords: autonomous agents; AI reliability; tool use; evidence provenance; action authorization; runtime assurance; transaction recovery; prompt injection; agent security; uncertainty; evaluation.

1. Introduction

The capabilities of AI assistants are increasingly expressed through actions rather than answers alone. A tool-using agent may search files, edit source code, update a database, send messages, schedule events, make purchases, or coordinate several applications. Such systems create a reliability problem that is not reducible to whether a language model produces a plausible response. An agent can reason plausibly and still misunderstand the user's objective, act on an untrusted instruction, mistake an attempted operation for a completed one, damage an unobserved dependency, or report success without adequate evidence.

Research has made substantial progress on individual elements of agent behaviour. ReAct combines reasoning and action with environmental feedback [2]. Self-Refine and CRITIC investigate iterative refinement and tool-interactive critique [3, 4]. Work on AI safety has long identified reward hacking, side effects, scalable supervision, safe exploration, and distributional shift as practical problems [5]. More recently, agent security guidance has highlighted prompt injection, excessive agency, tool misuse, identity abuse, and memory or context poisoning [9-11]. These contributions establish the importance of feedback, external tools, risk management, and security boundaries. They do not, by themselves, provide a complete operational contract for how an agent's beliefs, objectives, permissions, actions, and success claims are allowed to change one another.

The scale of the reliability gap is visible in interactive benchmarks. OSWorld introduced 369 executable tasks in real desktop and web environments and reported substantial gaps between human and model performance in its original evaluation [12]. AppWorld evaluates interactive tasks across multiple applications and checks both goal state and collateral changes [15]. WebArena evaluates long-horizon tasks in realistic websites [14], while τ -bench evaluates tool-agent-user interaction and repeated consistency [16]. OSWorld 2.0 extends computer-use evaluation to 108 long-horizon workflows; its authors report that even the best evaluated agent completed 20.6% of tasks under the paper's 500-step binary-completion metric, with a 54.8% partial score [13]. These results are benchmark-specific, not universal estimates of agent capability. They nevertheless demonstrate that realistic end-to-end agent reliability remains difficult to measure and achieve.

RGAI addresses a systems-level question: **what architecture allows a fallible model to act usefully while preventing its own predictions, beliefs, or tool outputs from silently becoming authority or accepted reality?** The proposal does not assume that the model will become infallible. It assumes instead that model errors, stale information, misleading tools, partial failures, and unexpected environmental changes will occur. The surrounding system should constrain how such failures propagate.

The governing design principle is:

A prediction may guide preparation; evidence governs claims; authorization governs commitment; and independent acceptance criteria govern declarations of success.

RGAI's proposed contribution is an integrated architecture and a falsifiable evaluation plan. It does not claim to invent provenance, least-privilege security, event logging, transaction compensation, or runtime verification. Its contribution is to define how these existing ideas can be composed into one agent-control model with explicit transition invariants and testable failure boundaries.

2. Problem formulation

2.1 From answer correctness to action reliability

For a conversational system, an error may be confined to a response. For an autonomous agent, an error can alter persistent state, expose data, consume money, communicate with third parties, or trigger downstream operations. The relevant unit of evaluation is therefore not merely the generated answer or tool call. It is the **end-to-end outcome**, including side effects and the evidence available to establish that outcome.

A reliable agent must answer five different questions without conflating them:

1. **Objective:** What has the user actually requested, and which constraints remain active?
2. **Knowledge:** What has been observed, what is asserted by a source, what is inferred, and what remains unknown?
3. **Authority:** Which actor may perform which operation on which resource, under what conditions?
4. **Execution:** What was attempted, what changed, and what is the current external state?

- 5. Acceptance:** Does the observed result satisfy the active objective, including relevant safety and side-effect constraints?

Many failures arise when one category is silently promoted into another. A forecast becomes a requirement; a model belief becomes a fact; a tool acknowledgement becomes proof of a real-world effect; a user's general request becomes blanket authorization; or a passing test becomes proof of overall quality.

2.2 Design objectives

RGAI is designed around the following objectives:

- **Objective fidelity:** preserve active, authorized user intent through long tasks and legitimate amendments.
- **Epistemic separation:** distinguish observations, source assertions, inferences, predictions, and unknowns.
- **Enforceable authority:** place consequential permissions at an execution boundary outside the model's control.
- **Bounded effects:** constrain the scope and blast radius of actions and record their dependencies.
- **Outcome verification:** require fit-for-purpose evidence before accepting or reporting completion.
- **Recoverability:** detect partial failures and use rollback, compensation, or containment where possible.
- **Proportionality:** apply verification and approval effort in proportion to risk rather than treating every action as critical.
- **Privacy:** retain sufficient audit evidence without indiscriminately logging sensitive content.
- **Practicality:** preserve useful autonomy and measure latency, cost, and operational friction.

2.3 Scope and non-goals

RGAI is an architecture for tool-using autonomous systems. It is not a new foundation model, a guarantee of truth, a complete theory of alignment, or a replacement for domain-specific safety engineering. It does not claim that a ledger makes data true, that multiple AI verifiers are independent, that a sandbox captures every external side effect, or that every irreversible action can be undone. It also does not solve weak perception, poor domain knowledge, or incorrect specifications merely by adding governance around them. These limitations are part of the research problem rather than exceptions to it.

