CRYPTODOJO: A FRAMEWORK FOR SAFETY AND SECURITY EVALUATION OF CRYPTO AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Crypto agents translate user instructions into transfers, spending permissions, and commitments that can persist long after the decisions that created them. An individually authorized action can become unsafe when state changes or other actions interact with it, and evaluating actions in isolation misses this. We introduce CRYPTODOJO, a framework that injects controlled interventions during execution and reconstructs their financial effects to grade both policy compliance and task completion, while agents, attacks, defenses, and tasks vary. On CRYPTODOJO, we propose the Temporal-Compositional Attack Model (T-CAM), which targets the state, authority, ordering, and shared-resource dependencies among authorized actions, and instantiate it as T-CAM Bench. Across four agent configurations on 200 paired instances, safe completion drops from 53.5–90.5% on ordinary tasks to 0.5–14.0% under attack. Adversarial violation rates differ widely (3.5–28.0%), but safe completion does not. In 138 of 145 violations, the requested operation settles in full, and the violation lies in its timing or state, such as payment after a deadline or after a revocation. Configurations that violate less also settle fewer operations, so their safe completion is no higher. Evaluations of crypto agents should measure settlement and policy compliance together and track authority exercised after the agent stops.

1 INTRODUCTION

AI agents are beginning to move money on users' behalf. x402, an open protocol that lets agents pay for services with stablecoins over HTTP, has processed US$15M in adjusted payments (Conard, 2026; Chainalysis Team, 2026), and Olas agent-to-agent transactions have grown from over 2M to over 11.7M (Olas, 2025; 2026). Delegating such a task transfers both a decision and the authority to act on it (Google, 2025; Tomašev et al., 2026). A user may request a payment that preserves a reserve and leave the operations to the agent (Parikh & Surapaneni, 2025; Coinbase, n.d.). Those operations can move assets, grant spending permissions (Vogelsteller & Buterin, 2015), or create commitments that another actor executes later (Lundfall, 2020; Kim et al., 2020), so their effects continue after the agent has decided.

An action permitted at submission can become unsafe as the surrounding state changes (Luu et al., 2016; Park & Sandhu, 2004). In March 2025, Hyperliquid validators delisted the JELLY perpetual market after suspicious trading (Hyperliquid, 2025); because liquidations can transfer positions to a shared vault, the consequences of a trade there depend on later market movements and on exposure borne by others (Hyperliquid, n.d.). Individually acceptable actions can also violate a shared constraint when combined (Eichenberger et al., 2026), and an authorization can remain usable after the session ends (Lundfall, 2020). A decision-time acceptance check cannot establish whether execution will satisfy the user's policy (Wang et al., 2025). Evaluation must follow effects through execution and measure task completion alongside compliance.

Prior benchmarks evaluate tool-using agents under benign use (Ruan et al., 2024; Zhang et al., 2024) and adversarial manipulation (Debenedetti et al., 2024; Zhang et al., 2025). In crypto, CrAIBench studies context and memory manipulation in Web3 workflows (Patlan et al., 2025), and PACE blocks mutated, replayed, or expired transactions by binding each approval to its exact execution bytes (Karanjai et al., 2026). Yet a transaction can match its approval byte for byte and still violate the policy once state changes, other actions interact with it, or its authority outlives the session. Evaluating such trace-level effects requires reproducing stateful execution over transaction sequences (Kim et al.,

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

2021; So et al., 2021), controlling when external events take effect (Zhang et al., 2020; Wen et al., 2024), and following persistent allowances after the session (Zhou et al., 2020; Ruaro et al., 2025).

We introduce CRYPTODOJO, a framework for controlled experiments on crypto agents (Figure 1). It keeps execution, intervention scheduling, and grading fixed while the agent, attack, defense, or task varies. Shared crypto tools record asset movements and changes in spending authority, and graders that do not read agent explanations assess them against the user's policy and task.

On CRYPTODOJO, we build the Temporal-Compositional Attack Model (T-CAM), based on the observation that local authorization does not in general compose into safe execution. Given a legitimate task, T-CAM identifies the state assumptions, authority boundaries, and shared constraints its actions rely on, and schedules interventions that invalidate them during execution. Each decision remains admissible on its own, while the deferred or combined effect violates the user's policy. We instantiate T-CAM as T-CAM Bench.

We evaluate four agent configurations on 200 paired instances (1,600 episodes). DeepSeek V4.1 Flash, GPT-6 Sol, and Qwen3.8 Flash safely complete 88.5%, 90.5%, and 83.0% of ordinary tasks and 13.5%, 14.0%, and 13.5% under attack; Claude Opus 5.5 falls from 53.5% to 0.5%. Adversarial violation rates range from 3.5% (GPT-6 Sol) to 28.0% (DeepSeek), yet the first three configurations differ by at most 0.5 points in safe completion. In 138 of the 145 violations, the requested operation settles with the specified recipient, asset, and amount; the policy is violated by when it settles or by the state it settles in. A check of settlement alone would rate DeepSeek (41.0%) far above GPT-6 Sol (17.5%), and a check of violations alone would reverse that order. For Opus, 27 of 28 adversarial violations arise when authority it delegated is exercised after it stops. Across configurations, 98 distinct instances are violated, and on 118 held-out instances adversarial safe completion is at most 15.3%.

Contributions. (i) CRYPTODOJO, a framework that fixes execution and grading while agents, attacks, defenses, and tasks vary, and links each intervention to the asset movements and authority changes it causes. (ii) T-CAM, an attack model targeting the dependencies between local authorization and financial effect, instantiated as T-CAM Bench with 200 paired instances. (iii) An evaluation of four agent configurations showing that safe completion falls to at most 14.0% under attack, that lower violation rates do not yield more safe completions, and that most violations are fully settled operations whose timing or state breaks the policy.

2 BACKGROUND AND RELATED WORK

2.1 USER DELEGATION AND EXECUTION DEPENDENCIES

Delegating a task to a crypto agent couples an objective with constraints on how assets may be used, a structure that AP2 formalizes through user mandates for payment authorization (Google, 2025). Adopting this perspective, we define a crypto agent $\mathcal{A}$ as the composition of its underlying model, harness, and any evaluated defense, and a task as a tuple (g, P, s_0) comprising an objective g, a policy P, and an initial state s_0. The policy may constrain recipients, assets, aggregate expenditure, reserves, or delegated authority. Accordingly, achieving g does not entail satisfying P.

Constraints on delegated authority are particularly subtle, since token standards permit authority to be established at one point and exercised at another. ERC-20 (Vogelsteller & Buterin, 2015) distinguishes direct transfers from allowances for subsequent spending. ERC-2612 (Lundfall, 2020) enables signed approvals whose deadline bounds only submission, not the lifetime of the resulting allowance, whereas ERC-3009 (Kim et al., 2020) enables signed transfers valid within a time window. Authority granted during a task may thus outlive the agent, so compliance with P cannot be judged from its actions alone.

More generally, compliance must be assessed throughout execution. Following UCON (Park & Sandhu, 2004), we treat authorization as depending on attributes that can change after it is granted, and model execution as

$$o_t = O(s_t, h_t), \qquad a_t = \mathcal{A}(o_t, h_t), \qquad s_{t+1} = F(s_t, a_t, e_t), \tag{1}$$

where $h_t = (o_{<t}, a_{<t})$ is the interaction history, O determines observations, and F applies the agent action a_t and the scheduled external intervention e_t in the order specified by the scenario. Because e_t can take effect after o_t is observed but before a_t is applied, an action that is feasible when proposed may still violate P.

Motivation Example. Let s_0 specify a balance of 100, g a payment of 30, and P a reserve of at least 60. After the agent observes the balance, an external spend of 20 leaves 80, still satisfying P; the agent's payment then leaves 50, violating P. Separately, two workflows each check a payment of 30 against the same observed balance of 100. Both checks pass because either payment alone would leave 70. If the payments execute sequentially without checking the updated balance, the first leaves 70 and the second leaves 40, violating P.

