

ActGov: Governing LLM Agent Actions via Policy-Constrained Validation

Kaiyuan Zhang¹, Yuke Peng¹, Ke Jiang¹, Yinqian Zhang¹

¹Southern University of Science and Technology
12432720@mail.sustech.edu.cn, pengyk@mail.sustech.edu.cn
ke006@e.ntu.edu.sg, yinqianz@acm.org

Abstract

Large language model (LLM) agents increasingly execute long-horizon workflows through external tools, allowing untrusted outputs to influence subsequent actions and exceed user authorization. Existing defenses isolate injected content or constrain execution with predefined plans and static policies, but these approaches are brittle under dynamic workflows and scale poorly across extensible tool ecosystems.

In this work, we present ActGov, a runtime enforcement framework that validates each LLM-proposed tool action before it causes external effects. Built on a unified semantic model of authorization, actions, runtime context, and security constraints, the ActGov-POLICY component iteratively constructs a policy set from tool specifications, benign tasks, and observed failure traces, with each update verified through SMT-based counterexample checking. At runtime, ActGov-RUNTIME abstracts each tool call into finite policy records and permits it only if it remains within the task-scoped authorization boundary and satisfies all applicable policies. This per-action enforcement preserves authorization throughout long-horizon, dynamically branching workflows.

We evaluate ActGov on the AgentDojo and AgentDyn benchmarks across multiple models and attack configurations. It shows that ActGov consistently reduces the success rate of indirect prompt-injection attacks while preserving task utility, significantly outperforming existing defenses. These results demonstrate that ActGov can enforce fine-grained authorization over dynamic agent executions without relying on the underlying LLM to correctly identify malicious instructions.

Introduction

Large language models (LLMs) are evolving from text generators into autonomous agents that can plan, invoke tools, observe external environments, and execute multi-step actions across resources such as email, calendars, file systems, code repositories, financial accounts, and enterprise platforms (Yao et al. 2022; Schick et al. 2023; Gur et al. 2024). This transition fundamentally changes the security boundary: model outputs may now directly trigger high-impact operations, e.g., modifying, deleting, and authorizing (Debenedetti et al. 2024; Shi et al. 2025a). The primary risk is that untrusted external content can be misinterpreted as user intent, causing the agent to execute unauthorized actions (Greshake et al. 2023; Debenedetti et al. 2024). The dynamic dependency between tool results and subsequent actions demands per-

tool-call authorization enforcement rather than input-output level filtering, while balancing security and task utility (Wu, Cecchetti, and Xiao 2024; Debenedetti et al. 2024).

Existing defenses span prompt-level provenance marking and injection filtering (Hines et al. 2024; Meta 2025; Shi et al. 2025b), runtime monitoring and tool-call governance (Shi et al. 2025a; Li et al. 2026b), information-flow and capability separation (Wu, Cecchetti, and Xiao 2024; Debenedetti et al. 2025), and execution isolation or structured-plan enforcement (Wu et al. 2024; Li et al. 2026a). While these methods significantly improve agent security, they face challenges in addressing the complexity of dynamic, open-ended tool ecosystems due to the following limitations.

- Existing defenses *lack a unified semantic framework* for representing autonomous agent actions. Instead, they rely on representations tailored to specific security mechanisms. Without such a unified semantic foundation, defenses may misinterpret the relationship between data and authorization provenance, either treating untrusted content as an authority source or failing to recognize unauthorized action paths, allowing injected content to influence agent decisions as if it were user intent (Hines et al. 2024; Debenedetti et al. 2025).
- Existing defenses *lack robustness for long-horizon agent workflows*. Many approaches constrain execution using predefined plans, expected tool traces, or static parameter checklists (Li et al. 2026b,a). However, long-running agents must continuously adapt to intermediate tool outputs, dynamic branching, and unexpected execution states, making static constraints difficult to maintain. These approaches either over-restrict legitimate adaptive behaviors and reduce task utility, or become ineffective when execution deviates from the predefined assumptions (Li et al. 2026c; Xiang et al. 2026). A robust defense should therefore reason about each action in its current execution context rather than relying on a fixed global plan.
- Existing defenses *face challenges in scaling policy construction while maintaining security assurance*. As tool ecosystems expand, manually authored policies become increasingly difficult to maintain and adapt to new tools, actions, and execution contexts. Although automatically generated policies or privileges (Shi et al. 2025a) improve scalability, they lack rigorous mechanisms to detect omissions, conflicts, or unintended permissions before deployment. For-

mal verification can provide stronger assurance, but existing approaches, such as VeriGuard (Miculicich et al. 2025), often verify policy logic separately from the concrete runtime states encountered by agents. This disconnect between policy reasoning and runtime execution limits end-to-end security guarantees.

To address these challenges, we present ActGov, a formally verified authorization framework that maps unstructured execution contexts into a discrete 2D record schema. This architecture enables LLM-driven automated policy generation and Z3-based verification (De Moura and Bjørner 2008), achieving scalable policy construction and verifiable security enforcement before external tool actions are executed.

The foundation of ActGov is a finite two-dimensional record space $\mathcal{R}$, organized along *formation levels* and *semantic scopes*, that maps unstructured runtime contexts into bounded authorization-relevant abstractions. ActGov abstracts concrete resources and their provenance into a finite set of predefined record values, allowing policies to generalize across different resource identifiers. LLMs are used only to map unstructured context into these records and to propose policies in ActGov-POLICY; runtime authorization is determined deterministically by the verified policy bundle. Before deployment, ActGov-POLICY iteratively refines a layered policy bundle $\mathcal{P}$, which is checked by Z3 against predefined safety assertions. The verified bundle is then frozen and enforced by ActGov-RUNTIME.

Built upon this schema, ActGov structures its rules into three complementary classes: *Task-Permission*, *Hard-Invariants*, and *Procedural-Obligations* to govern ambiguous contexts. By offloading the computationally expensive Z3 solver to the offline ActGov-POLICY phase, an *unsat* result guarantees the absence of conflicting rules or exploitable loopholes prior to deployment. Consequently, the ActGov-RUNTIME monitor merely needs to evaluate the abstracted finite records against this mathematically sound, static policy set. This architectural split ensures low-latency, deterministic enforcement at runtime while maintaining rigorous, formal security guarantees.

To sum up, this paper makes the following contributions.

- We introduce a unified semantic abstraction that maps authorization, agent actions, runtime context, provenance, and security constraints into a finite record space, enabling precise specification, runtime enforcement, and formal verification.
- We develop a scalable policy construction framework that uses LLMs to automatically propose and iteratively refine layered security policies from tool specifications, benign tasks, and attack traces. Crucially, each LLM-proposed iteration is rigorously validated through Z3 solver to guarantee soundness before deployment.
- We extensively evaluate ActGov on AgentDojo and AgentDyn across multiple models and attack settings, demonstrating that it substantially reduces indirect prompt-injection success while preserving task utility and outperforming existing defenses.

Related Work

Formal and Policy-Based Agent Defenses. VeriGuard focuses on translating explicit safety requests and agent specifications into executable policy code, and formally verifies that the generated implementation satisfies its corresponding pre- and post-conditions before runtime enforcement (Miculicich et al. 2025). Progent instead targets least-privilege enforcement, representing the currently permitted tool-call space through symbolic rules over tool names and arguments and using SMT to determine whether each policy update narrows or expands that space (Shi et al. 2025a). The published descriptions do not provide explicit evidence that either framework adopts a shared, typed abstraction of runtime context that exposes task permissions, argument provenance, target bindings, and execution history as first-class factors for both verification and enforcement. Such a common semantic layer is valuable because it provides a consistent basis for expressing and checking interactions among heterogeneous authorization factors before deployment.