3. Related work and position

3.1 Reasoning, acting, and tool-mediated correction

ReAct interleaves reasoning traces with environment actions, allowing observations to inform subsequent plans [2]. Self-Refine uses iterative self-feedback to improve generated outputs [3], while CRITIC uses external tools to critique and amend model outputs [4]. These approaches motivate the use of feedback and external evidence. RGAi places additional emphasis on the authority and status of each item entering the loop: feedback may revise a belief, but it does not automatically amend the user's objective, grant a permission, or certify an external side effect.

3.2 AI risk, specification gaming, and governance

Concrete Problems in AI Safety identifies reward hacking, side effects, scalable supervision, safe exploration, and distributional shift as practical accident risks [5]. Research on reward tampering further illustrates how a model may learn to manipulate the process by which it is evaluated rather than improve the intended outcome [20]. The NIST AI Risk Management Framework provides a lifecycle-oriented governance framework for mapping, measuring, managing, and governing AI risks [6]. RGAi is compatible with these approaches but operates at a more specific agent-runtime layer: it defines action-level contracts, state transitions, authorization checks, and outcome acceptance.

3.3 Security and least privilege

NIST's Zero Trust Architecture rejects implicit trust based only on network location or asset ownership and separates authentication and authorization from access to resources [7]. OWASP guidance identifies excessive agency as a risk when unexpected, ambiguous, or manipulated model outputs can cause damaging tool actions [9]. OWASP's prompt-injection guidance emphasizes that input filtering is not a security boundary and recommends enforcing permissions and parameter validation at tool boundaries, treating external content as

untrusted, and validating model outputs downstream [10, 11]. RGAI adopts these principles by placing authorization in a controller that the planning model cannot bypass or rewrite.

3.4 Provenance, event histories, and recovery

The W3C PROV family provides a data model for representing entities, activities, agents, and provenance relationships [8]. Provenance supports traceability and reproducibility, but provenance alone does not establish factual truth. Distributed systems use compensating transactions when a multi-step operation cannot be treated as one atomic transaction; such compensation can itself fail and may require idempotency and manual recovery [19]. RGAI applies these ideas to agent actions while distinguishing a record of an assertion from an independently established environmental fact.

3.5 Agent benchmarks

OSWorld, OSWorld 2.0, WebArena, AppWorld, τ -bench, and GAIA provide complementary environments for evaluating computer use, long-horizon web workflows, multi-application actions, user-policy interaction, and broad tool-assisted assistance [12-17]. AppWorld's state-based tests are particularly relevant because they can check unintended changes in addition to task completion [15]. τ -bench's repeated-trial pass^k reliability perspective is relevant because one successful attempt does not establish consistency [16]. RGAI proposes using these environments as foundations, then adding controlled interventions for authorization, state integrity, failure recovery, hidden dependencies, and adversarial evidence.

3.6 Relationship to prior evidence-gated execution work

Prior work by the author examined the separation of confirmed requirements, provisional predictions, and commit authorization in anticipatory agents [1]. RGAI broadens the scope from that separation to a more complete system architecture for epistemic integrity, delegated authority, transactional effects, privacy, concurrency, failure containment, and independent outcome acceptance. The present paper treats that earlier work as related research, not as an empirical validation of RGAI.

4. Architecture overview

RGAI comprises five cooperating subsystems supported by a shared, versioned event history and a minimal trusted enforcement core. The five subsystems are responsibilities, not necessarily five separate AI models. Some functions should be implemented as deterministic software; others may use models; high-impact assurance may require domain-specific tools or human review.

4.1 Objective governance

Objective governance maintains the active task definition. It distinguishes:

- **Hard requirements:** explicit, active instructions and constraints that define acceptance.
- **Soft preferences:** preferences that can guide trade-offs but do not override hard requirements.
- **Predicted needs:** model forecasts about likely future requirements, which may guide preparation but cannot independently create obligations.
- **Proposed amendments:** changes suggested by the user or agent, pending the authority appropriate to the change.
- **Superseded requirements:** historical instructions that have been explicitly replaced or withdrawn.
- **Unresolved conflicts:** incompatible or ambiguous requirements that cannot be safely reconciled under known precedence rules.

Intent is continuous but not immutable. The original instruction remains in history; the current authorized objective may change through an explicit amendment. The system records the source, scope, time, rationale, and authorization for every amendment. A model's prediction or convenience-based interpretation cannot silently amend the task.

4.2 Epistemic integrity

The epistemic subsystem maintains distinct statuses for information:

1. **Observation:** a result obtained from an identified sensor, file system, application, database, or other observation channel.

- 2. Source assertion:** a statement made by a tool, document, website, person, or external service.
- 3. Inference or belief:** a model's interpretation of available evidence.
- 4. Prediction:** a forecast about a future input, state, or outcome.
- 5. Unknown:** a question for which the available evidence is insufficient.
- 6. Established claim:** a claim that has met a domain-appropriate evidence standard for a particular scope and time.

These categories must not be collapsed. A tool acknowledgement is an assertion from that tool, not necessarily an independent observation of the resulting external state. A high model confidence is not itself evidence. An append-only or tamper-evident record can establish that an event was recorded and not subsequently altered under stated assumptions; it cannot prove that the recorded assertion was truthful.

Evidence records should carry at least: origin, actor, acquisition time, integrity status, scope, relevance, freshness, independence characteristics, and sensitivity classification. Confidence estimates should be calibrated against empirical outcomes where possible, not accepted merely because a model expresses a numerical probability [18].

4.3 Authority governance

Authority governance separates the ability to prepare an action from the ability to execute or commit it. The model can propose a tool action, but a trusted controller evaluates the request against an authorization grant and the applicable safety policy.