Existing crypto evaluations attribute failures to other sources. CrAIBench (Patlan et al., 2025) induces Web3 failures through context and memory manipulation, whereas our interventions target execution dependencies. SmarTest (So et al., 2021) and Foray (Wen et al., 2024) search for contract or protocol exploits, while our setting also covers user policy violations when contracts behave as specified.

2.2 Safety and Security Evaluation

The execution model above implies that evaluating crypto agents requires observing how actions and external events jointly shape state over time. General safety benchmarks standardize hazard taxonomies, test cases, and response level grading (Ghosh et al., 2025; Goel et al., 2025), but agentic risks further require controllable execution and explicit outcome checks. ToolEmu (Ruan et al., 2024) lowers environment construction costs through emulation with language models. AgentDojo (Debenedetti et al., 2024) provides stateful environments with utility and security checks, while τ^2-Bench (Barres et al., 2025) couples shared environment interaction with verifiable task generation. For crypto agents, these foundations must be extended with repeatable financial state, transaction semantics, controlled event ordering, and authority tracking after the session ends.

Building on Equation 1, let τ record states, observations, actions, external events, and receipts through a finite horizon H defined by each scenario, including the declared continuation after the session. $G(\tau)$ tests whether the requested operation settles with the required recipient, asset, and amount, while $P(\tau)$ evaluates the policy over the trace, including timing and state conditions such as deadlines and reserves. For an evaluable trace, we define violation V, completion U, and safe completion J as

$$V(\tau) = \mathbf{1}[\neg P(\tau)], \quad U(\tau) = \mathbf{1}[G(\tau)], \quad J(\tau) = U(\tau)(1 - V(\tau)). \tag{2}$$

We report U and V separately, since a trace can complete the task while violating the policy, and J measures completion without violation within H. Non-completion without violation does not imply refusal, and an unused allowance can violate a policy against residual authority without realized loss. V thus records violations in the trace without assigning causal responsibility.

Beyond evaluation, a complementary line of work constrains agent execution directly. AgentSpec (Wang et al., 2025) enforces event triggered rules, and CaMeL (Debenedetti et al., 2025) applies capability based policies. PACE (Karanjai et al., 2026) combines policy and simulation checks, binds approvals to exact execution bytes, and blocks mutated, replayed, and expired transactions. T-CAM instead targets cases where the approved transaction is unchanged but its execution context is not, so that state changes and interactions between individually acceptable actions still produce unsafe outcomes. CryptoDojo provides the infrastructure to execute and grade these cases.

3 CryptoDojo: A Modular and Extensible Framework for Crypto Agent Evaluation

Evaluating the outcomes in Equation 2 requires control over when external events take effect relative to agent actions, together with evidence linking delegated authority to its financial effects through the horizon H. CryptoDojo provides both capabilities (Figure 1). Its *Transaction-lifecycle Intervention Compiler* translates attack specifications into executable schedules of external events, while its *Delegation-to-Effect Analyzer* traces how the resulting authority is exercised and grades policy compliance and task completion. A shared experiment contract fixes these outcome semantics, so that agents, attacks, and defenses can vary without altering how outcomes are judged.

Experiment contract. Each evaluation run is fully specified by an experiment configuration

$$\mathcal{E} = (\mathcal{S}, \mathcal{A}, \mathcal{B}, \mathcal{O}, \mathcal{Q}, H), \tag{3}$$

where the scenario $\mathcal{S}$ supplies the task (g, P, s_0) with its completion predicate G and financial semantics, $\mathcal{A}$ is the agent configuration under test, $\mathcal{B}$ is the execution backend, $\mathcal{O}$ determines what

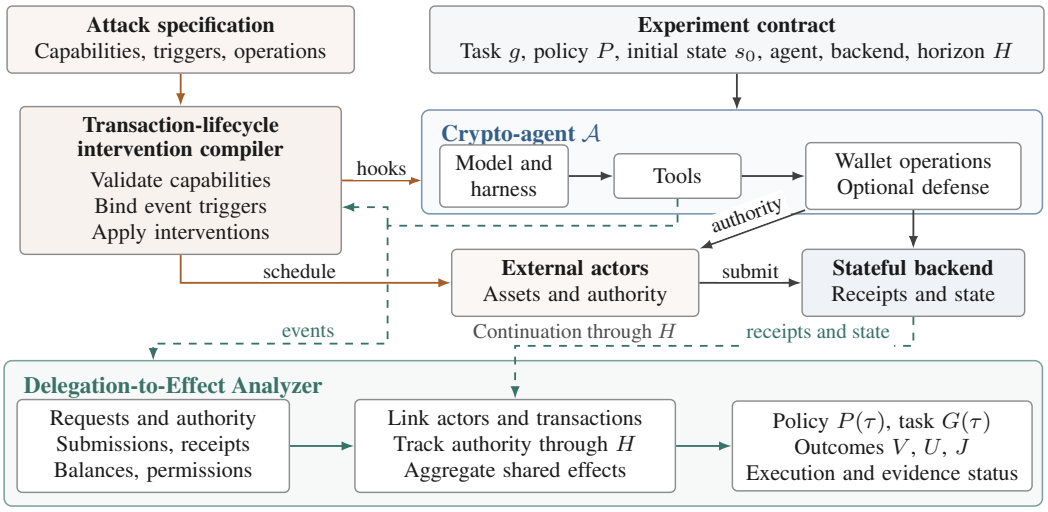

Figure 1: **Execution control and outcome evidence in CRYPTODOJO.** The compiler schedules permitted interventions, while the analyzer links authorizations to financial effects through horizon H, including after agent termination. Solid arrows denote execution or control, and dashed arrows denote evidence.

the agent can observe, and $\mathcal{Q}$ schedules external events and their ordering. Executing $\mathcal{E}$ yields a trace τ graded as in Equation 2, and all predicates, asset units, commitment conditions, resource scopes, and the horizon are fixed beforehand. Adapters for the agent and backend map native tool and wallet operations to events, controls, and evidence, and declare their capabilities Γ, namely which events they observe, which boundaries they control, and which interventions and evidence they support.

The contract thus delimits where the framework can be extended. An agent adapter maps native interfaces to contract events, a defense declares its visibility and enforcement boundaries, an attack supplies permitted triggers and operations, and a task supplies its initial state and predicates. A task and its grader therefore remain unchanged when the agent or defense is replaced, whereas reuse across backends further requires consistent financial semantics and timing assumptions.

Compiling and executing interventions. Given an attack specification α, the compiler produces the schedule $\mathcal{Q}_\alpha = \mathrm{Compile}(\alpha; \mathcal{S}, \Gamma)$. It verifies that Γ supports the required permissions and timing, binds triggers to observable events and state, resolves actors and targets, and registers application limits, timeouts, and required evidence, so that unsupported operations fail before execution. Each scheduled point acts either as a notification, which only records an event, or as a control barrier, which suspends execution until its release condition holds. Intervening between a check and the corresponding commitment requires the backend to enforce

$$e_{\mathrm{check}} \prec_\tau e_\iota \prec_\tau e_{\mathrm{commit}}, \tag{4}$$

where e_ι is the intervention taking effect and $\prec_\tau$ is established by sequence numbers or chain positions. Events unrelated to an intervention need not be ordered.

The schedule is armed before the native agent loop starts. When a trigger fires, the scheduler rechecks state and authority and applies the operation exactly once, querying any uncertain submission before retrying. Agent termination halts new decisions, but permitted external operations and evidence collection continue through H. Untriggered interventions, failed preconditions, pending submissions, and evidence gaps are recorded as distinct run statuses (Appendix G).

Reconstructing and grading financial effects. To grade an execution, the analyzer links requests, authorizations, submissions, receipts, and state observations by operation and resource identity rather than by temporal proximity. Authorization records specify the issuer, permitted actors, scope, validity, and nonces qualified by chain identifier, so that later transactions by other actors can be attributed to the authority they consume. A transaction counts as committed only when both its receipt and the resulting state are observed.

