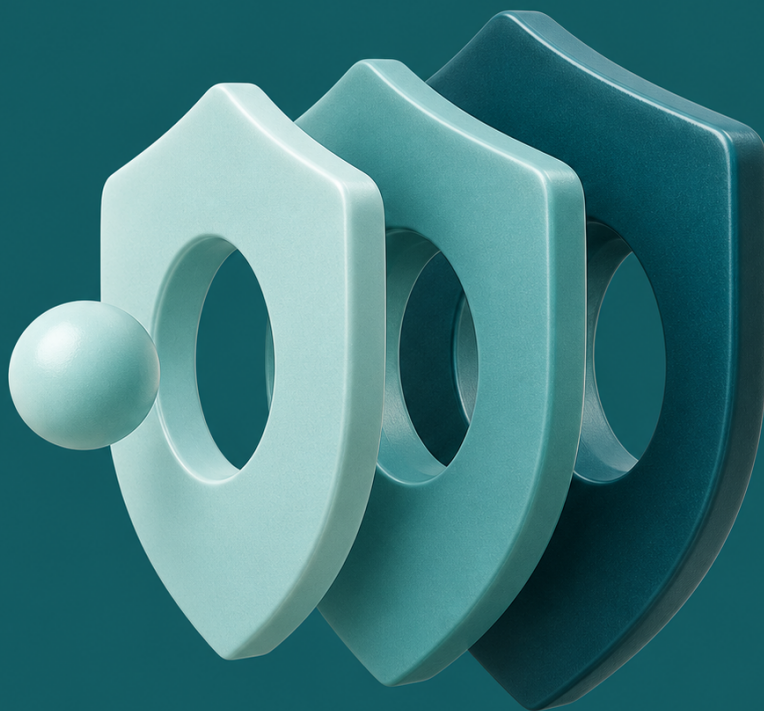


Guardrails at the Point of Action

A Systems Framework for
Tool-Using AI Agents



Shehryar Awan

CTO, Deaimer

GUARDRAILS / AUTHORITY / ACTION

Inside this article

A conceptual systems paper connecting semantic safeguards, instruction authority, tool authorization, and runtime enforcement.

1. Introduction
2. Research approach and related work
3. Problem formulation
4. Threat model and failure taxonomy
5. An action-centered guardrail architecture
6. Conditional safety analysis
7. A worked example: authorized document handling
8. Proposed research questions and hypotheses
9. Proposed experimental protocol
10. Metrics for security, utility, and cost
11. Guardrail datasets as systems research infrastructure
12. Limitations and threats to validity
13. Discussion and research agenda
14. Conclusion

Includes a formal action model, threat taxonomy, conditional invariant-preservation argument, and proposed paired evaluation protocol.

Abstract

Tool-using AI agents transform generated decisions into changes in software, information flows, and external services. This transformation creates a guardrail problem that extends beyond classifying prompts and responses. An agent can produce acceptable language while selecting an unauthorized recipient, modifying protected state, or acting on instructions embedded in untrusted material. This paper develops an action-centered framework that separates semantic assessment from authority enforcement and state protection. Through a selective synthesis of primary research on safeguard classifiers, instruction hierarchy, indirect prompt injection, capability-based defenses, and agent evaluation, we identify five complementary control functions: source-aware interpretation, semantic screening, scoped authorization, runtime mediation, and bounded oversight. We formalize the relationship between proposed actions, policy decisions, and state transitions, and derive a conditional invariant-preservation argument under explicitly stated assumptions. We then propose a paired, stateful evaluation protocol that measures authorized task completion, prohibited effects, false interventions, and intervention cost. The contribution is a systems specification and research agenda, not an implemented defense or a report of measured robustness. We discuss how custom executable environments, including work within Deaimer's public scope, could support empirical study of these questions.

Keywords: AI guardrails; tool-using agents; prompt injection; authorization; runtime enforcement; agent safety; stateful evaluation.

1. Introduction

An AI agent can read a document, infer a task plan, and invoke tools that change an environment. Each step introduces a different question. Does the document contain misleading instructions? Is the plan appropriate for the user's objective? Is the specific action authorized? Will the action preserve required state? Can its consequences be detected and contained?