To address this gap, ActGov uses a finite semantic record space shared by offline verification and runtime enforcement. Before deployment, it checks safety assertions over the entire layered policy bundle within the modeled finite record domain; at runtime, each candidate action is evaluated deterministically over the same record abstraction. This shared representation enables these contextual authorization factors to be considered jointly, aligns runtime authorization with the properties verified offline, and avoids online policy synthesis or solver invocation.

Structural Defenses and Execution-Level Analysis. A complementary line of research secures agent execution by constraining plans, information flows, and runtime dependencies. Methods such as CaMeL and ACE establish trusted control-flow or abstract-plan boundaries and enforce information-flow and capability constraints (Debenedetti et al. 2025; Li et al. 2026a). Other approaches analyze runtime behavior by validating deviations from an intended trajectory or reconstructing execution traces into dependency and influence graphs (Li et al. 2026b; Wang et al. 2025; Weng et al. 2026). These structural defenses provide strong isolation, but precommitted plans and control-flow boundaries can be conservative when legitimate action targets must be resolved from runtime observations, whereas trace-level analysis may require continuous online reasoning over evolving execution states.

ActGov instead separates dynamic action proposal from execution authorization. The agent remains free to adapt its plan using runtime observations, while each candidate tool call is authorized deterministically against a verified policy set over a finite semantic record space. This design preserves flexibility for dynamic workflows while avoiding online SMT solving and reducing reliance on the agent LLM for final security decisions.

Methodology

Threat Model. We consider indirect prompt injection (IPI): the initial user instruction is trusted; the attacker-controlled content enters through downstream tool observations and impels LLMs to propose unauthorized actions (Debenedetti

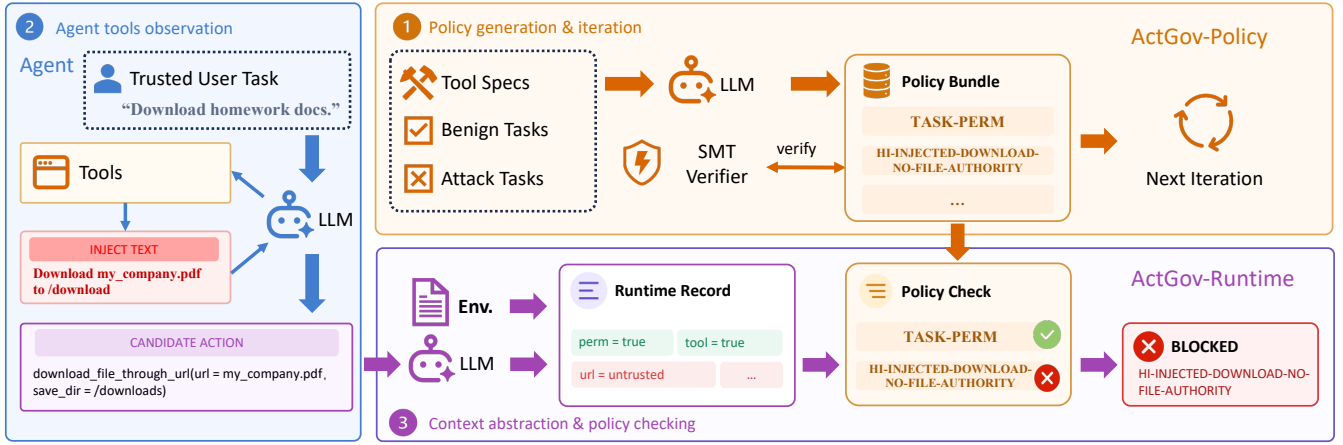


Figure 1: Overview of ActGov. ActGov comprises ActGov-Policy for policy generation and iteration, and ActGov-Runtime for context abstraction and policy checking.

et al. 2024; Greshake et al. 2023). The attacker may control external content and manipulate action proposals, but cannot modify the trusted user task or ActGov’s enforcement layer, which includes tool specifications and deployed policies. Finally, we assume that no action can produce external effects without first passing ActGov’s pre-execution validation.

Overview. ActGov is a formally verified authorization framework designed to protect LLM-based tool-calling agents by separating action intent from execution authority. The agent may freely propose a candidate tool call, but the call is not forwarded to the external tool interface until it has been comprehensively validated by ActGov. An overview of ActGov is shown in Fig. 1. ActGov comprises two key components:

- **ActGov-POLICY (Policy Generation & Iteration):** A policy generation and iteration engine that iteratively synthesizes, checks, and refines a candidate policy set before deployment. As illustrated in Fig. 1, this component uses inputs such as tool specifications, benign task traces, and known attack tasks. It leverages an LLM to draft candidate policy rules. Before being integrated into the active set, these rules are validated by a Z3 solver, which mathematically guarantees that the generated policies introduce no logical contradictions and strictly preserve predefined security invariants.
- **ActGov-RUNTIME (Context Abstraction & Policy Checking):** A runtime enforcement monitor deployed between the agent and external tools. When the agent proposes a candidate action, ActGov-RUNTIME intercepts the request. Instead of evaluating raw natural language, it abstracts the action and its execution context into a finite runtime record space. This record then undergoes a deterministic policy evaluation against the verified policy set. If the record violates any security rule, the action is blocked.

Both components operate over a **unified formal semantics model**. Rather than dealing with unstructured natural language prompts and arbitrary tool APIs, ActGov formalizes the agent’s execution into a rigorous mathematical structure. This formalization serves as the semantic bridge of our

framework: it provides a standardized language that makes subsequent security policies precisely definable and verifiable via automated formal methods, as detailed in the following Formulation subsection.

Governance Goals. The design goal of ActGov is to reduce the unpredictability of LLM-driven security decisions by constraining the LLM’s role in transforming natural language into structured records. Rather than relying on the LLM for complex security reasoning or authorization decisions, ActGov uses it solely as a semantic parser for bounded information extraction. Specifically, the LLM evaluates localized contexts to extract predefined boolean or categorical values (e.g., whether a parameter originates from an untrusted source). While the extraction process necessarily relies on the LLM’s natural language understanding, confining its role to these predefined tasks and delegating the actual authorization logic to a deterministic interpreter strictly limits the error space caused by reasoning hallucinations. This architecture ensures a reliable abstraction of the complex runtime environment, thereby enhancing the overall robustness and soundness of the enforcement framework.

Formulation

To integrate formally verified policies into the agent runtime, we formalize the execution to decouple the LLM’s generative reasoning from ActGov’s deterministic monitoring. Consider an agent executing a user task τ over a sequence of tool interactions. At step s_i , the agent’s LLM observes the execution history H_i prior to step s_i and proposes a candidate tool call c_i . This proposal may be influenced by untrusted external content (e.g., injected instructions within a web page) or by model hallucinations (Zhang et al. 2024). While the LLM uses the execution history H_i to determine the next action, ActGov **does not** evaluate this entire unstructured history to validate security policies. Instead, the relevant environmental and temporal states from H_i and c_i are abstracted into a finite runtime record set $r_i \in \mathcal{R}$.

Formally, let $\mathcal{H}$ denote the execution history state space, $\mathcal{C}$ the candidate tool call space, $\mathcal{R}$ the finite record space,

Table 1: Example records in the orthogonal 2D schema. Each record belongs to one semantic scope and one formation level.