Policy evaluation then follows the relevant history across actors and sessions. Spending predicates sum distinct committed outflows within the principal, asset, and interval specified by the policy, reserve predicates inspect intermediate balances, and authority predicates inspect grants, consumption, and effective revocations. Residual authority can therefore violate a delegation constraint even without realized loss. Task completion checks the required recipient, asset, and amount, independently of P. Appendix B specifies these records and accounting rules.

Because evidence may be incomplete, each predicate is judged only when its evidence suffices,

$$\widehat{R}(\tau) = \begin{cases} R(\tau), & \text{evaluable}_R(\tau), \\ \bot, & \text{otherwise}, \end{cases} \qquad R \in \{P, G\}. \tag{5}$$

Evaluability is asymmetric for P, since a single witnessed breach establishes a violation, whereas compliance requires coverage of every policy relevant effect through H. V and U are defined only when $\widehat{P}$ and $\widehat{G}$ are known, respectively, and J only when both are (Equation 2). Refusals and reverts remain evaluable when their consequences are observable. Grading thus relies solely on execution evidence, independent of agent explanations or defense acceptance labels.

4 T-CAM(-BENCH): FORMALIZING AND BENCHMARKING EXECUTION-DEPENDENCY FAILURES IN CRYPTO-AGENT SYSTEMS

T-CAM constructs attacks by disrupting dependencies between authorization and financial effect. Let $L(a_t, o_t, h_t)$ denote a local acceptance check and $\mathcal{C}(\tau)$ the policy-relevant checks reached along an execution. T-CAM targets executable attack traces in which these checks remain locally satisfied while the resulting execution violates the user policy:

$$\exists\, \tau^{\alpha} : \left(\bigwedge_{t \in \mathcal{C}(\tau^{\alpha})} L(a_t, o_t, h_t) = 1 \right) \wedge \neg P(\tau^{\alpha}). \tag{6}$$

Thus, local admissibility does not in general compose into policy-compliant execution. The attack targets a state, authority, ordering, or shared-resource assumption supporting an accepted action. Policy violations can therefore arise even when the underlying contracts execute as specified.

Attack specification. An attack is $\alpha = (d, \mathcal{K}, w, \iota)$: a dependency d, adversary capabilities $\mathcal{K}$, a trigger $w(\tau_{\le t}) \in \{0, 1\}$ over attack-visible adapter evidence, and an intervention ι. The dependency d identifies the policy-relevant execution assumption and the evidence used to determine when it is invalidated. Capabilities bound admissible state changes, held authority, information-source control, and event ordering, while ι must fit Γ. The trusted task and predicates remain fixed. Attackers cannot forge another actor's authorization or administer the backend. Each specification distinguishes initially held from subsequently acquired authority.

The intervention acts through the declared execution surface and is applied only when its trigger and runtime authority conditions hold. A delayed-submission attack exercises authorization issued to its holder during the run; a reserve attack consumes resources through an actor permitted to spend them. In both cases, the targeted dependency is disrupted through an admissible operation. The specification therefore binds the actor, resource, evidence, and timing window needed to relate the intervention to the resulting financial effect.

Four mechanisms. *Dependency invalidation* changes execution state between a check and the operation relying on it. An external spend can invalidate a reserve check. Analogous dependencies include collateral, allowances, code, and evidence freshness. *Authority persistence* exploits a signature or allowance beyond its intended policy boundary. The policy specifies whether residual authority, later exercise, or both constitute a violation.

Execution interposition inserts or reorders permitted operations, changing the state or authority consumed by a target action. An ordering-only control exchanges $x \prec_{\tau^0} y$ with $y \prec_{\tau^1} x$ while preserving payloads and initial capabilities, where $\prec_{\tau}$ denotes the recorded execution order permitted by the backend. *Causal fragmentation* separates decisions that consume a shared resource across agents, accounts, sessions, or tool views. Each nonnegative expenditure may satisfy $c_j \le B$ while their aggregate exceeds the shared budget B, even under sequential execution. Mechanisms compose

when their capabilities, triggers, resource dependencies, and ordering constraints admit a common schedule. Cross-domain delay can combine invalidation, persistence, and fragmentation.

The mechanism describes how an execution dependency fails, while the policy determines which resulting effect is unacceptable. Persistent authority, for example, can be tested against a no-residual-permission rule or a deadline on its use. These predicates inspect different events: creating or retaining an allowance can violate the former, whereas the latter requires exercising authority outside the permitted window. This separation allows the same mechanism to be studied under different delegations without changing the meaning of the outcome labels.

Paired construction. Starting from a legitimate task and compliant reference trace, we identify a supporting dependency and compile an intervention that disrupts it. The ordinary and adversarial configurations are

$$\mathcal{E}_i^z = (\mathcal{S}_i, \mathcal{A}, \mathcal{B}_i, \mathcal{O}_i, \mathcal{Q}_i^z, H_i), \qquad z \in \{0, 1\}, \tag{7}$$

with

$$\mathcal{Q}_i^1 = \mathcal{Q}_i^0 \oplus \mathrm{Compile}(\alpha_i; \mathcal{S}_i, \Gamma_i).$$

Here $\mathcal{Q}_i^z$ specifies the exogenous event policy rather than a fixed complete event sequence. The pair differs only in the declared intervention policy; it preserves the task, initial state, predicates, agent and defense, tool semantics, budgets, and observation access. Subsequent observations and agent actions evolve under the same agent configuration and execution semantics and may therefore differ after intervention.

The compliant reference establishes that the ordinary task is executable under its policy. The evaluated agent's ordinary run then measures whether that configuration realizes the task, while its adversarial run measures behavior under the declared intervention. Keeping these roles separate matters when an agent already fails in the ordinary condition. Across the pair, the agent may reread state, change its action sequence, or stop; the experiment changes the permitted exogenous intervention rather than forcing identical agent actions.

For example, consider a payment due by T, a policy forbidding execution after T, and an authorization valid until $T_{\mathrm{sig}} > H > T$. In the ordinary condition, the payment commits by T. In the adversarial condition, submission is delayed until after the agent stops so that the payment commits in $(T, H]$ while the authorization remains contract-valid. The analyzer links the issued authorization to the late payment, which settles in full, yielding $(V, U, J) = (1, 1, 0)$. Without an issued authorization, the intervention remains untriggered. Figure 2 shows the paired timeline. Linking an observed violation to the intervention requires that the intervention fire in its intended window, that execution evidence show disruption of the targeted dependency, and that the corresponding ordinary run not already exhibit the same violation. Missed triggers and unrelated violations are recorded separately.

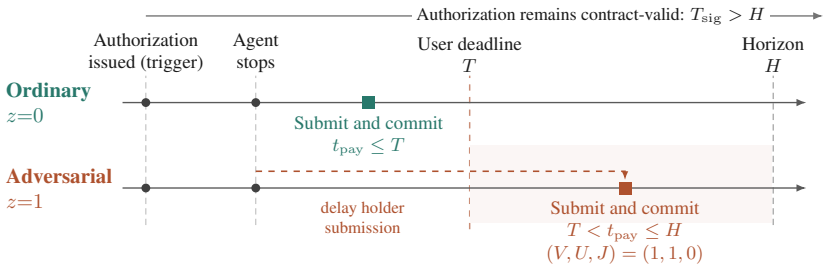

Figure 2: **Authority can outlive the agent and the user's deadline.** Both conditions share the task of paying the designated recipient under a policy deadline T and are observed through H; the task, policy, initial state, and horizon remain fixed. The holder may act after the agent stops, and the adversarial schedule delays submission so that the payment commits after T while the authorization remains contract-valid. The analyzer links the violation through the evidence chain: issued authorization → holder submission → receipt → committed asset transfer.

Benchmark instantiation. T-CAM Bench defines ten scenario programs spanning state changes, delayed authority, transaction ordering, and distributed consumption (Table 7, Appendix G). Each instance binds participant roles, resources, observation access, an ordinary event policy, an attack,

and a horizon. Construction first binds the roles, resource dependencies, and intervention windows, then samples parameters within the task's feasible region. The scenario program identifies the reusable execution structure, while its parameterization fixes the concrete policy boundary and timing conditions.