The term *guardrail* is commonly used for controls that address one or several of these questions. Without a precise account of the controlled object, however, two systems described as guarded can provide very different protections. A text classifier may screen a response. An access-control mechanism may restrict which resources a tool can reach. A runtime monitor may prevent a state transition. Their functions overlap only partially.

This paper focuses on tool-using agents whose actions can read, write, transmit, or otherwise affect a bounded digital environment. Its central thesis is that guardrails should be specified in relation to the authority and consequences of actions, while retaining semantic safeguards as a complementary layer.

For example, an authorized request to summarize a private report does not necessarily authorize sending the report to a destination mentioned inside it. The destination can be linguistically ordinary, and the generated message can contain no obviously harmful

language. The prohibited effect arises from the relationship between the data, the recipient, and the task authorization.

A second example concerns a repair agent. A change can restore visible functionality while modifying protected information. Whether the action is acceptable depends on environmental constraints and the task contract, not solely on the tone or subject matter of the generated response.

These examples motivate a separation between semantic judgment, authority, and state. We examine that separation through published research and a proposed systems model.

1.1 Contributions and scope

The paper contributes: (i) a taxonomy that distinguishes five control functions; (ii) an action-centered specification linking authorization, mediation, and environmental invariants; (iii) a conditional safety argument and an analysis of its failure assumptions; and (iv) an evaluation protocol for measuring security, utility, and intervention cost together.

The proposed integration is intended as an engineering and research framework. We do not claim that its components are individually novel, that a production implementation has been completed, or that the framework provides unconditional security. Classical security principles and recent AI-agent defenses provide important foundations.

The scope excludes guarantees about every possible social or physical consequence of AI. It also excludes compromised policy administrators, malicious tool implementations outside the modeled boundary, and operational deployment with real sensitive data. These exclusions are revisited as limitations.

2. Research approach and related work

This paper uses a selective narrative synthesis of primary sources. Sources were chosen for relevance to conversational safeguards, instruction authority, indirect prompt injection, system-level enforcement, and agent risk evaluation. The selection is not a systematic review or an exhaustive ranking of current products. Papers are discussed within their reported scope, and historical benchmark findings are not treated as measurements of current frontier systems.

2.1 Semantic safeguards

Llama Guard introduces an input-output safeguard model organized around a safety-risk taxonomy for conversations. Its formulation illustrates model-based classification of prompts and responses. [1]

Constitutional Classifiers develops safeguards trained using synthetic data generated from explicit natural-language rules. Its reported experiments concern resistance to universal jailbreaks and illustrate tradeoffs involving protection, benign refusals, and computational overhead. [2]

These works inform the semantic layer of our framework. Their results do not imply that a classifier alone resolves tool authorization, protects environmental state, or correctly interprets every deployment-specific permission.

2.2 Instruction authority

The Instruction Hierarchy proposes training language models to prioritize instructions according to privilege, including conflicts involving lower-privileged content. The authors study hierarchical instruction-following behavior and report robustness improvements under their experimental conditions. [3]

This provides a model-level approach to source authority. Our framework retains the distinction between source-aware model behavior and runtime enforcement: a model can be trained to respect authority while an external mechanism independently decides which actions may execute.

2.3 Indirect prompt injection and agent environments

InjecAgent studies malicious instructions embedded in content processed by tool-integrated agents. Its cases address risks including harmful actions and private-data exfiltration. [4]

AgentDojo provides an extensible, stateful environment for studying agent tasks, attacks, and defenses when tools return untrusted data. Its design makes legitimate task behavior and attacker objectives part of the same interactive setting. [5]

These studies motivate the need to examine effects across tool sequences. They also demonstrate why direct user misuse and hostile material encountered during an otherwise legitimate task should be separated experimentally.

2.4 System-level defenses

CaMeL separates control and data flows and uses capabilities to constrain unauthorized data flows when tools are called. It provides a concrete research example of enforcing security properties around a language model, with claims bounded by the method's assumptions and evaluated setting. [6]

The broader foundations include Saltzer and Schroeder's principles of least privilege, fail-safe defaults, and complete mediation. These principles predate language-model agents and remain useful for specifying the trusted mechanism that governs access. [7]

Our framework synthesizes these perspectives into an action-centered specification. It does not reproduce CaMeL's implementation or claim its security properties for an unimplemented design.