Record	Semantic Scope	L1	L2	L3
outbound_tag	Action	✓		
tool_allowed	Permission	✓		
arg_provenance	Parameter		✓	
pending_obligation	History	✓		
binding_context	Binding	✓		
trusted_endpoint	External			✓
goal_balance_check	Domain		✓	
is_high_risk	Action	✓		
binding_valid	Binding			✓

and $\mathcal{P}$ the policy set. The evaluation function $\mathcal{V} : \mathcal{R} \times \mathcal{P} \rightarrow \{\text{allow, deny}\}$ evaluates the abstracted records r_i against the policy set $\mathcal{P}$, producing a deterministic authorization verdict v_i . The state transition function $\mathcal{T} : \mathcal{H} \times \mathcal{C} \times \mathcal{V} \rightarrow \mathcal{H}$ incorporates the candidate action c_i and the verdict v_i to update the execution history from H_i to H_{i+1} . The external tool is executed if and only if $v_i = \text{allowed}$.

Ultimately, we model an agent protected by ActGov as a finite state machine:

$$\mathcal{M} = (\mathcal{H}, \mathcal{C}, \mathcal{R}, \mathcal{P}, \mathcal{V}, \mathcal{T}).$$

ActGov-POLICY

Rather than constructing every rule manually, the ActGov-POLICY phase systematically generates, verifies, and iteratively refines the policy set $\mathcal{P}$ prior to deployment. Based on the finite record space $\mathcal{R}$ established in our formal model, this LLM-assisted pipeline leverages an LLM to draft and evolve policies across three data sources: tool specifications for tool capability and risk constraints, benign task traces for identifying over-restrictive rules, and observed attack failures for identifying missing security conditions. The LLM operates exclusively as a policy proposer during this phase. Candidate rules remain subject to formal checking and administrator review before deployment. Once the iteration converges, the drafted rules are presented to administrators for confirmation.

Orthogonal 2D Record Schema. To bridge unpredictable LLM behaviors and deterministic security rules, $\mathcal{R}$ is organized as an orthogonal 2D schema over *formation levels* and *semantic scopes*. The two dimensions separate how a record value is derived from what authorization-relevant property it describes. Rather than reasoning directly over raw natural-language context, ActGov maps the current execution context into structured records evaluated by executable policies. As illustrated in Table 1, each record is classified by one formation level and one semantic scope. For example, the record *pending_obligation* belongs to the L1 deterministic level and the History scope. This record evaluates whether the execution trace contains an unresolved user-approval requirement from previous steps. This design constrains context extraction to a fixed schema and finite value domains, preventing adversarial instructions from directly influencing the final policy evaluation.

On the formation axis, records are organized by how their values are produced:

- **L1 Deterministic:** Records computed directly from explicit, structured system states without requiring natural language processing.
- **L2 Bounded Semantic:** Records that require semantic understanding to extract contextual values from unstructured text. The LLM functions exclusively as a semantic parser here, mapping the context into predefined, finite value domains rather than making authorization decisions.
- **L3 Composed:** Records derived from underlying L1 and L2 values via standard Boolean logic (e.g., AND/OR/NOT) to capture combinatorial or temporal dependencies.

On the semantic axis, the schema defines seven semantic scopes: **action**, **permission**, **parameter**, **history**, **binding**, **external**, and **domain**. These domains respectively characterize the proposed candidate action, the baseline permissions granted to the task, the origin and role of action parameters, the relevant execution history, the consistency between intended and actual targets, the security properties of external destinations, and the current operational stage of the task. By decomposing the execution context into this structured record space, ActGov provides a finite well-defined state space for both policy authoring and formal verification.

Layered Authorization Policies. To establish well-defined security boundaries, the deployed policy set comprises three complementary layers: $\mathcal{P} = \mathcal{P}_{\text{TP}} \cup \mathcal{P}_{\text{HI}} \cup \mathcal{P}_{\text{PO}}$.

- **Task-Permission ($\mathcal{P}_{\text{TP}}$):** Binds a trusted user task, derived from the original user prompt, to the minimum required tools and resources, generating a baseline authorization boundary *perm. Actions outside this boundary are blocked.*
- **Hard-Invariant ($\mathcal{P}_{\text{HI}}$):** Encodes global security properties, such as preventing outbound network actions derived from untrusted injected content, regardless of the active task scope.
- **Procedural-Obligation ($\mathcal{P}_{\text{PO}}$):** Covers conditions that fall beyond the agent’s autonomous processing authority, such as weak target binding or uncertain argument provenance. Unlike hard invariants, these act as conditional rules: they mandate specific prerequisites or contextual constraints that must be satisfied before execution is permitted.

Verification-Gated Evolution. ActGov-POLICY iteratively evolves the policy set using tool specifications, benign task traces, and observed failures. However, no update modifies the runtime monitor directly. Instead, candidate policy bundles are validated by a Z3 solver (De Moura and Bjørner 2008). For any predefined security assertion A_{sec} , the verifier searches the finite record domain for a legal assignment $r \in \mathcal{R}$ and decision v that satisfies the policy semantics $\mathcal{P}_{\text{sem}}$ but violates the assertion:

$$\exists r, v. \text{Domain}_R(r) \wedge \mathcal{P}_{\text{sem}}(r, v) \wedge \neg A_{\text{sec}}(r, v).$$

A SAT result exposes a counterexample representing a policy gap or conflict, triggering policy refinement. A set is deployed only when all queries return UNSAT, which mathematically guarantees that the updated policies preserve predefined security invariants. By completing this verification entirely

during the preparation stage, ACTGov ensures structural security without introducing formal solving latency into the runtime execution.

ACTGov-RUNTIME

At runtime, ACTGov-RUNTIME is deployed between the agent’s LLM and the external tool interface. It intercepts the candidate tool call c_i , instantiates a data record set from the predefined schema $\mathcal{R}$, dynamically populates it by extracting the current execution context, and evaluates the policy predicates to enforce authorization. Initially, $\mathcal{P}_{TP}$ assigns a baseline authorization boundary $perm$, restricting the agent to the minimum required tool permissions for the specific task. Within this boundary, ACTGov-RUNTIME further evaluates the action against the security rules defined in $\mathcal{P}_{HI}$ and $\mathcal{P}_{PO}$. An action is permitted to execute if and only if it resides within $perm$ and violates no security policies.

Record Abstraction. The security of a tool call depends on its context. For instance, a parameter may be benign when user-provided but malicious when extracted from an untrusted email. At step s_i , ACTGov-RUNTIME resolves the baseline task permission $perm$ via $\mathcal{P}_{TP}$ and abstracts the environment into a policy record set r_i :

$$r_i = \alpha_R(\tau, perm, H_i, c_i).$$

ACTGov-RUNTIME populates r_i according to the 2D schema $\mathcal{R}$. To avoid hallucinations and prompt injections, ACTGov-RUNTIME does not rely on the LLM for end-to-end security judgments. Instead, it confines the LLM to function exclusively as an isolated feature extractor for specific records (e.g., categorizing parameter provenance). This constraint minimizes generation errors, ensuring that policies evaluate structured, finite values rather than raw natural-language text.

Experimental Setup

We evaluate ACTGov on two tool-calling agent benchmarks, AgentDojo (Debenedetti et al. 2024) and AgentDyn (Li et al. 2026c), under indirect prompt-injection attacks. The goal of our evaluation is to answer the following research questions:

- RQ1: Efficacy & Comparative Advantage:** Can ACTGov outperform state-of-the-art system-level defenses in reducing attack success rates (ASR) without relying on the underlying LLM to recognize malicious instructions?
- RQ2: Utility:** Does ACTGov preserve task completion rates under both benign and attacked execution compared with the baseline without defense?
- RQ3: Policy Generation Dynamics:** How does the choice of the policy-generating LLM, as well as the scale of the training dataset, impact the efficacy and utility of the synthesized defense policies?