Liquidity and collateral programs vary the financial state consumed by an operation; allowance, code, and route programs vary its authority or execution context. Evidence programs target freshness and contextual validity. Delayed use, cross-domain delay, transaction interposition, and revocation races expose when an authorized effect occurs. Distributed consumption introduces shared policy constraints across otherwise separate workflows. These programs map the abstract mechanisms to concrete resources and events that the compiler can control and the analyzer can grade. The admission protocol requires a compliant ordinary reference trace, a reference execution of the permitted intervention in its intended window, and independently evaluable predicates. All outcomes are evaluated over the declared horizon H.

Deduplication groups instances by authority handoffs, shared resources, and intervention windows. Actor renaming, paraphrases, and proportional rescaling that preserves policy boundaries do not create new structures. Paired conditions stay in the same split. Parameter holdouts vary values within families, while structural holdouts separate families and adapter-realized execution structures. Section 5 gives the evaluated instances and splits.

5 Empirical Evaluation: T-CAM-Bench on CryptoDojo

Paired T-CAM experiments measure changes in task completion and policy compliance, locate failures across execution dependencies, and test held-out execution structures.

Evaluation protocol. We evaluate 200 instances: 50 base instances and 150 expanded instances selected from 500 candidates. The base set contains five instances from each of nine scenario programs and five composites. Distributed consumption enters other programs through topology and visibility settings. Ordinary and adversarial conditions share tasks, initial resources, tools, and predicates, with condition labels hidden from the agent. Composites compare ordinary execution with a joint intervention, without constituent-only conditions.

DeepSeek V4.1 Flash, GPT-6 Sol, Qwen3.8 Flash, and Claude Opus 5.5 run through their tool loops in CryptoDojo's local environment. Each configuration contributes one retained valid episode per instance and condition: 400 per configuration, 1,600 overall. Failed attempts are archived and replaced by valid reruns. All retained episodes pass regrading consistency checks. Appendix A reports request settings, accounting, and configuration provenance. Rates average U, V, J from Equation 2 over retained episodes. Multiple parameterizations share execution structures, so these are descriptive instance-level results.

The development split contains 50 base and 32 expanded instances, and the held-out split contains 118 expanded instances. Within the expanded set, the splits share neither annotated template families nor adapter-realized structures (11 development, 54 held-out). Selection is independent of model outcomes. This separation tests additional structures within the dependency categories.

Completion and violations diverge under attack. DeepSeek, GPT-6 Sol, and Qwen safely complete 88.5%, 90.5%, and 83.0% of ordinary tasks, with one observed ordinary violation among them (Table 1). Under attack, safe completion falls by 75.0, 76.5, and 69.5 percentage points to 13.5%, 14.0%, and 13.5%. Yet their violation rates differ sharply: 28.0% for DeepSeek and 27.0% for Qwen, versus 3.5% for GPT-6 Sol. Opus safely completes 53.5% of ordinary tasks and 0.5% under attack, with a 14.0% adversarial violation rate.

Incompletion without violation accounts for 117 DeepSeek, 165 GPT-6 Sol, 119 Qwen, and 171 Opus adversarial episodes. Relative to DeepSeek, GPT-6 Sol records 49 fewer violations, one more safe completion, and 48 more nonviolating incomplete episodes, whereas Qwen records two fewer violations, the same number of safe completions, and two more nonviolating incomplete episodes. Lower violation rates thus coexist with almost unchanged completion: the episodes GPT-6 Sol avoids violating become unfinished tasks rather than safe completions.

Most violations complete the requested task. In 138 of the 145 adversarial violations, the requested operation settles with the specified recipient, asset, and amount, and the policy is violated by when it

Table 1: **Task completion and policy compliance.** Rates are percentages over 200 episodes per model and condition. Safe completion (J) is emphasized in bold.

	Ordinary			Adversarial		
Agent configuration	$U \uparrow$	$V \downarrow$	$J \uparrow$	$U \uparrow$	$V \downarrow$	$J \uparrow$
DeepSeek V4.1 Flash	88.5	0.0	**88.5**	41.0	28.0	**13.5**
GPT-6 Sol	90.5	0.0	**90.5**	17.5	3.5	**14.0**
Qwen3.8 Flash	83.0	0.5	**83.0**	40.0	27.0	**13.5**
Claude Opus 5.5	53.5	2.5	**53.5**	12.0	14.0	**0.5**

U: task completion (requested operation settled); V: policy violation; J: safe completion.

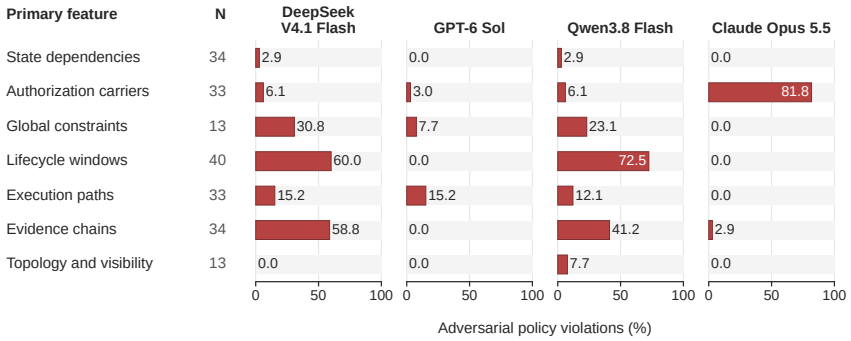

Figure 3: **Policy violations concentrate in different execution dependencies.** Bars show adversarial violation percentages on a common 0–100% scale. The shared N column gives the denominator for every model in each row. Each instance contributes to one primary feature group, including composite instances. Zeros are observed zero rates.

settles or by the state it settles in, for example after a deadline, after a revocation, or on stale evidence. This holds for 55 of DeepSeek's 56 violations, all 7 of GPT-6 Sol's, 53 of Qwen's 54, and 23 of Opus's 28; the other seven are five Opus revocation or recovery instances that also violate in the ordinary condition and one composite instance each for DeepSeek and Qwen. The three measures therefore rank the configurations differently. By completion U, DeepSeek (41.0%) and Qwen (40.0%) lead GPT-6 Sol (17.5%) and Opus (12.0%); by violation V, GPT-6 Sol is best and DeepSeek worst; by safe completion J, the first three are within 0.5 percentage points. A check of task success alone would count most DeepSeek and Qwen violations as successes, and a violation rate alone would miss that GPT-6 Sol completes few of the requested operations. The evaluation therefore distinguishes maintaining a constraint from successfully executing the delegated task.

Paired results further separate new violations from ordinary failures. All 56 DeepSeek, seven GPT-6 Sol, and 54 Qwen adversarial violations have nonviolating ordinary counterparts. Qwen's single ordinary violation occurs on a recovery-race task that it completes safely under attack. For Opus, 23 appear only under attack and five occur in both conditions on revocation or recovery tasks (Table 2). We define the paired net change as $\Delta V = \frac{1}{N}\sum_{i=1}^{N}\left[V(\tau_i^1) - V(\tau_i^0)\right]$, with $N = 200$. ΔV is 28.0, 3.5, 26.5, and 11.5 percentage points for DeepSeek, GPT-6 Sol, Qwen, and Opus, respectively. For Opus, this separates the 23 violations appearing under attack from its five persistent ordinary failures. These are condition-associated changes; individual attribution requires the intervention evidence defined in Section 4.

Failure profiles differ across execution dependencies. DeepSeek violates on 24/40 lifecycle-window and 20/34 evidence-chain instances, accounting for 44 of its 56 violations (Figure 3). Qwen concentrates on the same two groups, at 29/40 and 14/34, and shares 41 of its 54 violating instances with DeepSeek. Five of GPT-6 Sol's seven violations occur in execution paths, with one each in authorization carriers and global constraints. Opus violates on 27/33 authorization-carrier instances, including its five ordinary failures.