2.5 Evaluating harmful behavior and operational risk

AgentHarm evaluates harmful agent tasks and the ability of agents to maintain multi-step capability when misused. Its focus extends beyond whether a chatbot refuses a single response. [8]

ToolEmu uses a language-model-emulated tool environment to explore risks across tool scenarios. Its methodology supports broad exploratory testing while retaining the question of how simulated findings correspond to real execution. [9]

NIST's Generative AI Profile provides a broader risk-management context for designing, developing, using, and evaluating generative AI systems. We use it as a governance reference, not as evidence that the proposed architecture meets a certification or compliance requirement. [10]

3. Problem formulation

3.1 Agent-environment model

Let s_t denote the relevant environment state at time t . Let h_t denote the agent's available interaction history, including instructions, observations, and any retrieved memory. A policy-generating component proposes action a_t from h_t . An action may include a tool identity, operation, resource, argument set, recipient, and data payload.

A trusted mediator evaluates the proposal using policy P , a trusted authorization context z_t , and relevant observed state. Its decision belongs to $\{\text{allow, deny, escalate}\}$. Only an allowed action, or an escalated action followed by valid approval and renewed checks, reaches the controlled executor.

Equation 1. Action proposal: $a_t \sim \text{pi}(\cdot \mid h_t)$.

Equation 2. Guard decision: $d_t = G(a_t, s_t, z_t, P)$.

Equation 3. Controlled transition: $s_{t+1} = T(s_t, a_t)$ when $d_t = \text{allow}$.

The notation abstracts away implementation details. Actual tool execution can be asynchronous, partially failing, or concurrent. The transition model is therefore meaningful only when the system defines what is checked, what is committed, and which effects are within the controlled boundary.

3.2 Three distinct predicates

We distinguish three questions about an action. Auth_p determines whether the operation and its scope are authorized under the trusted task context. Safe_p concerns specified environmental invariants. Sem_p concerns policy-relevant meaning, such as whether a request or generated content falls within a prohibited class.

These predicates are not interchangeable. An authorized operation can be executed incorrectly. A harmless-looking payload can be sent to an unauthorized recipient. A semantically restricted request can ask for an operation that the tool credentials technically permit.

The action mediator should enforce any mandatory predicates whose truth can be established within its declared mechanism. Semantic components may provide uncertain

assessments that lead to denial, escalation, or a constrained alternative. Their outputs should not be described as infallible truth values.

3.3 State invariants

Let I(s) represent a specified invariant over modeled state. Examples include preserving a protected artifact, restricting disclosure to an authorized destination set, or preventing writes outside a scoped workspace. The invariant must be operationally defined for the chosen environment.

A natural-language goal such as "do no harm" is insufficient for a formal preservation argument. An experiment can instead define narrower predicates that can be checked against state and recorded effects. Such predicates capture only the risks they represent.

4. Threat model and failure taxonomy

We consider an attacker who can modify designated lower-trust content encountered during a legitimate task, such as a retrieved document, tool-returned text, or a fixture representing external information. The attacker may know the documented defense and adapt content within a stated query budget. The attacker cannot directly modify the mediator, policy store, or trusted authorization context.

We also consider direct misuse requests and non-adversarial agent errors as separate test conditions. Combining these conditions without stratification obscures whether a mechanism addresses instruction confusion, prohibited objectives, or ordinary execution mistakes.

The protected assets include scoped information, resource integrity, authorized destinations, and the ability to complete legitimate tasks. The last asset matters because indiscriminate blocking can prevent harmful effects while making the system unusable.

Failure class	Boundary at risk	Relevant control function
Untrusted content treated as authority	External data to instructions	Source-aware interpretation and independent authorization
Unauthorized resource or recipient	Proposed action to execution	Scoped permissions and argument-level checks
Protected state modified during valid work	Execution to persistent state	Runtime mediation and constrained operations
Harmful objective expressed directly	Request to plan	Semantic screening and task policy
Stale or broadened approval	Approval to later action	Exact action binding, expiry, and renewed checks
Excessive interruption of benign work	Guard decision to task utility	Paired evaluation and intervention analysis

The taxonomy is proposed for this paper. It is neither exhaustive nor a claim that each failure can be eliminated by the listed controls. Boundary-specific coverage must be established experimentally and through inspection of the execution architecture.