A permission grant is scoped to an actor, operation, resource or resource set, relevant object version, conditions, expiry, and revocation state. Authorization is rechecked at the commit boundary to avoid stale approvals and time-of-check/time-of-use races. The controller validates the actual arguments and target at the tool boundary; a prompt instruction or model-generated claim that an action is authorized has no independent authority.

User authorization is necessary for many actions but is not sufficient for every action. A distinct safety and legitimacy policy boundary must assess applicable prohibitions, privacy, third-party effects, and domain rules. This boundary is configured and governed outside the planning model. RGAI does not purport to settle difficult normative questions; it requires the deployment to make its operative policy explicit and testable.

4.4 Transactional execution and recovery

Execution is treated as a sequence of bounded state transitions rather than a single opaque tool call. A material action is prepared with preconditions, a target version, expected effects, side effects, verification steps, retry semantics, and a recovery plan. Where possible, changes are staged in a sandbox or branch before commit. The controller revalidates authorization and preconditions immediately before consequential execution.

An action outcome has at least three possible states:

- **Succeeded:** the tool reports success and independent postcondition evidence supports that the intended effect occurred.
- **Failed:** sufficient evidence establishes that the action did not occur or did not satisfy the required condition.
- **Indeterminate:** the system cannot establish whether the effect occurred.

Indeterminate is not equivalent to failure. A non-idempotent operation must not be blindly retried when its previous outcome is unknown. The agent should query transaction status, inspect the external system, or request intervention. For multi-system workflows, RGAI uses transaction coordination and compensating actions where available. Compensation is not assumed to be perfect rollback; it may fail, may not restore the original state, and may require human intervention [19].

4.5 Independent assurance

The assurance subsystem tests both the implementation and the specification. It asks whether the artifact or action satisfies the intended acceptance criteria and whether those criteria correctly represent the active objective.

Assurance is not defined as agreement between two language models. Independence is a property of failure modes and evidence channels. A compiler can check whether code compiles; a runtime test can check observed behavior; a numerical implementation can cross-check a calculation; a database invariant can detect

unexpected state changes; a primary source can substantiate a factual claim. Different models may assist, but correlated assumptions remain possible. High-impact cases may require human review or domain-specific certified procedures.

The assurance subsystem also examines side effects and evidence gaps. It must be able to conclude that the result is not established, even when the agent's internal process appears complete.

4.6 Shared event history and derived views

All subsystems use a common, versioned event history. Events may include task declarations, requirement amendments, observations, source assertions, belief updates, predictions, permission grants and revocations, action proposals, authorization decisions, execution attempts, outcomes, verification records, incidents, and compensating operations.

Current intent, belief, execution, dependency, authority, and assurance views are derived projections of the event history. They are not independent truth databases. Projections can be rebuilt, compared, and invalidated when upstream events change. A correction is recorded as a new event rather than silently rewriting history.

The event history should be tamper-evident and access-controlled, but not an indiscriminate permanent store of raw user data. Sensitive payloads should be minimized, encrypted, separated from metadata, access-controlled, and retained only according to explicit policy. Where deletion is required, the design should support deletion or cryptographic erasure of sensitive payloads while preserving only legally and operationally justified audit metadata. Hashes and metadata can themselves be sensitive and must be assessed accordingly.

5. Operational model

5.1 Shared state

Let the agent's derived operational state at time t be:

$$S(t) = \{I(t), E(t), B(t), P(t), A(t), X(t), D(t), V(t)\}$$

where:

- **I** is the active objective and requirement view;
- **E** is the evidence and observation view;
- **B** is the current belief and inference view;
- **P** is the prediction view;
- **A** is the authority and capability view;
- **X** is the execution and external-state view;
- **D** is the dependency and provenance graph;
- **V** is the assurance and acceptance view.

The event history is the common record from which these views are derived. No view may promote its own contents into another category without an explicit transition event and the required evidence or authorization.

5.2 Action contract

Every material action must carry a machine-checkable or reviewable contract. The minimum contract contains:

- 1. Purpose:** the active requirement or permitted preparatory forecast motivating the action.
- 2. Scope:** target resources, data classes, operation, and object versions.
- 3. Preconditions:** conditions that must be true before execution.
- 4. Authority:** required capability grant, approving actor, expiry, and revocation check.
- 5. Safety policy:** applicable restrictions and affected-party considerations.
- 6. Expected effects:** intended state changes and success predicates.
- 7. Side effects:** direct, indirect, and dependent effects considered.
- 8. Evidence plan:** how preconditions and postconditions will be observed and by which channels.
- 9. Reversibility:** reversible, compensatable, or irreversible classification.

10. Failure handling: timeout, retry, idempotency, indeterminate outcome, rollback or compensation procedure.

11. Acceptance: objective criteria for deciding whether the result is complete.

12. Data handling: sensitivity, retention, disclosure, and logging constraints.

A contract may be abbreviated for low-risk operations, but required fields cannot be omitted for high-impact actions. The executor—not the language model's recollection of policy—enforces mandatory fields.

5.3 Commit rule and acceptance rule

For a consequential action a at time t , the controller permits commitment only if:

$\text{CommitAllowed}(a,t) = \text{ContractValid}(a) \text{ AND } \text{PreconditionsHold}(a,t) \text{ AND } \text{AuthorizationCurrent}(a,t) \text{ AND } \text{SafetyAllows}(a,t) \text{ AND } \text{RiskControlsSatisfied}(a,t)$

$\text{AuthorizationCurrent}$ includes a check that the grant remains valid for the exact operation, resource, and relevant resource version at the commit boundary. $\text{RiskControlsSatisfied}$ includes any required isolation, review, backup, or emergency-envelope conditions.