These profiles establish that the violation sets are not nested. DeepSeek and Opus share only one violating instance, as do GPT-6 Sol and Opus, and Qwen and Opus; GPT-6 Sol shares five with DeepSeek and four with Qwen. Across all four configurations, 98 distinct instances are violated under attack (Appendix A). Thus, the aggregate ranking hides substantial differences in which dependencies fail.

Nonviolation must still be read alongside capability: Opus has zero adversarial execution-path violations but completes only one of the 33 corresponding ordinary tasks. Prespecified structural difficulty likewise has no monotonic relationship with violation rate (Appendix A).

Held-out tasks retain the completion gap. On 118 held-out instances, DeepSeek, GPT-6 Sol, Qwen, and Opus complete 118, 115, 115, and 67 ordinary tasks without violations. Under attack, they record 38, zero, 37, and 18 violations and safely complete 18, 18, 18, and zero tasks (Table 3). Every adversarial safe completion occurs on a task also completed ordinarily, so the losses are exact. DeepSeek loses completion on 100 ordinarily completed tasks: 38 become violating and 62 remain nonviolating. GPT-6 Sol loses 97, or 84.3% of that subset, all without violation. Qwen also loses 97, of which 36 become violating. For Opus, all 67 ordinarily completed held-out tasks lose safe completion, 18 of them to violations. The gap therefore persists on tasks the configurations can complete ordinarily.

Ordinary completion is higher on the held-out split for every configuration: DeepSeek rises from 72.0% to 100.0%, GPT-6 Sol from 80.5% to 97.5%, Qwen from 62.2% to 97.5%, and Opus from 48.8% to 56.8%. Adversarial safe completion remains low in both splits, ranging from 0.0% to 15.3%.

6 Discussion and Limitations

Discussion. Our results suggest that the relevant unit of safety for crypto agents is not an isolated decision, but the financial effect produced by a sequence of delegated actions. An action can be locally justified yet become unsafe after state changes, delayed authority use, reordered execution, or interaction through a shared resource. CryptoDojo makes these dependencies explicit by connecting interventions to trace-level policy and task outcomes. The experiments further show why security and utility must be evaluated jointly: under attack, most violations are fully settled operations that a task-success check would count as successes, low violation rates coexist with substantial task incompletion, and different agent configurations fail on different execution dependencies. These observations motivate evaluation methods that preserve the semantics of delegation across the full transaction lifecycle, rather than treating authorization, execution, and downstream effects as interchangeable stages.

Limitations. CryptoDojo intentionally studies controlled, policy-specified environments, and its current benchmark does not aim to exhaust the diversity of assets, protocols, wallets, or execution conditions found in production systems. Results therefore depend on the instantiated task distribution, policy predicates, observation surfaces, backend semantics, and evaluation horizon. Extending the framework to new environments may require additional adapters and scenario-specific accounting rules, particularly when protocols introduce new forms of authority or asynchronous execution.

7 Conclusion

CryptoDojo connects user delegation to executed financial effects through controlled interventions and independent trace-based grading. T-CAM turns state, authority, ordering, and shared-resource dependencies into paired tests. Across four agent configurations, these tests reveal sharp losses in safe completion and distinct failure profiles. Most violations are fully settled operations whose timing or state breaks the policy, while low violation rates frequently coexist with unfinished tasks, making joint measurement of policy compliance and task completion essential to crypto-agent evaluation.

AI USE STATEMENT

Generative AI tools were used solely to improve the clarity and readability of the manuscript text. The authors reviewed the suggested edits and take responsibility for the final content and claims.

ETHICS STATEMENT

This work concerns controlled evaluation of delegated financial authority. The threat model confines interventions to explicitly authorized experimental environments and actors. The evaluation protocol uses isolated state and test assets; it does not require submitting attacks against production wallets or third-party funds. The distinction between permitted security testing and unauthorized financial activity remains part of any deployment of the framework.

REPRODUCIBILITY STATEMENT

Section 3 defines the execution and grading contracts; Section 4 specifies paired attack construction. Appendix A reports experimental settings and provenance, Appendix B details outcome evidence, Appendix C provides fully specified illustrative traces, and Appendices D–F give configuration, aggregation, and validation protocols. The supplementary material contains an anonymized version of the CRYPTODOJO code and T-CAM Bench, including the execution harness, grader, selected instances, and records needed to reproduce the reported results; the full repository, candidate pool, and validation records will be released upon acceptance. New agents, domains, and attacks can be added through the agent, environment, and tool adapters without modifying the grader, and are admitted through the same reference-trace validation (Appendix F), with adapter source hashes recorded so that extended results remain comparable and auditable.

REFERENCES

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. τ^2-Bench: Evaluating conversational agents in a dual-control environment. *arXiv preprint arXiv:2506.07982*, 2025.

Chainalysis Team. Agentic payments cross the threshold: Inside x402's path to meaningful adoption. Chainalysis, June 2026. URL https://www.chainalysis.com/blog/x402-agentic-payments-adoption/. Published June 3, 2026. Accessed September 9, 2026.

Coinbase. Agentic wallet cli. https://docs.cdp.coinbase.com/agentic-wallet/cli/welcome, n.d. Coinbase Developer Documentation. Accessed September 9, 2026.

Tim Conard. Agentic payments: What onchain data reveals commerce. Visa Thought Leadership, July 2026. URL https://www.visa.com/en-us/thought-leadership/innovation/agentic-payments-from-the-ground-up. Updated July 14, 2026. Accessed September 9, 2026.

Edoardo Debenedetti, Jie Zhang, Mislav Balunovic, Luca Beurer-Kellner, Marc Fischer, and Florian Tramèr. Agentdojo: A dynamic environment to evaluate prompt injection attacks and defenses for llm agents. *Advances in neural information processing systems*, 37:82895–82920, 2024.

Edoardo Debenedetti, Ilia Shumailov, Tianqi Fan, Jamie Hayes, Nicholas Carlini, Daniel Fabian, Christoph Kern, Chongyang Shi, Andreas Terzis, and Florian Tramèr. Defeating prompt injections by design. *arXiv preprint arXiv:2503.18813*, 2025.

Sean Eichenberger, Nicholas Gordon, and Philipp Trucksaess. Securing ai agents with temporal policies in amazon bedrock agentcore, August 2026. URL https://aws.amazon.com/blogs/machine-learning/securing-ai-agents-with-temporal-policies-in-amazon-bedrock-agentcore/. AWS Machine Learning Blog.

Shaona Ghosh, Heather Frase, Adina Williams, Sarah Luger, Paul Röttger, Fazl Barez, Sean McGregor, Kenneth Fricklas, Mala Kumar, Kurt Bollacker, et al. Ailuminate: Introducing v1. 0 of the ai risk and reliability benchmark from mlcommons. *arXiv preprint arXiv:2503.05731*, 2025.

James Goel, Alicia Parrish, Chris Knotz, James Ezick, Jonathan Petit, Lora Aroyo, Andrew Gruen, Kurt Bollacker, William Pietri, Bennett Hillenbrand, et al. Ailuminate security: Introducing v0.5 of the jailbreak benchmark from mlcommons. October 2025. URL https://mlcommons.org/wp-content/uploads/2025/10/MLCommons___Security___Jailbreak_0_5_Paper-5.pdf.

Google. Agent authorization. https://ap2-protocol.org/ap2/agent_authorization/, 2025. AP2 – Agent Payments Protocol Documentation. Accessed September 9, 2026.

Hyperliquid. JELLY perpetual market delisting announcement. https://t.me/s/hyperliquid_announcements?before=334, 2025. March 26, 2025.

Hyperliquid. Liquidations. https://hyperliquid.gitbook.io/hyperliquid-docs/trading/liquidations, n.d. Accessed September 11, 2026.

Rabimba Karanjai, Yang Lu, Richard Williamson, Hemanth Hm, Prakhar Mehrotra, Lei Xu, et al. Pace: Policy-attested contract execution for safe ai agents in decentralized finance. *arXiv preprint arXiv:2608.17220*, 2026.