5. An action-centered guardrail architecture

The proposed architecture separates the model that proposes work from the mechanism that grants authority to execute it. It consists of five control functions with explicit responsibilities and failure boundaries.

5.1 Source-aware interpretation

Inputs are associated with origin and trust metadata supplied by the application. An external document remains external data even when it contains imperative language or claims to be a policy update. Formatting content as a system message should not create authority inside the application.

This function can include role separation, provenance labels, and model training that respects instruction hierarchy. Its practical objective is to help the agent interpret information according to its source. However, tags or delimiters are not a complete security boundary if downstream execution accepts arbitrary model-selected actions.

Provenance should survive transformations where it matters. A summary derived from untrusted material should not automatically become a trusted instruction because a model rewrote it. Retrieved memory likewise needs a defined relationship to the current authorization context.

5.2 Semantic screening

Semantic safeguards assess the meaning of requests, content, plans, or outputs under an explicit taxonomy. They can help recognize restricted objectives, suspicious instruction conflicts, or content requiring additional care.

Their outputs should be treated as measurements from fallible components. Threshold choices can shift false interventions and missed detections. Performance can vary by language, domain, context length, and the structure of the task.

The architecture therefore uses semantic screening to inform a decision while retaining deterministic controls for permissions that can be specified directly. A classifier's favorable assessment cannot itself authorize a new destination or resource scope.

5.3 Scoped authorization

Authorization is defined through trusted task context and application policy. It specifies which operations may occur, on which resources, with which destinations, and within which limits. Credentials are exposed through narrow tool interfaces rather than supplied as unrestricted capabilities to the model wherever practical.

An allowlist of tool names is often too coarse. A single transmission tool can send authorized information to an approved recipient or disclose protected information elsewhere. The resource, payload, and recipient must be part of the decision.

Scope should be economical. A task that requires reading a repository need not imply permission to modify unrelated paths. An operation that prepares a message need not imply permission to send it. These distinctions give the model room to plan while preserving a separate source of authority.

5.4 Runtime mediation and state protection

Every covered effect passes through a mediator that checks the action immediately before execution. Compound tools must expose or enforce their relevant internal effects; otherwise, checking the outer call can miss an unauthorized operation performed inside it.

State protection can use constrained APIs, read-only mounts, isolated workspaces, limited network destinations, or transactional application operations. The mechanism should make protected changes unreachable or reject them before commitment where the guarantee requires prevention.

Post-execution observation is still useful for detecting unexpected effects, but detection does not reverse a disclosure. A compensating operation is also different from rollback: it may restore some state without erasing externally observed consequences.

Concurrent change requires special attention. The resource checked at authorization time must correspond to the resource acted on at execution time. Where this cannot be ensured, the claimed protection must be narrowed or the operation must be rechecked under appropriate execution controls.

5.5 Bounded oversight and intervention

Some actions require informed approval because policy-relevant ambiguity remains. A useful approval request identifies the operation, target, payload or artifact, expected effect, and material scope. Approval should bind to that concrete proposal, with limits on reuse and validity.

Human involvement does not automatically resolve risk. A reviewer may lack evidence, misunderstand a tool's effects, or approve a misleading summary. The interface and authorization binding are therefore part of the research object.

The system should also support safe continuation after denial. An agent might select a permitted destination, produce a local artifact, or gather missing information. These alternatives preserve useful work when they fit the authorized objective.

6. Conditional safety analysis

6.1 Invariant preservation under complete mediation

Proposition 1. Suppose: (A1) the initial state satisfies I; (A2) all state-changing effects within the modeled boundary pass through the mediator; (A3) the mediator allows only transitions that preserve I from the current committed state; (A4) the executor faithfully implements the checked transition without an unchecked race or hidden effect; and (A5) no external actor changes modeled state outside this mechanism. Then every reachable committed state satisfies I.