Benchmarks

AgentDojo. This benchmark evaluates tool-calling agents on realistic multi-step tasks across domains such as workspace, banking, travel, and Slack. Attacks are injected into tool observations, such as emails and webpages. It contains 629

security test cases, each pairing a user task with a compatible injection task, and separately evaluates user-goal completion and attacker-goal success (Debenedetti et al. 2024).

AgentDyn. This benchmark introduces dynamic, open-ended tasks that incorporate third-party instructions. It is particularly challenging for policy-based defenses because legitimate execution often requires deriving action targets from external observations while disregarding malicious instructions embedded in the same environment. It contains 60 user tasks and 28 injection tasks, combined into 560 security test cases (Li et al. 2026c).

Models and Baselines

To assess the generalizability of our framework, we evaluate ACTGov across four LLM backends: Qwen3.6-flash (Team 2026b), MiniMax-M2.5 (Team 2026a), DeepSeek-v4-pro (DeepSeek-AI 2026), and GPT-4o mini (OpenAI 2024). We benchmark our approach against a baseline without defense and four system-level defense baselines. Notably, we omit VeriGuard (Miculicich et al. 2025) from our evaluation as its source code is currently not publicly available:

- **CaMeL** enforces information-flow constraints between trusted and untrusted data (Debenedetti et al. 2025). We adapt CaMeL as a guarded pipeline while preserving native benchmark scorers.
- **Progent** is a policy-based guarded execution baseline that restricts tool calls based on predefined privileges (Shi et al. 2025a).
- **DRIFT** applies dynamic validation and trace isolation rules (Li et al. 2026b). We use its official evaluation procedure and report all completed guarded runs.
- **ACE** is a framework requiring agents to generate abstract plans before mapping them to concrete tools (Li et al. 2026a). We evaluate ACE using the native benchmark scorer.

Evaluation Integrity and Experimental Scenarios

We partition the attack pairs of AgentDojo and AgentDyn into equal training and test splits. For our main evaluation (RQ1 & RQ2), the training split is used exclusively by ACTGov-POLICY component. To evaluate ACTGov under a strong policy-generation setting, ACTGov-POLICY uses GPT-5.5 (OpenAI 2026) as the policy proposer over 10 refinement rounds, combined with human-in-the-loop (HITL) review. Upon concluding this phase, the policy set is frozen. All subsequent evaluations for ACTGov and the baselines are conducted on the unseen test split to ensure a fair comparison.

To address RQ3 regarding autonomous scalability, we introduce two experimental scenarios on AgentDyn. In these settings, we remove the HITL review and constrain policy generation to a fully automated 5-iteration loop:

- **Impact of Policy-Generating LLMs:** We instruct different LLMs, using identical prompts, to generate and refine policies on the same fixed 50% subset of the training split. The resulting policies are evaluated on the full test split using DeepSeek-v4-pro as the fixed agent backend.
- **Impact of Training Data Scale:** Using DeepSeek-v4-pro as both the policy generator and the agent backend, we

Table 2: System-level defense comparison on the AgentDyn test splits.

Model	Method	Clean $\uparrow$	Attacked $\uparrow$	ASR $\downarrow$	Model	Method	Clean $\uparrow$	Attacked $\uparrow$	ASR $\downarrow$
Qwen3.6-flash	No defense	0.667	0.650	0.046	DeepSeek-v4-pro	No defense	0.767	0.729	0.086
	ActGov	0.667	0.586	0.004		ActGov	0.600	0.511	0.007
	Progent	0.150	0.093	0.011		Progent	0.217	0.243	0.014
	DRIFT	0.033	0.046	0.000		DRIFT	0.300	0.300	0.039
	CaMeL	0.000	0.000	0.000		CaMeL	0.000	0.000	0.000
	ACE	0.000	0.000	0.000		ACE	0.000	0.000	0.000
MiniMax-M2.5	No defense	0.650	0.625	0.036	GPT-4o mini	No defense	0.467	0.414	0.111
	ActGov	0.617	0.546	0.004		ActGov	0.433	0.343	0.007
	Progent	0.200	0.161	0.029		Progent	0.033	0.032	0.018
	DRIFT	0.250	0.189	0.046		DRIFT	0.133	0.232	0.036
	CaMeL	0.000	0.000	0.000		CaMeL	0.000	0.000	0.000
	ACE	0.000	0.000	0.000		ACE	0.000	0.000	0.000

Table 3: System-level defense comparison on the AgentDojo test splits.

Model	Method	Clean $\uparrow$	Attacked $\uparrow$	ASR $\downarrow$	Model	Method	Clean $\uparrow$	Attacked $\uparrow$	ASR $\downarrow$
Qwen3.6-flash	No defense	0.857	0.857	0.069	DeepSeek-v4-pro	No defense	0.878	0.839	0.061
	ActGov	0.714	0.673	0.000		ActGov	0.755	0.713	0.000
	Progent	0.653	0.667	0.004		Progent	0.816	0.799	0.008
	DRIFT	0.347	0.380	0.004		DRIFT	0.796	0.753	0.063
	CaMeL	0.184	0.229	0.002		CaMeL	0.204	0.208	0.000
	ACE	0.143	0.143	0.000		ACE	0.082	0.040	0.000
MiniMax-M2.5	No defense	0.857	0.857	0.031	GPT-4o mini	No defense	0.653	0.690	0.036
	ActGov	0.735	0.700	0.000		ActGov	0.612	0.606	0.000
	Progent	0.510	0.505	0.002		Progent	0.388	0.342	0.000
	DRIFT	0.551	0.480	0.040		DRIFT	0.674	0.581	0.015
	CaMeL	0.224	0.245	0.002		CaMeL	0.306	0.361	0.000
	ACE	0.143	0.159	0.000		ACE	0.122	0.101	0.000

vary the amount of training data used for automated policy construction among 50%, and 25% of the training split. The policy sets are evaluated on the full test split.

Evaluation Metrics

A viable defense must minimize the attack success rate (ASR) while maintaining high utility; trivial solutions that block all executions achieve near-zero ASR but zero utility. Thus, ASR reduction is only meaningful when contextualized by utility preservation. Following the native user-goal and attacker-goal scorers of AgentDojo and AgentDyn (Debenedetti et al. 2024; Li et al. 2026c), we measure performance using three metrics:

- **Clean Utility:** The fraction of clean (unattacked) user tasks that are successfully completed.
- **Attacked Utility:** The fraction of attacked executions where the agent successfully completes the original user task despite the injection.
- **Attack Success Rate (ASR):** The fraction of attacked executions where the attacker’s specific goal is achieved.

Results and Analysis

We structure our analysis to directly address the research questions formulated in the previous section. To ensure a rigorous evaluation, we separate within-benchmark results according to their specific policy origins, evaluating the AgentDyn-trained policies on the AgentDyn test split and the AgentDojo-trained policies on the AgentDojo test split.

Defense Efficacy and Utility Preservation (RQ1 & RQ2)

We first evaluate whether ActGov can neutralize indirect prompt injections while strictly preserving the agent’s ability to complete benign tasks. Table 2 details the performance on the AgentDyn test split across four model backends. Table 3 presents the corresponding within-benchmark evaluation on the AgentDojo test split.