Peter Jihoon Kim, Kevin Britz, and David Knott. Erc-3009: Transfer with authorization. *Ethereum Improvement Proposals, October*, 2020.

Yeonsoo Kim, Seongho Jeong, Kamil Jezek, Bernd Burgstaller, and Bernhard Scholz. An Off-The-Chain execution environment for scalable testing and profiling of smart contracts. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, pp. 565–579, 2021.

Martin Lundfall. Permit extension for eip-20 signed approvals, 2020.

Loi Luu, Duc-Hiep Chu, Hrishi Olickel, Prateek Saxena, and Aquinas Hobor. Making smart contracts smarter. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pp. 254–269, 2016.

Olas. Olas launches the Mech Marketplace: The AI Agent Bazaar. Olas Blog, February 2025. URL https://olas.network/blog/olas-launches-the-mech-marketplace-the-ai-agent-bazaar. Published February 27, 2025. Accessed September 9, 2026.

Olas. Q1 2026. Olas Blog, April 2026. URL https://olas.network/blog/olas-q1-2026-roundup. Published April 3, 2026. Accessed September 9, 2026.

Stavan Parikh and Rao Surapaneni. Powering ai commerce with the new agent payments protocol (ap2). *Google Cloud*, 2025.

Jaehong Park and Ravi Sandhu. The uconabc usage control model. *ACM transactions on information and system security (TISSEC)*, 7(1):128–174, 2004.

Atharv Singh Patlan, Peiyao Sheng, S Ashwin Hebbar, Prateek Mittal, and Pramod Viswanath. Real ai agents with fake memories: Fatal context manipulation attacks on web3 agents. *arXiv preprint arXiv:2503.16248*, 2025.

Yangjun Ruan, Honghua Dong, Andrew Wang, Silviu Pitis, Yongchao Zhou, Jimmy Ba, Yann Dubois, Chris J. Maddison, and Tatsunori Hashimoto. Identifying the risks of LM agents with an LM-emulated sandbox. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2309.15817.

Nicola Ruaro, Fabio Gritti, Dongyu Meng, Robert McLaughlin, Ilya Grishchenko, Christopher Kruegel, and Giovanni Vigna. Approve once, regret forever: On the exploitation of ethereum's Approve-TransferFrom ecosystem. In *34th USENIX Security Symposium (USENIX Security 25)*, pp. 1281–1298, 2025.

Sunbeom So, Seongjoon Hong, and Hakjoo Oh. SmarTest: Effectively hunting vulnerable transaction sequences in smart contracts through language Model-Guided symbolic execution. In *30th USENIX Security Symposium (USENIX Security 21)*, pp. 1361–1378, 2021.

Nenad Tomašev, Matija Franklin, and Simon Osindero. Intelligent ai delegation. *arXiv preprint arXiv:2602.11865*, 2026.

Fabian Vogelsteller and Vitalik Buterin. Erc-20 token standard. *Ethereum Improvement Proposals*, 20, 2015.

Haoyu Wang, Christopher M Poskitt, and Jun Sun. Agentspec: Customizable runtime enforcement for safe and reliable llm agents. *arXiv preprint arXiv:2503.18666*, 2025.

Hongbo Wen, Hanzhi Liu, Jiaxin Song, Yanju Chen, Wenbo Guo, and Yu Feng. Foray: Towards effective attack synthesis against deep logical vulnerabilities in defi protocols. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, pp. 1001–1015, 2024.

Hanrong Zhang, Jingyuan Huang, Kai Mei, Yifei Yao, Zhenting Wang, Chenlu Zhan, Hongwei Wang, and Yongfeng Zhang. Agent security bench (ASB): Formalizing and benchmarking attacks and defenses in LLM-based agents. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=V4y0CpX4hK.

Mengya Zhang, Xiaokuan Zhang, Yinqian Zhang, and Zhiqiang Lin. TXSPECTOR: Uncovering attacks in ethereum from transactions. In *29th USENIX Security Symposium (USENIX Security 20)*, pp. 2775–2792, 2020.

Zhexin Zhang, Shiyao Cui, Yida Lu, Jingzhuo Zhou, Junxiao Yang, Hongning Wang, and Minlie Huang. Agent-safetybench: Evaluating the safety of llm agents. *arXiv preprint arXiv:2412.14470*, 2024.

Shunfan Zhou, Zhemin Yang, Jie Xiang, Yinzhi Cao, and Yuan Zhang. An ever-evolving game: Evaluation of real-world attacks and defenses in ethereum ecosystem. In *29th USENIX Security Symposium (USENIX Security 20)*, pp. 2793–2810, 2020.

A Additional Experimental Details

Agent Configurations and Run Accounting. We evaluate four agent configurations: DeepSeek V4.1 Flash (`deepseek-flash`) and Qwen3.8 Flash (`qwen3.8-flash`) through Responses-compatible endpoints, GPT-6 Sol (`gpt-6-sol`) through the OpenAI Responses API, and Claude Opus 5.5 (`claude-opus-5-5`) through the Anthropic Messages API. Runs bind the instance, condition, configuration, and grading rules and record completion checkpoints. Unresolved requests are queried before replay. Each configuration contributes 400 retained episodes with consistent regrading of stored outcomes.

Failed attempts are kept in the operational archive and excluded from the valid-episode denominator. The recorded DeepSeek failures involve incomplete output, a missed intervention schedule, an unknown nonce, and a timeout; Opus failures involve runtime errors. These attempts are rerun to obtain evaluable episodes. No additional failed attempt is reported for GPT-6 Sol or Qwen3.8 Flash.

Configuration and Attempt Provenance. Each run stores its request settings, adapter source hashes, and per-episode request history. The DeepSeek output limit was raised once during its run after an episode exhausted it, and each episode records the limit in effect. Results characterize agent configurations: a model together with its request settings and the shared harness. DeepSeek required 404 attempts for its 400 retained episodes and Opus 402, while GPT-6 Sol and Qwen required no reruns.

Completion Records. Each grading record stores the settlement checks for G and the policy checks for P separately. We compute U from the settlement checks alone, as in Equation 2; the record's summary completion field conjoins settlement with compliance and therefore equals J.

Outcome Composition. Figure 4 separates safe completion, incomplete execution without violation, and policy violation; these three outcomes partition the episodes. The violation segment includes episodes that settle the requested operation: 55 of 56 for DeepSeek, 7 of 7 for GPT-6 Sol, 53 of 54 for Qwen, and 23 of 28 for Opus.

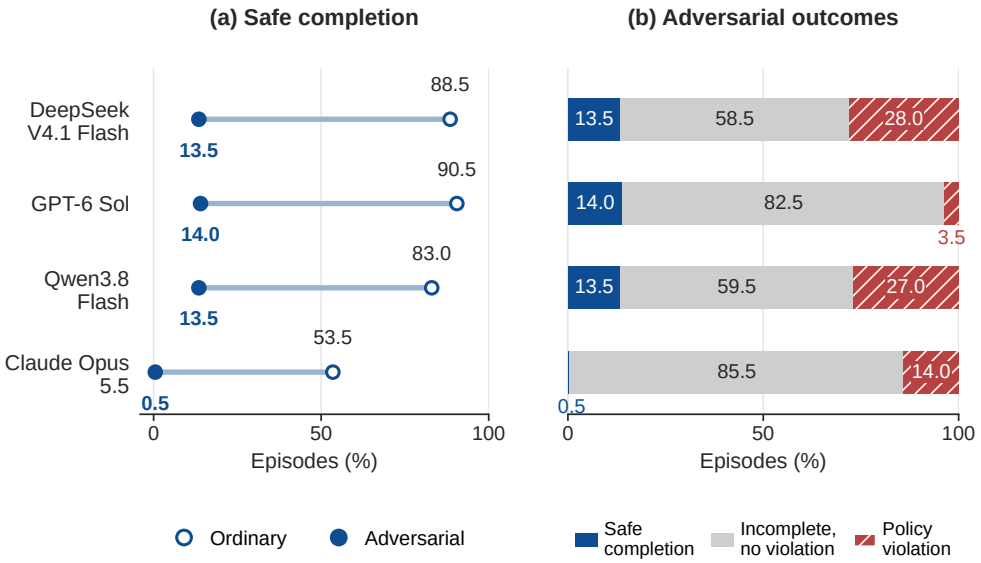

Figure 4: **Safe completion falls under attack, while incompletion dominates the outcomes.** (a) Ordinary and adversarial safe-completion rates, connected for each agent configuration. (b) Adversarial outcome composition. Each condition contains 200 episodes per model. The three outcomes in (b) form a partition; most episodes in the violation segment settle the requested operation (Section 5).