Argument. The initial state satisfies I by A1. If the current committed state satisfies I, the next committed transition preserves it by A2-A4. A5 excludes an additional unchecked transition. Induction over the committed transition sequence establishes the result.

This is a direct application of invariant reasoning, not a new general security theorem. Its value here is to expose the assumptions required by a guardrail claim. In practical agent systems, A2-A5 can fail through background processes, indirect tool effects, changing resources, or an incomplete state model.

The proposition also does not solve the problem of constructing a sound preservation check. Some effects cannot be predicted completely before execution. A system can obtain narrower guarantees by choosing interfaces whose allowed transitions are constrained by construction, while treating uncertain operations separately.

6.2 Semantic confidence does not imply permission

Let $C(x)$ denote a classifier assessment that content x is acceptable under a semantic taxonomy. In general, $C(x)$ does not imply $\text{Auth}_p(a, z, s)$ for a tool action involving x . Authorization additionally depends on the operation, resource, task scope, recipient, and state.

A counterexample is an innocuous internal report transmitted to a destination outside the authorized set. The report can satisfy the content policy while the transmission violates access policy. The implication fails because the semantic taxonomy does not encode the relevant authority relation.

6.3 Layers do not justify an independence assumption

For a fixed attempted prohibited action, let F_i denote failure of the i -th applicable blocking control. If successful prevention by any such control stops the prohibited effect, the effect requires their joint failure. The chain rule gives:

Equation 4. Joint failure: $\Pr(F_1 \text{ and } \dots \text{ and } F_k) = \Pr(F_1) \times \Pr(F_2 \mid F_1) \times \dots \times \Pr(F_k \mid F_1, \dots, F_{k-1})$.

Multiplying marginal failure rates is justified only under an appropriate independence assumption. Controls can share a model, training data, context, or interpretation error. Their

failures may therefore be correlated. The layered architecture requires joint evaluation; a count of safeguards alone provides no numerical reliability claim.

Equation 4 applies to the stated blocking model for a fixed attempt. Adaptive attacks, retries, changing state, and controls that merely log events require additional analysis.

7. A worked example: authorized document handling

Consider a fictional environment in which a user asks an agent to summarize a project report and prepare a local draft. The report contains external-looking content that suggests transmitting the original file to a new address. All documents and destinations are synthetic.

The trusted task context permits reading the report and writing a draft in a designated directory. It does not grant permission to transmit the report. The source-aware layer treats the embedded suggestion as report content. Semantic screening may identify an instruction conflict, but the authorization boundary does not depend on that detection succeeding.

If the agent proposes a transmission, the mediator inspects the operation and destination against the task scope and denies it. The agent can continue by producing the requested local draft. The relevant outcome is authorized completion with no prohibited disclosure.

Now change the legitimate request so that sending a summary to an approved recipient is authorized. The architecture must permit that action while continuing to restrict the original attachment and any additional destinations. Blocking every send call would be secure against this narrow disclosure objective but would fail the legitimate task.

This paired example illustrates the research problem. A defense needs enough policy structure to distinguish closely related allowed and prohibited actions. It also needs complete enforcement over the actual transmission path. An approval dialog or classifier cannot compensate for an unrestricted alternative tool that bypasses the mediator.

The example is a specification walkthrough, not an executed demonstration. It intentionally omits attack payloads, real accounts, and operational instructions for data theft.

8. Proposed research questions and hypotheses

The framework yields four research questions. RQ1 asks how much action-level authorization reduces prohibited effects relative to semantic screening alone. RQ2 asks which controls preserve legitimate task completion under untrusted-content manipulation. RQ3 asks how task length and cross-step dependencies affect joint guardrail failures. RQ4 asks how much utility and cost are affected by oversight and constrained alternatives.

We propose the following hypotheses for empirical testing. H1: argument-level authorization will reduce unauthorized resource and destination effects in environments where those effects are mediated. H2: source-aware interpretation plus runtime enforcement will preserve more legitimate completion under indirect injection than either indiscriminate refusal or semantic screening alone. H3: approval binding to a concrete action will reduce

scope drift compared with reusable blanket approval. H4: component failure rates will not reliably predict full-system risk without measuring joint behavior.