A task outcome is accepted only if:

$\text{Accept}(\text{outcome}) = \text{ActiveRequirementsVerified} \text{ AND } \text{PostconditionsSupported} \text{ AND } \text{CriticalSideEffectsChecked} \text{ AND } \text{NoUnresolvedCriticalContradiction}$

The acceptance predicate is domain-specific. A passing unit test proves only what that test covers. A successful HTTP response proves only what the response semantics and independent state inspection establish. When evidence is incomplete, the system reports partial or indeterminate status rather than silently converting uncertainty into success.

5.4 Lifecycle

RGAI uses an event-driven operating cycle:

- 1. Observe:** capture relevant environment changes and source outputs with provenance.
- 2. Interpret:** update active requirements and classify incoming material as instruction, data, assertion, inference, or prediction.
- 3. Model:** maintain beliefs, uncertainty, conflicting hypotheses, and dependencies.
- 4. Anticipate:** forecast plausible future needs; prefer preparations useful across multiple futures.
- 5. Select:** compare expected utility, cost, integration burden, discard cost, privacy impact, and residual risk.
- 6. Prepare:** create an action contract and isolate speculative work where appropriate.
- 7. Authorize:** enforce capabilities, safety rules, resource scope, expiry, and approvals in the trusted controller.
- 8. Execute:** apply bounded changes and record actual tool and environment outcomes.
- 9. Verify and reconcile:** inspect postconditions, challenge assumptions, invalidate dependent beliefs when contradicted, recover or compensate, and update accepted state.

The cycle is not a requirement to run every phase at every token or for every trivial operation. Reviews are triggered by material changes and events such as contradictions, policy denials, external-state surprises, failed critical checks, permission revocations, scope changes, and high-risk commits. Fixed review intervals may be tested as an implementation policy, but no particular numerical cadence is treated as a source of correctness.

5.5 Risk-adaptive assurance

Verification effort should scale with the potential consequences of an action. RGA considers at least five factors:

- severity of plausible harm;
- uncertainty in the evidence and environment;
- irreversibility of the operation;
- exposure to third parties or external systems;
- blast radius and dependency reach.

These factors produce a risk tier, not a claim of mathematically exact probability. Low-risk actions may proceed autonomously within a narrow capability grant. Moderate-risk actions require read-back and checkpoints.

High-risk actions require staged execution, stronger evidence, and possibly human approval. Critical actions require explicit authorization, independent controls, and an applicable domain-specific procedure.

The controller must avoid two opposing errors: allowing risky actions through because the model is confident, and blocking useful work merely because uncertainty exists. Unknown information should change the risk assessment in proportion to its relevance. It should not automatically force either execution or refusal. Each deployment should also define a bounded investigation budget in time, tool calls, and cost. Exhausting that budget does not convert an unknown into a fact or grant additional authority; it triggers a risk-appropriate choice among constrained execution, abstention, or escalation.

5.6 Emergency envelopes

Emergency autonomy is pre-authorized rather than self-granted. An emergency envelope specifies triggering evidence, allowed actions, resource scope, duration, maximum permitted impact, notification requirements, and stop conditions. The agent may invoke only an envelope established by an authorized party before the incident. If no applicable envelope exists, the system follows the domain's defined escalation or safe-state procedure; it does not invent emergency powers. The safe state is domain-dependent: shutting down a server, maintaining power to a medical device, or stopping an industrial process may have different consequences.

6. Core invariants

RGAI proposes the following invariants as system-level requirements. They should be enforced by software and deployment policy wherever possible, not solely by natural-language instructions to the model.

1. A prediction cannot create or amend a hard requirement.
2. A model belief cannot become an observation merely through confidence or repetition.
3. A source assertion remains attributable to its source until adequate corroboration establishes the relevant claim.
4. An action proposal is not an authorization.
5. Authorization is specific to actor, operation, resource, scope, and time, and is revalidated at commit.
6. The model cannot grant itself privileges or modify the enforcement policy that governs its own actions.
7. An attempted action is not a completed effect.
8. An acknowledged effect is not automatically an accepted outcome.
9. An unknown state is not equivalent to confirmed absence or confirmed failure.
10. Contradictory evidence cannot be silently discarded; affected dependent decisions must be reconsidered.
11. Evaluation criteria and evaluator state must be outside the evaluated agent's unilateral control.
12. Reversible, compensatable, and irreversible actions must not be treated as equivalent.
13. Auditability does not justify unlimited collection or retention of sensitive data.
14. High-risk assurance must use evidence channels with explicitly examined common-mode failure risks.
15. Any claim of completion must be bounded by the actual scope of verification performed.

7. Threat model and failure analysis

7.1 Threat actors and failure sources

RGAI considers both adversarial and non-adversarial failure sources: mistaken model reasoning; malicious users; prompt injection in documents or webpages; compromised tools and APIs; poisoned memory; malicious or stale peer agents; concurrent environment changes; compromised credentials; evaluator tampering; controller vulnerabilities; and incomplete observability of external systems.

The architecture assumes that the language model is not a trusted reference monitor. It also assumes that external tools may be unreliable or compromised and that observations may be incomplete. It does not assume that the event store, operating system, hardware root of trust, or organizational policy is infallible. These components are part of the deployment's trusted computing base and require separate engineering and assurance.

7.2 Minimal trusted enforcement core

The minimal trusted core should contain only the components required to enforce access and record security-relevant events:

- identity and capability validation;
- the policy decision and enforcement path;
- the tool gateway through which consequential operations must pass;
- commit-time authorization and resource-version checks;
- protected configuration and revocation mechanisms;
- tamper-evident audit metadata;
- an out-of-band suspension or emergency-stop mechanism where appropriate.