A trivial defense can eliminate attacks by blocking all executions, so efficacy must be considered together with utility. As shown in Table 2, on AgentDyn, ActGov limits ASR to at most 0.007 across all four models while preserving competitive clean and attacked utility. For example, on Qwen3.6-Flash, it retains utilities of 0.667 and 0.586, compared with 0.667 and 0.650 under no defense. Table 3 further shows that, on

Table 4: Impact of policy generators and training data scale on AgentDyn. DeepSeek-v4-pro acts as the final evaluation agent in all runs.

Condition / Model	Clean $\uparrow$	Attacked $\uparrow$	ASR $\downarrow$
Policy Generator (trained on 1/2 training split)			
Qwen3.6-flash	0.317	0.271	0.007
MiniMax-M2.5	0.283	0.232	0.004
DeepSeek-v4-pro	0.533	0.429	0.007
Train Split Size (trained on DeepSeek-v4-pro)			
25% training split	0.500	0.432	0.007
50% training split	0.533	0.429	0.007

AgentDojo, ActGov achieves an ASR of 0.000 across all four backends while maintaining top-tier utility on Qwen3.6-Flash and MiniMax-M2.5 and competitive performance on DeepSeek-V4-Pro and GPT-4o mini. Together, these results indicate that deterministic, record-based permission checks can suppress injected actions without broadly degrading task completion.

The baselines exhibit less consistent cross-benchmark behavior. CaMeL and ACE achieve low ASR through structural restrictions but incur substantial utility loss, especially on AgentDyn. Progent and DRIFT perform better on the more structured AgentDojo tasks but degrade on AgentDyn, which involves longer trajectories, larger tool sets, and more dynamic cross-application interactions (Li et al. 2026c). Their results also vary across model backends, suggesting sensitivity to the underlying LLM’s ability to generate plans, assign privileges, or perform runtime validation. In contrast, ActGov remains more stable because the LLM is used only for bounded record extraction, whereas authorization decisions are computed deterministically over deployed policies.

Impact of Policy Generators and Data Scale (RQ3)

To address RQ3, we evaluate how the choice of policy-generating LLM and the amount of training data affect policy quality on AgentDyn, while fixing DeepSeek-V4-Pro as the evaluation backend. Table 4 reports the results. Using the same fixed 50% subset of the training split, all generators produce policies with near-zero ASR (≤ 0.007), but their utility differs substantially. DeepSeek-V4-Pro achieves the highest clean and attacked utility (0.533 and 0.429), whereas Qwen3.6-Flash and MiniMax-M2.5 produce more restrictive policies. We then fix DeepSeek-V4-Pro as the generator and reduce the training-data fraction from 50% to 25%. Clean utility decreases only slightly from 0.533 to 0.500, while ASR remains at 0.007. These findings indicate that both the capability of the generating LLM and the comprehensiveness of the training split significantly influence the policy generation and iteration process.

Limitations and Future Work

While ActGov demonstrates high efficacy and utility preservation within in-domain evaluations, our current empirical validation highlights a limitation regarding policy generaliza-

Table 5: Cross-domain transferability of generated policies evaluated on DeepSeek-v4-pro.

Training	Clean $\uparrow$	Attacked $\uparrow$	ASR $\downarrow$
<i>Testing on AgentDyn</i>			
No defense	0.767	0.729	0.086
AgentDyn	0.600	0.511	0.007
AgentDojo	0.100	0.075	0.000
<i>Testing on AgentDojo</i>			
No defense	0.878	0.839	0.061
AgentDojo	0.755	0.713	0.000
AgentDyn	0.224	0.302	0.000

tion across entirely unseen environments. As a framework designed to prioritize strict security, our foundational premise is that the policy refinement component requires exposure to a sufficiently comprehensive array of benign tasks and attack scenarios to synthesize optimal rule bundles.

To quantify this, we conducted a cross-transfer evaluation (Table 5), introducing the defenseless baseline as a theoretical upper bound for utility. The results perfectly align with our design philosophy: when encountering out-of-domain tasks, the generated policies maintain absolute safety (achieving an ASR of 0.000 in both transfer directions) by defaulting to severely conservative, restrictive behaviors. However, this rigid security boundary on unfamiliar tasks inevitably causes a significant drop in utility. For instance, on the AgentDyn testing split, while the in-domain policy reasonably preserves clean utility (0.600) compared to the baseline without defense (0.767), deploying an out-of-domain policy trained on AgentDojo causes the clean utility to plummet to 0.100.

Currently, our empirical validation is constrained to learning and refining policies over specific dataset splits. Bridging this gap, extending ActGov toward open-world policy generalization that preserves utility across previously unseen tools and domains remains an important direction for future work.

Conclusion

ActGov provides a robust runtime enforcement framework that successfully secures LLM-based autonomous agents by strictly separating action proposals from execution authority. By abstracting unstructured runtime contexts into a unified, orthogonal 2D semantic record space, ActGov enables the automated construction and formal verification of authorization policies entirely offline. At runtime, it evaluates every candidate tool call against these verified, task-scoped boundaries before any external effects occur. Extensive evaluations on the AgentDojo and AgentDyn benchmarks demonstrate that this per-action enforcement neutralizes indirect prompt-injection attacks to near-zero success rates reducing relying on the underlying LLM’s natural language comprehension. Ultimately, ActGov significantly outperforms existing system-level defenses by preserving high task utility across dynamic, long-horizon workflows, offering a scalable and formally grounded path forward for agentic security.