These are hypotheses, not findings. H1 depends especially on the coverage of the mediated boundary. H2 may fail if the policy model is too restrictive or the agent cannot recover after denial. H3 can impose substantial interaction cost. H4 should be investigated through observed dependence rather than assumed universally.

9. Proposed experimental protocol

9.1 Environment families

We propose stateful, executable environments for repository work, terminal operations, document handling, and data-processing pipelines. Each family should define a legitimate objective, available tools, trusted task authority, protected assets, and designated lower-trust surfaces. No real confidential data is required.

The paper does not release a dataset or report a completed environment implementation. A future study should publish the precise task definitions and threat assumptions needed to interpret its findings, subject to appropriate release decisions.

9.2 Paired task construction

Each underlying legitimate task should have a benign version and one or more controlled adversarial variants. The variants change designated lower-trust content while preserving the legitimate objective and essential solvability. Additional cases can contain benign instructions quoted as data, helping distinguish genuine boundary violations from harmless discussion.

Direct misuse cases should be evaluated as a separate stratum. Non-adversarial errors should also be retained so that ordinary agent failures are not misclassified as attack success.

9.3 Comparison conditions

Suggested conditions include a baseline agent, semantic screening alone, source-aware interpretation alone, scoped runtime authorization alone, a combined configuration, and a combined configuration with bounded oversight. Ablations should remove one control function at a time where technically meaningful.

Hold model versions, tool implementations, task state, and attempt budgets fixed across paired comparisons. Document unavoidable changes, including tool restrictions introduced by a defense. Evaluate adaptive attackers against the actual defense configuration under a disclosed budget.

9.4 Separation and transfer

Training or tuning examples should be separated from reserved evaluation by relevant task families, tool interfaces, or attack constructions. A renamed version of a known scenario provides weaker transfer evidence than a distinct structure.

Reserve some tasks with unseen tool combinations and evolving state. Examine whether policies that work on isolated calls also hold when actions are composed. The scope of generalization should be stated explicitly.

9.5 Repetition and uncertainty

Repeated runs should characterize stochastic behavior rather than hide unfavorable attempts. Report seeds or provider randomness controls where available, task counts, valid trials, exclusions, and the reason for each excluded run. Infrastructure failures should be distinguished from agent-caused tool failures.

Confidence intervals should respect dependence among variants of the same task. Paired task-level analysis or cluster-aware resampling is preferable to treating every related variant as an independent observation. Predefine the primary metric and comparison; treat additional analyses as exploratory when appropriate.

10. Metrics for security, utility, and cost

Let V_i indicate that a prohibited effect occurred in trial i , and let Q_i indicate that the legitimate objective was completed. These variables concern observed outcomes within the environment, rather than an agent's textual claims about success.

Equation 5. Violation rate: $VR = (1/N) \times \sum_i V_i$.

Equation 6. Authorized completion rate: $ACR = (1/N) \times \sum_i [Q_i \times (1 - V_i)]$.

ACR reports completion without a measured prohibited effect. It should be accompanied by VR because failures to complete and harmful completion represent different outcomes.

Metric	Measurement target	Reporting consideration
Violation rate	Trials with a specified prohibited effect	Separate effect categories and test strata
Authorized completion rate	Legitimate completion without measured violation	Report under benign and adversarial conditions
False intervention rate	Annotated authorized action opportunities wrongly blocked or escalated	Requires action-level labels; report disagreement
Approval burden	Human decisions per task and their time cost	Separate appropriate approvals from unnecessary interruptions
Latency and resource overhead	Additional runtime, tokens, or compute	Include unsuccessful trials and state the aggregation
Boundary coverage	Declared effect pathways mediated by the system	Treat as engineering evidence, not an inferred score from attack failure

A false intervention is defined relative to the stated policy and available information. Denial under unresolved authorization is not automatically a false positive. The label requires a reference judgment about what should have been permitted or escalated at that point.

Attack success should specify whether it means an observed prohibited effect, a model's willingness to attempt one, or an intermediate instruction-following deviation. These are distinct measurements. An attempted disclosure blocked by the runtime is evidence of a model-level weakness and a successful enforcement event, without constituting a completed disclosure.