The planning model must not be able to call privileged tools through an alternate path, rewrite policy, mint its own capabilities, or disable logging. The core should be small, testable, and isolated. This does not eliminate controller compromise; it reduces the trusted surface and makes compromise an explicit catastrophic failure mode to test.

7.3 Prompt injection and untrusted content

Retrieved documents, webpages, emails, code comments, and tool outputs are data, not policy authorities. Input screening may help detect known attacks but cannot serve as the security boundary. Tool permissions, parameter validation, output handling, and data disclosure must be enforced at the relevant interface [10, 11]. A malicious document must not be able to grant tool access or change the user's active objective merely by containing imperative language.

The output channel also requires governance. An agent can disclose sensitive information in a final response without making any tool call. Therefore, RGAI applies data-access and disclosure rules to generated outputs, not only to external actions.

7.4 Hidden dependencies and open-world uncertainty

A system may not observe every dependency of a resource. RGAI distinguishes: (a) no dependency was found in the inspected scope; (b) the dependency search was incomplete; and (c) absence was established within a defensible, bounded scope. A destructive action affecting an incompletely observed resource should receive a higher risk classification. This does not make hidden dependencies knowable; it prevents missing evidence from being misrepresented as evidence of absence.

7.5 Poisoned memory and dependency invalidation

A persistent belief may be wrong for a long time. Provenance helps locate its origin but cannot guarantee the belief will be challenged. RGAI attaches dependencies to consequential inferences and decisions. When source evidence expires or a belief is contradicted, dependent decisions are marked stale and selectively re-evaluated. This prevents an error correction from being limited to the memory entry while downstream work remains contaminated.

7.6 Multi-agent concurrency

Agents sharing resources must use version checks, ownership or lease policies, conflict detection, and explicit reconciliation. A shared event history can expose concurrent edits, but cannot decide which edit is correct. Conflict resolution must be governed by resource ownership, active intent, merge semantics, and re-verification of affected requirements. Last-write-wins is not a correctness policy.

7.7 Reward and evaluator tampering

The agent must not have unilateral write access to the acceptance criteria, evaluation harness, hidden test set, or scoring state used to judge its own work. Evaluation infrastructure should be isolated and externally controlled. Changes to test code, configuration, and evidence should be recorded and independently reviewed. This reduces a class of specification-gaming attacks but does not prove that the evaluator measures the user's true objective [5, 20].

7.8 Privacy and auditability tension

Complete logging can create a sensitive secondary database. RGAI therefore requires data minimization, scoped retention, access controls, encryption, and separation of event metadata from sensitive payloads. A system should preserve the evidence needed to reconstruct a decision without copying every secret or private document into a permanent log. Retention and deletion policies must be part of the action contract and event-store design.

8. Adversarial thought experiments

The following scenarios are architectural walkthroughs, not empirical test results. They identify predicted control behaviour and residual vulnerabilities.

Scenario	RGAI mechanism	Expected containment	Residual failure condition
A tool falsely reports that a file was created	Independent filesystem read-back; assertion/observation separation	The agent does not report completion when the read-back fails	The read-back channel is compromised or observes the wrong location
A document contains instructions to disclose private data	Untrusted-content classification; tool and output disclosure controls	Embedded text cannot grant authority; output disclosure is checked	The model can leak through an ungoverned channel or policy is incomplete
A user changes a requirement mid-task	Authorized amendment event; supersession history	New intent takes effect without erasing prior rationale	The change is ambiguous or authority to amend is unclear
A shared interface change breaks many dependents	Staged branch; dependency graph; regression tests; checkpoint	Failure is detected before broad acceptance where coverage exists	Tests omit an affected dependency or external state cannot be rolled back
A payment times out after submission	Indeterminate outcome; transaction-status query; idempotency controls	The agent avoids a blind duplicate payment	Payment provider has no reliable status query or idempotency contract
Two verifiers share the same wrong specification	Specification challenge and diverse evidence	Implementation correctness is separated from specification correctness	All evaluators share the same mistaken premise
An emergency occurs outside ordinary permissions	Pre-authorized emergency envelope	Bounded emergency action proceeds without self-granted authority	No applicable envelope exists or trigger evidence is unavailable
An external application secretly depends on a file	Unknown-state classification; risk escalation	Deletion is delayed, isolated, or made more conservative	The dependency is genuinely unobservable and the operation remains necessary
The agent edits the evaluation tests	Evaluator isolation and immutable acceptance configuration	The agent cannot unilaterally improve its score by weakening the tests	Evaluator credentials or infrastructure are compromised
The authority controller is bypassed	Mandatory reference monitor and isolated tool gateway	All consequential operations are forced through the policy boundary	A privileged alternate path or controller exploit exists
A false belief contaminates a long project	Provenance and belief-dependency graph	Contradiction invalidates downstream decisions for review	The false belief is never challenged or dependency tracking is incomplete
Audit logging captures secrets	Data minimization and sensitive-payload separation	Logs retain minimal decision evidence	Metadata itself leaks sensitive information or retention controls fail

These walkthroughs suggest that RGAI is most likely to help when failures involve unauthorized side effects, long-horizon state drift, partial transactions, false success claims, and recoverable changes. It is less likely to help when the primary bottleneck is model perception, domain knowledge, or a specification that no available evidence can disambiguate. RGAI may also lose to a simpler agent on low-risk tasks if its controls are poorly calibrated.