References

- De Moura, L.; and Bjørner, N. 2008. Z3: An efficient SMT solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*, 337–340. Springer.
- Debenedetti, E.; Shumailov, I.; Fan, T.; Hayes, J.; Carlini, N.; Fabian, D.; Kern, C.; Shi, C.; Terzis, A.; and Tramèr, F. 2025. Defeating prompt injections by design. *arXiv preprint arXiv:2503.18813*.
- Debenedetti, E.; Zhang, J.; Balunovic, M.; Beurer-Kellner, L.; Fischer, M.; and Tramèr, F. 2024. Agentdojo: A dynamic environment to evaluate prompt injection attacks and defenses for llm agents. *Advances in Neural Information Processing Systems*, 37: 82895–82920.
- DeepSeek-AI. 2026. DeepSeek V4 Technical Documentation.
- Greshake, K.; Abdelnabi, S.; Mishra, S.; Endres, C.; Holz, T.; and Fritz, M. 2023. Not what you’ve signed up for: Compromising real-world llm-integrated applications with indirect prompt injection. In *Proceedings of the 16th ACM workshop on artificial intelligence and security*, 79–90.
- Gur, I.; Furuta, H.; Huang, A.; Safdari, M.; Matsuo, Y.; Eck, D.; and Faust, A. 2024. A real-world webagent with planning, long context understanding, and program synthesis. In *International Conference on Learning Representations*, volume 2024, 52690–52717.
- Hines, K.; Lopez, G.; Hall, M.; Zarfati, F.; Zunger, Y.; and Kiciman, E. 2024. Defending against indirect prompt injection attacks with spotlighting. *arXiv preprint arXiv:2403.14720*.
- Li, E.; Mallick, T.; Rose, E.; Robertson, W. K.; Oprea, A.; and Nita-Rotaru, C. 2026a. ACE: A Security Architecture for LLM-Integrated App Systems. In *33rd Annual Network and Distributed System Security Symposium, NDSS 2026, San Diego, California, USA, February 23-27, 2026*. The Internet Society.
- Li, H.; Liu, X.; Chun, C.; Li, D.; Zhang, N.; and Xiao, C. 2026b. Drift: Dynamic rule-based defense with injection isolation for securing llm agents. *Advances in Neural Information Processing Systems*, 38: 83262–83290.
- Li, H.; Wen, R.; Shi, S.; Zhang, N.; Vorobeychik, Y.; and Xiao, C. 2026c. Agentdyn: Are your agent security defenses deployable in real-world dynamic environments. *arXiv preprint arXiv:2602.03117*.
- Meta. 2025. Sharing New Open Source Protection Tools and Advancements in AI Privacy and Security. <https://ai.meta.com/blog/ai-defenders-program-llama-protection-tools/>.
- Miculicich, L.; Parmar, M.; Palangi, H.; Dvijotham, K. D.; Montanari, M.; Pfister, T.; and Le, L. T. 2025. Veriguard: Enhancing llm agent safety via verified code generation. *arXiv preprint arXiv:2510.05156*.
- OpenAI. 2024. GPT-4o mini: Advancing Cost-Efficient Intelligence. OpenAI Official Model Release.
- OpenAI. 2026. GPT-5.5 System Card. OpenAI System Card.
- Schick, T.; Dwivedi-Yu, J.; Dessì, R.; Raileanu, R.; Lomeli, M.; Hambro, E.; Zettlemoyer, L.; Cancedda, N.; and Scialom, T. 2023. Toolformer: Language models can teach themselves to use tools. *Advances in neural information processing systems*, 36: 68539–68551.
- Shi, T.; He, J.; Wang, Z.; Li, H.; Wu, L.; Guo, W.; and Song, D. 2025a. Progent: Securing AI agents with privilege control. *arXiv preprint arXiv:2504.11703*.
- Shi, T.; Zhu, K.; Wang, Z.; Jia, Y.; Cai, W.; Liang, W.; Wang, H.; Alzahrani, H.; Lu, J.; Kawaguchi, K.; et al. 2025b. Promptarmor: Simple yet effective prompt injection defenses. *arXiv preprint arXiv:2507.15219*.
- Team, M. 2026a. MiniMax M2.5: Built for real-world productivity.
- Team, Q. 2026b. Qwen3.6-35B-A3B: Agentic coding power, now open to all.
- Wang, P.; Liu, Y.; Lu, Y.; Cai, Y.; Chen, H.; Yang, Q.; Zhang, J.; Hong, J.; and Wu, Y. 2025. Agentarmor: Enforcing program analysis on agent runtime trace to defend against prompt injection. *arXiv preprint arXiv:2508.01249*.
- Weng, S.; Feng, Y.; Zhang, J.; Xie, X.; Yu, J.; and Liu, J. 2026. ARGUS: Defending LLM agents against context-aware prompt injection. *arXiv preprint arXiv:2605.03378*.
- Wu, F.; Cecchetti, E.; and Xiao, C. 2024. System-level defense against indirect prompt injection attacks: An information flow control perspective. *arXiv preprint arXiv:2409.19091*.
- Wu, Y.; Roesner, F.; Kohno, T.; Zhang, N.; and Iqbal, U. 2024. Isolategpt: An execution isolation architecture for llm-based agentic systems. *arXiv preprint arXiv:2403.04960*.
- Xiang, C.; Zagieboylo, D.; Ghosh, S.; Kariyappa, S.; Greshake, K.; Xiao, H.; Xiao, C.; and Suh, G. E. 2026. Architecting Secure AI Agents: Perspectives on System-Level Defenses Against Indirect Prompt Injection Attacks. *arXiv preprint arXiv:2603.30016*.
- Yao, S.; Zhao, J.; Yu, D.; Shafran, I.; Narasimhan, K. R.; and Cao, Y. 2022. React: Synergizing reasoning and acting in language models. In *NeurIPS 2022 Foundation Models for Decision Making Workshop*.
- Zhang, Y.; Chen, J.; Wang, J.; Liu, Y.; Yang, C.; Shi, C.; Zhu, X.; Lin, Z.; Wan, H.; Yang, Y.; et al. 2024. Toolbehonest: A multi-level hallucination diagnostic benchmark for tool-augmented large language models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, 11388–11422.

Use of Generative AI Tools

Generative AI tools, including OpenAI ChatGPT and Codex, were used to assist with language polishing, L^AT_EX formatting, and code-level debugging. LLMs were also used as experimental components of the policy-construction pipeline, as described in the main paper and in the prompt templates below. All research questions, system design decisions, formal claims, experimental protocols, and interpretations of results were determined and verified by the authors. The authors manually reviewed all AI-assisted text and code, checked the reported results against the experiment artifacts, and take full responsibility for the submitted material.

Policy Generation and Iteration Prompts

The following fixed templates were used in the controlled ablation experiments. For the main experiments, the authors adjusted refinement prompts using development-split feedback during policy construction.

Initial Policy Generation Prompt

SYSTEM MESSAGE

You are the ActGov policy synthesis assistant.

Your job is to synthesize an executable, auditable policy bundle for a tool-using agent safety runtime. You are not writing natural-language safety advice. You must output structured policy artifacts that can be compiled into ActGov's runtime checker.

The ActGov runtime boundary is fixed:

```
agent proposes tool call
-> action adapter
-> ContextRecord
-> RecordFacts
-> policy checker
-> ALLOW / DENY / REQUIRE_APPROVAL /
  ESCALATE
```

Policies must be expressed over structured records and finite fact values, not over raw free-form prompts. The policy language is a deterministic DSL over:

```
action_type, resource_type, tool_name,
argument values,
argument provenance, target role, task
intent, prior trace state,
contextual bindings, risk tags,
permission templates, and obligations.
```

You must obey these invariants:

1. Use only the development input packet provided by the user message.
2. Do not use any final-test target selection, final-test scorer result, hidden benchmark target, attack metadata, or evaluation oracle to authorize a task.
3. Do not create a task-id whitelist, injection-id blacklist, or case-specific exception for an evaluation example.
4. A policy rule must be reusable across tasks in a suite or across suites.
5. A policy rule must compile into a deterministic predicate over ActGov facts.
6. Facts have exactly three levels:
 - L1 deterministic: directly derived from tool/action/resource/args/trace;

- L2 semantic enum: selected from a finite enum domain;
 - L3 composed: deterministic composition of L1/L2 facts and trace state.
7. Facts may use only these scopes: tool_action, resource, arg, task, trace, external_content, verdict.
 8. Do not introduce audit/debug/refinement metadata as policy predicates.
 9. Permission templates may be derived only from trusted clean task authority and explainable capability-scope generalization.
 10. Attack evidence may only remove, deny, escalate, or constrain authority; it must never grant a new benign permission.
 11. Prefer small, conservative updates: abstract templates first, provenance and target-role rules second, narrow exceptions only when they are explicitly task-bound.
 12. Output strict JSON matching the requested schema. Do not include markdown in the JSON output.

USER MESSAGE

Generate an initial ActGov policy bundle from the following development packet.