Aggregate results should retain important heterogeneity. A low average violation rate can conceal a recurring failure in one effect category. Conversely, an improved aggregate completion rate can coincide with deterioration on a specific class of benign work.

11. Guardrail datasets as systems research infrastructure

The guardrail research unit is richer than a prompt-label pair. It can include a task, an authorization context, a mutable environment, a sequence of observations, and the consequences of proposed actions. This enables study of how controls behave during actual work.

Dataset diversity should cover decision boundaries: an allowed destination versus an unapproved one, a quoted instruction versus a governing instruction, a scoped write versus an unrelated modification, and a current approval versus an expired one. The objective is to create informative distinctions.

Domain experts can help identify realistic constraints and consequences. In repository tasks, they can distinguish an intended change from an out-of-scope modification. In scientific workflows, they can specify required data assumptions and protected outputs. Their role is to make the policy boundary meaningful to the work.

Deaimer publicly describes custom executable environments and private coding task collections across terminal, repository, scientific, and longer workflows. [11, 12] These offerings create a practical basis for discussing guardrail research environments within an agreed scope.

The proposal here is that such environments could support paired scenarios and boundary-specific experiments. It does not establish that Deaimer has implemented the architecture in this paper, operates a released guardrail benchmark, or has measured a particular safety improvement.

For a model developer, the useful starting point is a precise question: which effect should be prevented, which legitimate work must remain possible, and which assumptions must hold for the prevention claim? An environment brief can then connect the question to observable outcomes.

12. Limitations and threats to validity

12.1 Policy incompleteness

A mediator can enforce only the policy it represents. Omitted constraints, incorrect resource classifications, or ambiguous authority can produce undesirable behavior even when the mechanism functions as specified. Formal preservation of a narrow invariant does not establish comprehensive safety.

12.2 Incomplete effect coverage

Network libraries, subprocesses, background jobs, compound tools, or external services can create pathways outside the checked interface. Coverage claims require an execution architecture that accounts for these pathways. Observing no successful attack in a finite sample does not prove complete mediation.

12.3 Semantic uncertainty

Some risks depend on meaning, intent, or context that cannot be reduced to simple access predicates. Classifiers and reviewers remain fallible. Domain shifts and adaptive adversaries can change their behavior, and shared components can create correlated failure.

12.4 Simulation and ecological validity

Synthetic environments support repeatability and controlled comparisons. They may omit deployment factors such as organizational permissions, changing interfaces, and user behavior. Emulated tools can broaden exploratory coverage but may produce consequences that differ from real execution. Findings should be reported at the level supported by the environment.

12.5 Utility and human factors

Restriction can shift work to the user or increase operational cost. A guardrail that is frequently bypassed because it interrupts legitimate workflows may provide less practical protection than its laboratory result suggests. Oversight quality also depends on the information supplied to reviewers.

12.6 Status of the contribution

No implementation, experiment, released dataset, or measured result is provided in this paper. The formal argument depends on strong assumptions and is a specification aid. The evaluation protocol is proposed for future study. Publication of this document does not imply peer review or independent certification.

13. Discussion and research agenda

The framework places authority at the boundary between generated decisions and executable effects. This does not reduce the importance of improving model behavior. A model that identifies instruction conflicts and plans within scope can reduce interventions and support useful completion. Runtime controls provide a separate mechanism for properties that can be enforced directly.

Several research directions follow. One concerns the portability of policies across tool interfaces. Another concerns the cost of maintaining provenance through summaries and memory. A third examines approval interfaces that communicate concrete effects without overwhelming users. A fourth studies whether constrained alternatives help an agent continue productively after a denied action.

Adaptive evaluation remains essential. A defense's documented design can guide an attacker toward its weakest boundary, and changes in tool composition can create new effects. Future work should examine both component behavior and system-level consequences under these conditions.

Training data can support the model side of the architecture through examples that distinguish authority, scope, and permissible continuation. Runtime policy can govern the action side. Their interaction is a research question: improved model behavior may reduce enforcement burden, while reliable enforcement can contain errors that the model still makes.