9. Evaluation programme

9.1 Research questions and hypotheses

The central research question is whether RGAI improves externally verified safe task completion relative to a competent conventional agent using the same underlying model and equivalent tools.

- **H1 — Safe completion:** RGAI increases the proportion of tasks completed with active requirements satisfied and no critical unintended side effects, particularly in long-horizon and high-risk tasks.
- **H2 — Authority integrity:** RGAI reduces unauthorized consequential actions and successful prompt-injection-to-action chains.
- **H3 — State integrity:** RGAI reduces false completion claims and discrepancies between reported and externally observed state.
- **H4 — Recovery:** RGAI improves containment and recovery after partial failures, interruptions, and concurrent modifications.
- **H5 — Proportionality:** Risk-adaptive controls preserve most low-risk task productivity while increasing cost for tasks with material side effects.
- **H6 — Limits:** RGAI does not necessarily improve core reasoning, perception, or specification understanding when these are the dominant bottlenecks.

9.2 Baselines and experimental arms

A fair comparison should include at least three arms:

- **Conventional agent:** the same model with ordinary tool-use scaffolding and standard deployment safeguards. It must not be intentionally weakened.
- **RGAI full system:** all proposed subsystems and enforcement mechanisms enabled.
- **RGAI ablations:** selected subsystems removed or simplified to identify which mechanisms produce any observed effect.

The same model weights, context budget, tool set, environment, task instructions, and sampling configuration should be used across matched arms. Because RGAI adds control logic, results should be reported both under equal model/tool budgets and under equal total compute or wall-clock budgets. Otherwise the comparison may confound architecture with additional resources.

9.3 Benchmark coverage

Existing suites provide realistic task environments but should be supplemented with controlled adversarial variants:

1. **Computer use:** OSWorld and OSWorld 2.0 [12, 13]. Add dynamic UI state, late-arriving information, revocation, and hidden-state interventions.
2. **Web workflows:** WebArena [14]. Add duplicate-submission hazards, transaction timeouts, changing prices, and external side effects.
3. **Multi-application tasks:** AppWorld [15]. Measure collateral state changes and partial completion across app boundaries.
4. **User-policy interaction:** τ -bench [16]. Test policy compliance across repeated conversations and mid-task amendments.
5. **General assistance:** GAIA [17]. Measure factuality, source support, and calibrated abstention in tool-assisted research tasks.
6. **Software engineering:** use executable repository tasks with hidden regression tests, protected evaluator state, dependency interventions, and recovery trials.
7. **Tool calling:** test argument correctness separately from authorization and postcondition verification.
8. **Security and governance:** add prompt injection, tool compromise, capability revocation, evaluator tampering, output leakage, and controller-bypass fixtures.
9. **Open-world operations:** introduce incomplete dependency maps, stale external state, and indeterminate transaction outcomes.
10. **Emergency operations:** test both pre-authorized emergency envelopes and cases where no envelope exists.

Existing benchmarks should not be assumed to test RGAI's claims directly. Their tasks and metrics need to be inspected, and custom test fixtures should be published so that the added governance phenomena are reproducible.

9.4 Metrics

A single aggregate score can conceal important trade-offs. RGAI should report a metric vector, with a strict primary outcome:

- **Safe Task Completion Rate (STCR):** fraction of tasks where the active objective is satisfied, required evidence is available, and no critical policy violation or unintended side effect occurs.
- **Requirement satisfaction:** fraction of active hard requirements independently verified.
- **Critical violation rate:** frequency of forbidden or unauthorized consequential effects.
- **False completion rate:** frequency with which the agent claims success when acceptance criteria are not met.
- **State discrepancy rate:** frequency and magnitude of differences between the agent's reported state and the external environment.
- **Recovery completeness:** proportion of induced failures for which the system restores an acceptable state or completes a documented compensation procedure.
- **Collateral change rate:** unintended changes outside the authorized task scope.
- **Uncertainty calibration:** calibration of forecast and belief estimates, stratified by domain and horizon.
- **Unknown-state quality:** whether the system correctly distinguishes unknown, not-observed, and confirmed-absent states.
- **Prompt-injection success rate:** proportion of adversarial attempts that cause an unauthorized action or disclosure.
- **Operational cost:** wall-clock latency, token use, tool calls, verifier calls, storage, and human interventions.
- **Usability burden:** unnecessary approval requests, blocked safe actions, and user bypass attempts.

Task success and safety should be reported separately as well as through STCR. A system that refuses all tasks can achieve a low violation rate but is not a useful agent. A system that completes every task while causing harmful side effects is not reliable.

9.5 Repetition and statistical analysis

A pilot should first validate task instrumentation and failure injections. A confirmatory study should use matched task instances and repeated randomized seeds across architectures. Task order should be randomized, evaluators blinded to system identity where feasible, and the evaluation harness isolated from the agents. Confidence intervals should be computed at the task-instance level, with clustering for repeated trials of the same task. Binary outcomes can be analysed with mixed-effects logistic models; continuous cost and recovery measures should use distributions appropriate to their observed shape. Multiple comparisons should be controlled, and negative or null findings should be reported.

A feasible initial design is 12-24 task families with at least 30 distinct task instances per family, three architecture arms, and repeated seeds for stochastic tasks. The final sample size should be set by a power analysis using pilot variance and the smallest operationally meaningful effect, not selected after observing results. High-severity security tests should be separately reported and should not be diluted by a large number of trivial tasks.