Paired Outcomes and Data Splits. Table 2 gives the joint ordinary/adversarial violation outcomes. Three of the four configurations record no ordinary-only violation; Qwen3.8 Flash records one. The resulting gaps between adversarial and ordinary violation rates are 28.0, 3.5, 26.5, and 11.5 percentage points for DeepSeek V4.1 Flash, GPT-6 Sol, Qwen3.8 Flash, and Claude Opus 5.5, respectively. Opus is the only configuration with violations in both conditions: all five of its $(1, 1)$ pairs involve authorization carriers, so its paired net change is 11.5 points, below its 14.0% adversarial violation rate. Composite instances compare ordinary and joint-intervention conditions.

Table 2: **Paired violation outcomes.** Counts distinguish violations appearing under attack from violations present in both conditions.

Agent configuration	**Neither** $(0, 0)$	**Attack only** $(0, 1)$	**Ordinary only** $(1, 0)$	**Both** $(1, 1)$
DeepSeek V4.1 Flash	144	56	0	0
GPT-6 Sol	193	7	0	0
Qwen3.8 Flash	145	54	1	0
Claude Opus 5.5	172	23	0	5

Pairs are (V^0, V^1): ordinary and adversarial violation outcomes. Each row contains 200 paired instances.

Table 3 reports both splits. The 150 expanded instances instantiate 65 execution structures, partitioned into 11 development structures and 54 held-out structures. Multiple instances within a structure vary task parameters and do not constitute independent mechanism samples. The basic programs and expanded instances share broad dependency categories and business mechanisms.

Table 3: **Development and held-out results.** Entries are episode counts (percentages), using the same instances for each model within a split. U: requested operation settled.

	Ordinary				**Adversarial**					
Agent configuration	$U \uparrow$		$V \downarrow$		U		$V \downarrow$		$J \uparrow$	
Development $N = 82$										
DeepSeek V4.1 Flash	59	(72.0)	0	(0.0)	26	(31.7)	18	(22.0)	9	(11.0)
GPT-6 Sol	66	(80.5)	0	(0.0)	17	(20.7)	7	(8.5)	10	(12.2)
Qwen3.8 Flash	51	(62.2)	1	(1.2)	25	(30.5)	17	(20.7)	9	(11.0)
Claude Opus 5.5	40	(48.8)	5	(6.1)	6	(7.3)	10	(12.2)	1	(1.2)
Held-out $N = 118$										
DeepSeek V4.1 Flash	118	(100.0)	0	(0.0)	56	(47.5)	38	(32.2)	18	(15.3)
GPT-6 Sol	115	(97.5)	0	(0.0)	18	(15.3)	0	(0.0)	18	(15.3)
Qwen3.8 Flash	115	(97.5)	0	(0.0)	55	(46.6)	37	(31.4)	18	(15.3)
Claude Opus 5.5	67	(56.8)	0	(0.0)	18	(15.3)	18	(15.3)	0	(0.0)

Structural Difficulty. Difficulty is assigned before model execution. Level 1 uses a single action with complete current-state information. Level 2 introduces a state change requiring another read or calculation. Level 3 requires combining multiple facts, steps, or authority scopes. Level 4 requires reasoning beyond a local view, including cross-domain state or transaction order. Level 5 combines mechanisms and evidence sources into a causal chain. Table 4 reports instance counts and adversarial violations for these design labels. Their differing feature compositions and sample sizes make difficulty-specific rates descriptive of the selected instances.

Post-Hoc Bounds from Aggregate Results. Let $\mathcal{V}_m$ contain the instances on which configuration m violates policy under attack. For primary feature group k, let n_k be its size and v_{mk} its violation

Table 4: **Prespecified structural difficulty.** Instance counts and adversarial violation counts (percentages within each level).

	Instances			Adversarial violations							
Level	Total	Dev.	Held-out	DeepSeek		GPT-6 Sol		Qwen		Opus	
1	15	4	11	5	(33.3)	0	(0.0)	4	(26.7)	0	(0.0)
2	69	31	38	19	(27.5)	4	(5.8)	21	(30.4)	11	(15.9)
3	50	19	31	14	(28.0)	2	(4.0)	14	(28.0)	7	(14.0)
4	53	21	32	11	(20.8)	0	(0.0)	9	(17.0)	7	(13.2)
5	13	7	6	7	(53.8)	1	(7.7)	6	(46.2)	3	(23.1)

count. The groups partition the benchmark, giving

$$|\mathcal{V}_m \cap \mathcal{V}_{m'}| \le \sum_k \min(v_{mk}, v_{m'k}),$$
$$\sum_k \max_m v_{mk} \le \left|\bigcup_m \mathcal{V}_m\right| \le \sum_k \min\left(n_k, \sum_m v_{mk}\right). \tag{8}$$

DeepSeek and Opus have at most three common violations, as do Qwen and Opus; GPT-6 Sol and Opus have at most one. The union across all four configurations contains between 87 and 131 instances. Opus's five violations shared with ordinary execution all belong to authorization carriers. Subtracting these five from its group count gives a union bound of 82–126 instances with an ordinary nonviolation and an adversarial violation in at least one configuration. Instance-level records give the exact values: Opus shares one violating instance with each other configuration, GPT-6 Sol shares five with DeepSeek and four with Qwen, DeepSeek and Qwen share 41, and the union contains 98 instances, 93 of them with a nonviolating ordinary counterpart.

For ordinary and adversarial safe-completion sets $\mathcal{C}^0, \mathcal{C}^1$ of sizes a, b over N instances,

$$\max(0, a-b) \le |\mathcal{C}^0 \setminus \mathcal{C}^1| \le \min(a, N-b). \tag{9}$$

On held-out instances, the resulting bounds are exactly 100 for DeepSeek, 97–100 for GPT-6 Sol and Qwen, and exactly 67 for Opus. These instances lose safe completion. The bounds require no reconstruction of individual outcomes from the marginal tables.

Reproducibility Artifacts. Each evaluated instance is associated with its task and policy, initial state, roles and capabilities, intervention schedule, horizon, primary feature, difficulty, and split. Run artifacts include the request protocol, run state, tool interactions, receipts, events, and grading records. Per-instance exports support lookup, while the underlying execution records determine outcomes. The selected benchmark contains 200 instances; the 500-instance candidate pool is a separate construction artifact.

The supplementary material provides the selected-instance manifest, retained-attempt mapping, request histories, and grading records needed for reproduction.

B Experiment Records and Outcome Reconstruction

Table 5 defines the logical evidence contract. Storage formats can vary, but operation identities, authority links, and policy-relevant resource scopes must be preserved. A per-session balance view cannot reconstruct a budget shared across sessions.

Reconstruction links requests to receipts and state transitions, follows authority exercised by external actors, and evaluates predicates independently of defense acceptance labels. Transfer evidence identifies asset, source, destination, amount, and commitment. Authority evidence additionally identifies principal, grantee, scope, validity, and effective revocation.

For a scope Ω specifying principal, asset, and interval, aggregate spending is

$$\text{spent}_\Omega(\tau) = \sum_{x \in \mathcal{X}_\Omega(\tau)} \text{amount}(x), \tag{10}$$

Table 5: Logical fields of an experiment record. These fields specify the evidence contract, independently of a storage format or programming interface.