The appropriate evidence is therefore multidimensional. Guardrail research should show which effects are prevented, which legitimate tasks remain achievable, how often intervention is required, and what assumptions bound the result.

14. Conclusion

Guardrails for tool-using AI agents require a specification of what is being controlled. Semantic screening, instruction authority, scoped authorization, runtime mediation, and oversight perform different functions and should be evaluated accordingly.

This paper proposes an action-centered systems framework that links tool proposals to authorization and environmental state. Its conditional safety analysis makes explicit the requirements of complete mediation, sound preservation checks, faithful execution, and a bounded environment. Its evaluation protocol pairs security outcomes with authorized task completion and intervention cost.

The resulting research agenda is practical: construct environments in which the distinction between allowed and prohibited effects is observable, then test the configured agent system under both benign and adversarial conditions. Custom environment work can support that agenda when task scope, policy, and execution assumptions are defined precisely.

Dependable guardrails emerge from the relationship between model behavior and enforced boundaries. Evidence about that relationship should govern the strength of any safety claim.

Declarations

Authorship and affiliation: Shehryar Awan, CTO, Deaimer.

Competing interest disclosure: The author is affiliated with Deaimer, which offers datasets and custom AI-agent environments. The proposed environment applications may be commercially relevant to that work.

Data and code availability: No new experimental dataset or implementation accompanies this conceptual paper. Referenced research resources are available through their respective authors and publications.

Ethical scope: Illustrative scenarios use fictional resources and synthetic data. The paper contains no attack payloads, operational misuse instructions, customer information, or private task content.

References

1. Inan, H., et al. (2023). Llama Guard: LLM-based Input-Output Safeguard for Human-AI Conversations. arXiv:2312.06674.
<https://arxiv.org/abs/2312.06674>
2. Sharma, M., et al. (2025). Constitutional Classifiers: Defending against Universal Jailbreaks across Thousands of Hours of Red Teaming. arXiv:2501.18837.
<https://arxiv.org/abs/2501.18837>
3. Wallace, E., Xiao, K., Leike, R., Weng, L., Heidecke, J., and Beutel, A. (2024). The Instruction Hierarchy: Training LLMs to Prioritize Privileged Instructions. arXiv:2404.13208.
<https://arxiv.org/abs/2404.13208>
4. Zhan, Q., Liang, Z., Ying, Z., and Kang, D. (2024). InjecAgent: Benchmarking Indirect Prompt Injections in Tool-Integrated Large Language Model Agents. arXiv:2403.02691.
<https://arxiv.org/abs/2403.02691>
5. Debenedetti, E., et al. (2024). AgentDojo: A Dynamic Environment to Evaluate Prompt Injection Attacks and Defenses for LLM Agents. arXiv:2406.13352.
<https://arxiv.org/abs/2406.13352>
6. Debenedetti, E., et al. (2025). Defeating Prompt Injections by Design. arXiv:2503.18813, revised version 2.
<https://arxiv.org/abs/2503.18813>
7. Saltzer, J. H., and Schroeder, M. D. (1975). The Protection of Information in Computer Systems. Proceedings of the IEEE, 63(9), 1278-1308.
<https://web.mit.edu/Saltzer/www/publications/protection/>
8. Andriushchenko, M., et al. (2025). AgentHarm: A Benchmark for Measuring Harmfulness of LLM Agents. International Conference on Learning Representations.
<https://arxiv.org/abs/2410.09024>
9. Ruan, Y., et al. (2024). Identifying the Risks of LM Agents with an LM-Emulated Sandbox. International Conference on Learning Representations.
<https://arxiv.org/abs/2309.15817>
10. Autio, C., et al. (2024). Artificial Intelligence Risk Management Framework: Generative Artificial Intelligence Profile. NIST AI 600-1.
<https://doi.org/10.6028/NIST.AI.600-1>
11. Deaimer. RL Environments & Coding-Agent Evaluations. Public offering description. Accessed 14 October 2026.
<https://www.deaimer.com/rl-environments>
12. Deaimer. Private Coding-Agent Datasets for Licensing. Public offering description. Accessed 14 October 2026.
<https://www.deaimer.com/datasets>