9.6 Ablation programme

The following ablations are necessary:

- remove objective governance;
- merge beliefs with observations;
- remove provenance and dependency invalidation;
- bypass the external authorization controller in a controlled test environment;
- remove commit-time permission revalidation;
- remove output-channel disclosure controls;
- remove transactional recovery and idempotency handling;
- replace independent assurance with model-only critique;
- disable risk adaptation and apply uniform gating;
- remove emergency envelopes;
- remove concurrency and version checks;
- disable privacy minimization.

Ablations should measure both the mechanism's direct effect and interactions with other mechanisms. Some components may provide little value alone but become important when combined.

9.7 Falsification criteria

RGAI should be considered unsuccessful, or substantially revised, if:

1. It does not improve safe task completion or critical-violation rates on tasks where its mechanisms are relevant.
2. Its controller can be bypassed through an alternate path available to the model.
3. Its verifier frequently shares the same specification or evidence error as the producer.
4. It lowers false-success reporting only by refusing a large fraction of useful tasks.
5. It creates more harmful failures through controller complexity than it prevents through governance.
6. Its privacy controls make audit records insufficient or its audit design creates unacceptable new disclosure risks.
7. Its risk controller does not keep low-risk overhead within a pre-registered practical bound.
8. It cannot correctly handle indeterminate outcomes and non-idempotent retries.
9. It performs well only on tests designed by its own authors and not on independent hold-out tests.

The primary claim is falsifiable: if a matched, independently evaluated comparison fails to show an operationally meaningful improvement in safe task completion at acceptable cost, the architecture has not demonstrated its intended value.

10. Limitations and open research problems

10.1 The model remains fallible

RGAI governs actions and evidence but does not directly improve the model's underlying reasoning, perception, world knowledge, or ability to interpret a specification. If the model cannot identify the relevant dependency or understand the task, the architecture may only make the uncertainty more visible.

10.2 Verification is not omniscience

Independent evidence can still be incomplete, compromised, or based on an incorrect specification. Diversity of tools is not proof of statistical independence. Domain-specific assurance must identify common-mode failure risks and define the scope of every conclusion.

10.3 The controller becomes a high-value target

The minimal trusted core concentrates authority. It must be isolated, least-privileged, tested, monitored, and designed to fail in a domain-appropriate manner. A compromised controller can defeat the architecture. RGAi reduces reliance on model self-restraint; it does not remove the need for secure systems engineering.

10.4 Risk classification is difficult

A risk-adaptive controller may underestimate an unfamiliar failure or overestimate benign uncertainty. Static labels are not sufficient when environments change. Risk policies need calibration, audit, conservative fallback rules, and human governance. The optimum threshold may vary substantially by domain.

10.5 Open-world uncertainty remains irreducible

No agent can inspect every external dependency or observe every relevant future event. RGAi can represent unobserved state and avoid claiming certainty, but it cannot eliminate hidden dependencies. Some operations remain unsafe without human knowledge or domain-specific guarantees.

10.6 Privacy and auditability can conflict

Provenance and event history improve accountability but can create sensitive records. Data minimization, redaction, retention limits, encryption, access control, and deletion processes are necessary, and they may reduce future forensic detail. This is a governance trade-off, not a storage implementation detail.

10.7 Complexity and adoption risk

If controls are slow, intrusive, or difficult to integrate, users and developers may bypass them. The architecture must be evaluated as a product system, not just a formal design. The most robust control that operators routinely disable may be less effective than a simpler control that remains enabled.

10.8 Normative policy is not solved by architecture

RGAI requires an explicit safety and legitimacy boundary, but it cannot determine all contested legal, ethical, or social questions. The policies governing third-party rights, privacy, safety, and acceptable risk must be established through legitimate institutional and domain processes.

11. Implementation roadmap

A minimum implementation should avoid building every subsystem at once. The first prototype should wrap an existing agent with:

1. a typed task and requirement record with amendment history;
2. a typed evidence record separating observation, assertion, inference, prediction, and unknown;
3. an external tool gateway with scoped capability checks and commit-time revocation checks;
4. action contracts for consequential tool calls;
5. execution states including succeeded, failed, and indeterminate;
6. postcondition read-back and a simple dependency graph;
7. a privacy-minimized event history;
8. an independent acceptance evaluator.

The prototype should initially target one environment with observable state and reversible operations, such as a sandboxed multi-application workflow or a repository task environment. It should then add prompt injection, false acknowledgements, interrupted operations, concurrent writes, and hidden dependencies. Only after the core controller survives these tests should it expand to real external side effects.

The implementation should document its trusted computing base, threat model, permitted alternate execution paths, event-retention policy, and recovery guarantees. A design review should explicitly identify which safeguards are enforced by code, which depend on external infrastructure, and which remain advisory.

12. Conclusion

Reliable autonomous AI requires more than better instructions or more capable language models. When an agent can change external state, the system must govern how objectives are interpreted, how information acquires evidential status, how permissions are granted and revoked, how actions are committed, and how outcomes are accepted.

RGAI proposes an integrated architecture built around five responsibilities: objective governance, epistemic integrity, authority governance, transactional execution and recovery, and independent assurance. A shared event history preserves provenance and supports derived state views, while a minimal trusted core enforces action boundaries independently of the planning model. Risk-adaptive verification, emergency envelopes, indeterminate execution states, dependency invalidation, concurrency control, and privacy-minimized audit records address failure modes exposed by adversarial analysis.