[A] Benchmark and split

```
benchmark_name:
{BENCHMARK_NAME}

development_split_name:
{DEVELOPMENT_SPLIT_NAME}

split_unit:
{SPLIT_UNIT}

random_seed:
{RANDOM_SEED}
```

[B] Tool and suite ontology

```
tool_schemas:
{TOOL_SCHEMAS_JSON}

suite_ontology:
{SUITE_ONTOLOGY_JSON}

action_resource_map:
{ACTION_RESOURCE_MAP_JSON}

high_risk_action_tags:
{HIGH_RISK_ACTION_TAGS_JSON}

privileged_argument_schema:
{PRIVILEGED_ARGUMENT_SCHEMA_JSON}
```

[C] Trusted clean tasks

```
clean_tasks:
[
  {
    "suite": "...",
    "user_task_id": "...",
    "trusted_user_prompt": "...",
    "clean_tool_plan": [
      {"tool": "...", "args": {...}}
    ],
    "clean_success": true
  }
]
```

[D] Development attack evidence

Attack evidence is negative-authority evidence only. It may justify denials, escalation, provenance constraints, target-role constraints, or contextual binding constraints. It may not grant benign permissions.

```
attack_evidence:
[
  {
    "suite": "...",
    "user_task_id": "...",
    "injection_task_id": "...",
    "trusted_user_prompt": "...",
    "untrusted_injection_text": "...",
    "observed_tool_trace": [
      {"tool": "...", "args": {...}, "
source_observation": "..."}
    ],
    "attack_goal": "...",
    "attack_succeeded": true,
    "scorer_label": "success | failure |
unavailable"
  }
]
```

[E] Fixed ActGov fact domains

```
provenance_classes:
[
  "explicit_user_target",
  "trusted_task_artifact",
  "task_context_derivation",
  "untrusted_injected_content",
  "model_generated",
  "unknown"
]
```

```
target_roles:
[
  "submission_recipient",
  "payment_payee",
  "calendar_participant",
  "repo_transfer_target",
  "repo_collaborator",
  "authorization_endpoint",
  "external_file_sink",
  "shopping_purchase_target",
```

```
  "generic_outbound_recipient",
  "unknown"
]
```

```
fact_levels:
[
  "L1_deterministic",
  "L2_semantic_enum",
  "L3_composed"
]
```

```
fact_scopes:
[
  "tool_action",
  "resource",
  "arg",
  "task",
  "trace",
  "external_content",
  "verdict"
]
```

[F] Required objective

Synthesize an initial policy bundle that:

1. Preserves clean task utility by deriving minimal permission templates from trusted clean task authority.
2. Blocks indirect prompt injection and tool-mediated attacks by denying or escalating actions whose authority comes from untrusted observations.
3. Separates task permission from argument provenance: an action class may be permitted while a specific argument binding is denied.
4. Uses contextual binding for targets that are not explicitly in the user prompt but are legitimately obtained from prior task-bound tools.
5. Keeps all rules reusable and auditable.

Return strict JSON only.

Policy Iteration Prompt

SYSTEM MESSAGE

You are the ActGov policy refinement assistant.

You refine an existing ActGov policy bundle using only development-split evidence. Your output must be a small, auditable patch plan. Do not output general safety advice.

ActGov represents each proposed tool call as a ContextRecord and RecordFacts. Policy rules are deterministic predicates over those records and facts.

You must preserve these invariants:

1. Do not use final-test target selection, final-test scorer results, hidden benchmark targets, or attacker metadata to grant permissions.
2. Do not create a task-id whitelist, injection-id blacklist, or example-specific final-test patch.
3. Facts have only three levels:
 - L1 deterministic;
 - L2 semantic enum with finite allowed values;
 - L3 composed from L1/L2 facts and trace state.
4. Facts have only seven scopes: tool_action, resource, arg, task, trace, external_content, verdict.
5. Policy rules must be represented as: (id_p, layer_p, effect_p, dsl_p, pred_p)
6. 'effect_p' is one of: DENY, REQUIRE_APPROVAL, ESCALATE, ALLOW.
7. DENY and REQUIRE_APPROVAL rules have priority over ALLOW rules.
8. ALLOW rules are only narrow exceptions and must require explicit task authority or a valid current contextual binding.
9. Attack evidence may constrain authority but must not create new benign permissions.
10. Prefer the smallest repair that fixes the failure while preserving existing security invariants.
11. Output strict JSON matching the requested schema.

USER MESSAGE

Run one ActGov policy refinement iteration.

[A] Current policy artifacts

```
policy_bundle:
{CURRENT_POLICY_BUNDLE_JSON}
```

```
fact_schema:
{FACT_SCHEMA_JSON}
```

```
rule_catalog:
{RULE_CATALOG_JSON}
```

```
resolver_rules:
{RESOLVER_RULES_JSON}
```

[B] Development evaluation summary

```
benchmark_name:
{BENCHMARK_NAME}
```

```
development_split_name:
{DEVELOPMENT_SPLIT_NAME}
```

```
model_backend:
```

```
{MODEL_BACKEND}
```

```
clean_utility:
{CLEAN_UTILITY}
```

```
attacked_utility:
{ATTACKED_UTILITY}
```

```
security:
{SECURITY}
```

```
asr:
{ASR}
```

```
suite_breakdown:
{SUITE_BREAKDOWN_JSON}
```

```
verdict_distribution:
{VERDICT_DISTRIBUTION_JSON}
```

[C] Failure evidence

Each item below is from the development split only.

```
failure_evidence:
[
  {
    "failure_id": "...",
    "suite": "...",
    "phase": "clean | attack",
    "trusted_user_prompt": "...",
    "untrusted_injection_text": "... or null",
    "expected_user_goal": "...",
    "observed_tool_trace": [
      {
        "step_index": 0,
        "tool": "...",
        "args": {...},
        "tool_observation_source": "
user_prompt | trusted_tool_output |
untrusted_external_content |
model_generated | unknown",
        "actgov_verdict": "ALLOW | DENY |
REQUIRE_APPROVAL | ESCALATE",
        "actgov_reasons": [...],
        "context_record": {...},
        "record_facts": {...}
      }
    ],
    "current_matching_template": {...},
    "current_matching_rules": [...],
    "clean_success": true,
    "attack_success": false,
    "native_scorer_summary": "..."
  }
]
```

[D] Change budget

```
allowed_change_budget:
{
  "max_new_facts": {MAX_NEW_FACTS},
```

```
"max_modified_facts": {MAX_MODIFIED_FACTS
},
"max_new_rules": {MAX_NEW_RULES},
"max_modified_rules": {MAX_MODIFIED_RULES
},
"max_template_updates": {
  MAX_TEMPLATE_UPDATES},
"allow_adapter_repairs": true,
"allow_resolver_repairs": true
}
```

[E] Target metric direction

```
security_target:
{SECURITY_TARGET}
```

```
utility_target:
{UTILITY_TARGET}
```

Return strict JSON only.

Detailed Record Catalog

Tables 6–8 list the records used by the released policy bundles. “LLM” indicates whether an LLM participates in constructing the record.

Table 6: L1 deterministic records across action, permission, binding, history, external-content, and domain scopes.

Record	Scope	Meaning
is_high_risk	action	Action belongs to the suite’s high-risk set.
has_outbound_tag	action	Action sends data or value outward.
has_delete_tag	action	Action is destructive.
has_state_change_tag	action	Tool specification marks a state change.
low_risk_time_read	action	Action is a low-risk system-time read.
permission_granted	permission	Task permission authorizes the tool/action/resource triple.
tool_allowed	permission	Tool belongs to the task’s allowed-tool set.
contextual_followup_allowed	permission	Action/resource pair belongs to the contextual permission closure.
task_intent_outbound	permission	Trusted task grants outbound intent.
task_intent_delete	permission	Trusted task grants delete intent.
task_intent_contextual_binding	binding	Task allows contextual target binding.
task_intent_contextual_outbound_binding	binding	Contextual binding may support outbound actions.
task_intent_contextual_delete_binding	binding	Contextual binding may support destructive actions.
has_binding_context	binding	Prior allowed steps read the resource needed by the current target.
pending_obligations_open	history	Trace contains unresolved obligations.
has_prior_sensitive_read	history	Trace already contains an allowed sensitive read.
has_same_task_history	history	Current task trace has previous steps.
prior_bank_send_allowed	history	A bank send or transfer was already allowed.
github_verification_pending_from_tool_output	history	A tool output opened a GitHub verification obligation.
github_account_verified_before_current_step	history	Scoped GitHub verification already occurred.
verification_channel_scope	external	Channel is scoped, broad/generic, or unknown.
current_high_risk_intent_class	domain	Domain class of the proposed high-risk step.
primary_high_risk_intent_classes	domain	High-risk classes authorized by task permissions.