The architecture's promise is not that mistakes disappear. It is that a model's mistake should not automatically become an authorized action, an accepted fact, or a declared success. Its central hypothesis is that explicit, evidence-governed transitions can improve the reliability of autonomous agents enough to justify their implementation and operating costs. That hypothesis remains unverified. The appropriate next step is a matched, budget-controlled, independently scored implementation study with adversarial hold-outs, ablations, and externally observable outcomes. RGA should be adopted only to the extent that such tests demonstrate a practical improvement.

AI assistance and reproducibility disclosure

This manuscript was prepared with AI assistance at the direction of the named author. It describes a proposed architecture and an evaluation programme; it does not report completed implementation or empirical

experiments. Before submission or publication, the author should verify the references, review the novelty claims against current literature, revise the text, and approve the final manuscript in accordance with the relevant venue's authorship and AI-assistance policies.

References

- [1] RM369. (2026). *PECA v3: Evidence-Gated Anticipatory Execution: Separating Requirement Compliance, Predictive Utility, and Commit Authorization in AI Agents*. Preprint, Life of Research, 26 October 2026. <https://lifeofresearch.com/preprints/140>
- [2] Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., & Cao, Y. (2023). ReAct: Synergizing reasoning and acting in language models. *International Conference on Learning Representations*. <https://arxiv.org/abs/2210.03629>
- [3] Madaan, A., Tandon, N., Gupta, P., Hallinan, S., Gao, L., Wiegrefe, S., et al. (2023). Self-Refine: Iterative refinement with self-feedback. arXiv:2303.17651. <https://arxiv.org/abs/2303.17651>
- [4] Gou, Z., Shao, Z., Gong, Y., Shen, Y., Yang, Y., Duan, N., & Chen, W. (2024). CRITIC: Large language models can self-correct with tool-interactive critiquing. *International Conference on Learning Representations*. <https://arxiv.org/abs/2305.11738>
- [5] Amodei, D., Olah, C., Steinhardt, J., Christiano, P., Schulman, J., & Mané, D. (2016). Concrete problems in AI safety. arXiv:1606.06565. <https://arxiv.org/abs/1606.06565>
- [6] Tabassi, E. (2023). *Artificial Intelligence Risk Management Framework (AI RMF 1.0)*. NIST AI 100-1. National Institute of Standards and Technology. <https://doi.org/10.6028/NIST.AI.100-1>
- [7] Rose, S., Borchert, O., Mitchell, S., & Connelly, S. (2020). *Zero Trust Architecture*. NIST Special Publication 800-207. National Institute of Standards and Technology. <https://doi.org/10.6028/NIST.SP.800-207>
- [8] World Wide Web Consortium. (2013). *PROV-DM: The PROV Data Model*. W3C Recommendation. <https://www.w3.org/TR/prov-dm/>
- [9] OWASP GenAI Security Project. (2025). *LLM06:2025 Excessive Agency*. <https://genai.owasp.org/llmrisk/llm062025-excessive-agency/>
- [10] OWASP GenAI Security Project. (2025). *LLM Prompt Injection Prevention Cheat Sheet*. https://cheatsheetseries.owasp.org/cheatsheets/LLM_Prompt_Injection_Prevention_Cheat_Sheet.html
- [11] OWASP Agentic Security Initiative. (2025). *Agentic AI: Threats and Mitigations*. <https://genai.owasp.org/resource/agentic-ai-threats-and-mitigations/>
- [12] Xie, T., Zhang, D., Chen, J., Li, X., Zhao, S., Cao, R., et al. (2024). OSWorld: Benchmarking multimodal agents for open-ended tasks in real computer environments. arXiv:2404.07972. <https://arxiv.org/abs/2404.07972>
- [13] Yuan, M., Zhou, Z., Xiong, X., Wu, W., Sun, J., Song, J., et al. (2026). OSWorld 2.0: Benchmarking computer use agents on long-horizon real-world tasks. arXiv:2606.29537. <https://arxiv.org/abs/2606.29537>
- [14] Zhou, S., Xu, F. F., Zhu, H., Zhou, X., Lo, R., Sridhar, A., et al. (2024). WebArena: A realistic web environment for building autonomous agents. *International Conference on Learning Representations*. <https://arxiv.org/abs/2307.13854>
- [15] Trivedi, H., Khot, T., Hartmann, M., Manku, R., Dong, V., Li, E., et al. (2024). AppWorld: A controllable world of apps and people for benchmarking interactive coding agents. *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, 16022-16076. <https://doi.org/10.18653/v1/2024.acl-long.850>
- [16] Yao, S., Shinn, N., Razavi, P., & Narasimhan, K. (2024). τ -bench: A benchmark for tool-agent-user interaction in real-world domains. arXiv:2406.12045. <https://arxiv.org/abs/2406.12045>
- [17] Mialon, G., Fourrier, C., Wolf, T., LeCun, Y., Scialom, T., et al. (2024). GAIA: A benchmark for general AI assistants. *International Conference on Learning Representations*. https://proceedings.iclr.cc/paper_files/paper/2024/hash/25ae35b5b1738d80f1f03a8713e405ec-Abstract-Conference.html
- [18] Geng, J., Cai, F., Wang, Y., Koeppl, H., Nakov, P., & Gurevych, I. (2024). A survey of confidence estimation and calibration in large language models. *Proceedings of NAACL-HLT 2024*, 6577-6595. <https://doi.org/10.18653/v1/2024.naacl-long.366>
- [19] Microsoft. (2026). *Compensating Transaction Pattern*. Azure Architecture Center. <https://learn.microsoft.com/azure/architecture/patterns/compensating-transaction>
- [20] Anthropic. (2024). *Sycophancy to subterfuge: Investigating reward tampering in language models*. <https://www.anthropic.com/research/reward-tampering>