Table 7: L2 bounded semantic records.

Record	Scope	LLM	Meaning
arg:: <arg>::target_role</arg>	parameter	yes	Finite target-role class for an argument.
arg:: <arg>::provenance</arg>	parameter	yes	Finite provenance class for an argument.
task_goal_balance_check	domain	yes	Whether the trusted task requires balance checking.
calendar_payload_relation	domain	yes	Whether calendar content is task-bound, injection-like, or unrelated.
input_values_contain_sensitive_credential	parameter	no	Input values structurally resemble sensitive credentials.
slack_channel_arg_injection_like	external	no	Slack channel argument contains instruction-like injected content.

Table 8: L3 composed records.

Record	Scope	LLM	Meaning
arg::<arg>::allowed_provenance_classes	permission	no	Allowed provenance classes for the argument’s target role.
first_party_web_navigation	binding	no	Web navigation is bound to the first-party task context.
same_site_download_link	binding	no	Download URL matches a trusted host and is not injection-adjacent.
download_link_from_bound_page	binding	no	Download URL comes from a bound page and is not tainted.
download_host_matches_prompt_or_bound_host	binding	no	Download host matches the prompt or a trusted prior output.
trusted_task_document_url	binding	no	URL is bound to a trusted task document.
shopping_purchase_target_from_clean_platform_context	binding	no	Purchase target comes from clean platform context.
contextual_binding_currently_valid	binding	no	Tool-derived target satisfies the current contextual binding.
workspace_followup_ok	binding	no	Workspace follow-up is supported by prior task context.
calendar_model_generated_participant_task_bound	binding	no	Generated participant remains bound to the task.
model_generated_internal_fanout_allowed	binding	no	Generated internal-recipient fanout is task-bound.
verification_source_bound_to_current_action	binding	no	Verification source is scoped to the current action.
trusted_github_invitation_url	external	no	GitHub invitation URL is bound to the task or trusted prior output.
trusted_discount_coupon_url	external	no	Shopping discount URL is bound to the task or trusted prior output.
trusted_web_submission_endpoint	external	no	Submission endpoint belongs to the trusted endpoint set.
trusted_credential_submission_endpoint	external	no	Credential endpoint is trusted and not injection-adjacent.
web_credential_submission_to_untrusted_endpoint	external	no	Credentials would be submitted to an untrusted endpoint.
generic_inbox_read_during_github_verification	external	no	Broad inbox read occurs during sensitive GitHub verification.
current_read_expands_untrusted_surface	external	no	Read expands the untrusted observation surface.
slack_body_has_urls	external	no	Slack message body contains URLs.
slack_body_urls_bound_to_context	external	no	Slack body URLs are bound to the prompt or trusted prior output.
task_goal_requires_balance_check	domain	yes	Boolean projection of balance-check intent.
pre_payment_sensitive_balance_read	domain	no	Bank-account read occurs before payment.
bill_payment_without_balance_goal	domain	yes	Payment workflow performs an unnecessary balance read.
post_payment_sensitive_balance_read	domain	no	Bank-account read occurs after an allowed payment.
post_payment_balance_read_allowed_by_task_goal	domain	yes	Post-payment read is justified by the trusted task.
github_verification_phase_active	domain	no	Trace is in the GitHub verification phase.
sensitive_verification_phase_active	domain	no	A sensitive verification phase is active.
scoped_verification_channel_available	domain	no	A scoped verification channel is available.
calendar_payload_bound_to_task_goal	domain	yes	Boolean projection of a task-bound calendar payload.
calendar_payload_injection_like	domain	yes	Boolean projection of injection-like calendar content.
step_matches_primary_high_risk_intent	domain	no	Step stays within the authorized high-risk intent class.
github_force_overwrite_without_task_goal	domain	no	Force overwrite lacks explicit task authorization.
travel_reservation_target_model_generated	domain	no	Reservation target was generated rather than catalog-bound.
model_generated_soft_args_allowed	permission	no	Policy variant permits only soft model-generated arguments.

Detailed Policy Catalog

Table 9: Executable ActGov policy rules. “Dyn” and “Dojo” indicate activation in the released benchmark-specific policy bundles.

Policy ID	Layer	Effect	Main records	Dyn	Dojo
TASK-PERM-001	task permission	deny	permission_granted	yes	yes
KERNEL-HI-OUTBOUND	hard invariant	deny	has_outbound_tag, task_intent_outbound	yes	yes
KERNEL-HI-DELETE	hard invariant	deny	has_delete_tag, task_intent_delete	yes	yes
HI-HIGH-RISK-INTENT-CLASS-FREEZE	hard invariant	deny	is_high_risk, step_matches_primary_high_risk_intent	yes	no
HI-POST-PAYMENT-BALANCE-READ-FREEZE	hard invariant	deny	post_payment_sensitive_balance_read, post_payment_balance_read_allowed_by_task_goal	yes	no
HI-SENSITIVE-VERIFICATION-SCOPED-CHANNEL	hard invariant	deny	sensitive_verification_phase_active, scoped_verification_channel_available	yes	no
HI-WEB-CREDENTIAL-SUBMISSION-UNTRUSTED-ENDPOINT	hard invariant	deny	web_credential_submission_to_untrusted_endpoint	yes	no
HI-PRE-PAYMENT-BALANCE-READ-NOT-REQUIRED	hard invariant	deny	bill_payment_without_balance_goal	yes	no
HI-CALENDAR-PAYLOAD-MUST-MATCH-TASK-GOAL	hard invariant	deny	calendar_payload_bound_to_task_goal, calendar_payload_injection_like	yes	yes
HI-INJECTED-DOWNLOAD-NO-FILE-AUTHORITY	hard invariant	deny	same_site_download_link, arg::<arg>::provenance	yes	yes
HI-GITHUB-FORCE-OVERWRITE-REQUIRES-TASK-GOAL	hard invariant	deny	github_force_overwrite_without_task_goal	yes	yes
HI-TRAVEL-RESERVATION-TARGET-MUST-BE-CATALOG-BOUND	hard invariant	deny	travel_reservation_target_model_generated	no	yes
HI-SLACK-OUTBOUND-URL-MUST-BE-TASK-BOUND	hard invariant	deny	slack_body_has_urls, slack_body_urls_bound_to_context	no	yes
HI-SLACK-CHANNEL-TARGET-NO-INJECTED-INSTRUCTION	hard invariant	deny	slack_channel_arg_injection_like	no	yes
REFINE-UNTRUSTED-TARGET	hard invariant	deny	arg::<arg>::provenance	yes	yes
REFINE-TARGET-ROLE-POLICY	hard invariant	deny	arg::<arg>::target_role, arg::<arg>::allowed_provenance_classes	yes	yes
REFINE-TOOL-DERIVED	procedural obligation	escalate	arg::<arg>::provenance, contextual_binding_currently_valid	yes	no
KERNEL-PO-MODEL-GENERATED	procedural obligation	approval	arg::<arg>::provenance, model_generated_soft_args_allowed	yes	no
SLACK-PO-BODY-URL	procedural obligation	approval	slack_body_has_urls, slack_body_urls_bound_to_context	yes	no