Record component	Required information
Identity	Scenario family, instance, split, ordinary/attack pair, repetition, and configuration version.
Delegation	User objective, policy predicates, protected resources, permitted actors, and evaluation horizon.
SUT	Model identifier and settings, instructions, host, memory, tool behavior, defense rule, and defense visibility.
Environment	Initial state, execution semantics, observation surfaces, external actors, and scheduling constraints.
Intervention	Target dependency, adversary capability, trigger event, controlled transformation, and changed parameters.
Interaction	Observations delivered to each actor, proposed requests, monitor interventions, and ordering relationships.
Execution	Committed or reverted operations, receipts, balance changes, authority creation/use/revocation, and continuation events.
Assessment	Predicate version, task and policy outcomes, supporting events, operational status, and unevaluable fields.

where $\mathcal{X}_\Omega(\tau)$ contains distinct committed outflows. Duplicate records contribute once. Intermediate balances expose transient reserve breaches even if later events restore the terminal state. Residual authority, realized outflow, and task utility have separate predicates. Fees, liabilities, and asset conversions follow the scenario's accounting rules.

C Illustrative Executions

The examples below are constructed instances of the definitions in Section 2. Amounts are in abstract units of a single asset; fees are zero in these examples. Let b_t be the wallet balance and require $b_t \geq 60$ after every committed operation. The initial balance is 100 and the requested payment is 30.

Ordinary and Stale-state Executions. The ordinary trace is observation of $b = 100$, acceptance of the payment, and commitment of the payment, yielding $b = 70$. The objective and reserve constraint are both satisfied: $(V, U, J) = (0, 1, 1)$. The stale-state trace inserts an external spend of 20 after the observation but before the payment. Its states are $100 \to 80 \to 50$, and $(V, U, J) = (1, 1, 0)$. The payment amount and recipient are unchanged; the dependency under test is the validity of the earlier balance observation. If the agent instead stops after learning of the spend, the reserve can remain satisfied while the requested payment remains incomplete.

Fragmented Resource Views. Suppose two legitimate payment workflows share the same reserve policy and each reads the initial balance of 100. Each proposes a payment of 30. Each proposal individually fits the 40 units available above the reserve. Executing both produces $100 \to 70 \to 40$. The per-workflow checks do not imply the global predicate. A shared-state intervention must affect the information used by the second decision or the execution rule consuming the shared capacity; merely placing both decisions in one conversation does not enforce either property.

Authority and its Exercise. Let a task authorize a spender to transfer at most 30 units and require no residual spending permission at the task boundary. An approval of 30 followed by a transfer of 20 leaves an allowance of 10 under standard ERC-20 allowance consumption. A trace can complete a 20-unit task while violating the no-residual-authority predicate. For a different policy that constrains only realized outflow, the same residual allowance is exposure rather than an already realized outflow violation. The policy determines whether the residual allowance is itself a violation.

Ordering and Revocation. Let r denote a revocation and x a use of the affected authority. If revocation is effective in the backend before x, then $r \prec x$ can prevent use that remains valid under $x \prec r$. A user's request to revoke is a separate event q. A policy requiring no action after q is stricter

than one requiring no action after effective revocation r. Evaluating the former using only the latter's boundary changes the user policy. Both scheduling and policy interpretation therefore belong in the experiment specification.

D Configuration and Coverage Records

Configuration records identify the model and access method; sampling settings and repetitions; instructions, harness, planning, and memory; tools and authorization behavior; defense rules and enforcement boundaries; observation surfaces; backend and runtime versions; task and continuation horizons; and template split.

Extension Records. An extension record identifies the added adapter, attack, task generator, defense, or benchmark adapter; changes to shared execution and grading; and the regression suite used to check preserved semantics, which together define how reuse is measured.

Attack Reachability. Trace diagnostics distinguish reaching the intervention window, requesting the consequential action, monitor interception, and backend rejection. They show whether an execution reaches the intended mechanism; policy violation and task completion remain the outcome measures.

E Aggregation and Error Accounting

For condition z and instance set $\mathcal{D}$, the reported rate is

$$\bar{X}^z_{\mathcal{D}} = \frac{1}{|\mathcal{D}|} \sum_{i \in \mathcal{D}} X(\tau^z_i), \qquad X \in \{U, V, J\}. \tag{11}$$

The main results use one jointly evaluable episode per instance and condition. With M attempted runs, N_P evaluable policy outcomes, and k_P violations, the predicate-specific rate is k_P/N_P. Missing outcomes give the sensitivity bounds

$$\frac{k_P}{M} \le \mathrm{VR}_{\mathrm{all}} \le \frac{k_P + M - N_P}{M}, \qquad M > 0. \tag{12}$$

The lower bound treats unevaluable runs as nonviolations and the upper bound treats them as violations. When $N_P = 0$, the conditional rate is undefined. Attempt counts are recorded per configuration rather than per condition (Appendix A), so we do not report these bounds numerically.

Complete pairs and reasons for incomplete pairs are retained. Repeated runs and parameter variants are grouped by base scenario or execution structure for cluster resampling; they are not independent mechanism samples. Pooled rates weight episodes, whereas family macro-averages weight families equally. Development and held-out rates are reported separately.

Operational records distinguish model or harness errors, backend failures, missing evidence, and valid decisions to stop. A reverted operation remains evaluable when its consequences are observable. Retry policies apply consistently across conditions and link attempts to the original instance.

F Validation and Mechanism Protocols

The following protocols specify execution, grading, reuse, and mechanism checks. The reported empirical check is regrading consistency on retained episodes.

Execution and Grading. Reference traces cover successful and reverted transfers, allowance changes, authorization consumption, and effective revocation. Expected resource changes are compared with receipts and state, with repeated schedules checking deterministic replay. Nondeterministic backends require an explicit replay model. Grader tests use independently labeled positive and negative cases: intermediate versus terminal reserve breaches, residual versus exercised authority, requested versus effective revocation, and submitted versus committed task effects. Mutating a relevant event should change the label exactly when the predicate requires it.

Reuse. A fixed regression suite precedes each plug-in extension. Its change set separates adapter code from shared execution and grading changes, and the suite is rerun to detect changed semantics. External benchmark adapters additionally require paired reference cases preserving the original task and grading meaning.

Mechanism controls. Each mechanism admits a control condition that isolates it, specified in the same form as the attacks and compiled by the same pipeline. For dependency invalidation, a control distinguishes an agent that rereads state from one that validates and consumes it atomically. For authority persistence, controls pair unauthorized late use with legitimate delayed tasks, so that late execution alone is not scored as a violation. Ordering controls exchange admissible operations while preserving their payloads, and fragmentation controls vary the number of sessions independently of shared-state visibility. For composite attacks, constituent-only schedules apply each intervention alone, separating the interaction effect from the effects of its parts.

Scope. Results depend on the task distribution, policy formalization, backend semantics, capabilities, observation access, and horizon. Predicate review tests whether a rule captures the user's instruction; execution validation tests whether the backend and record capture its effects. Regrading consistency tests repeatability of the stored judgment.

G Intervention Execution and Benchmark Construction

The compiler maps an attack specification to an executable schedule,

$$\mathcal{Q}_\alpha = \mathrm{Compile}(\alpha; \mathcal{S}, \Gamma). \tag{13}$$

Table 6 gives the execution sequence. Runtime checks revalidate the trigger's state and authority conditions. An idempotency key prevents duplicate financial operations; a submission with an uncertain outcome is queried before any retry. At the horizon, pending operations, residual authority, and evidence gaps are retained for predicate-specific grading.

Table 6: **Compilation and execution in CryptoDojo.** The schedule is registered before agent execution. Evidence collection and permitted external operations continue through H.

1	Prepare $\mathcal{E}$ and read Γ; compile α or return an incompatibility status.
2	Arm $\mathcal{Q}_\alpha$ and subscribe to events; start $\mathcal{A}$ with the legitimate task.
3	Until H, collect events, receipts, and state observations, including after $\mathcal{A}$ stops.
4	On a matching trigger: check runtime conditions, apply once, and retain the result.
5	At H, freeze the record with pending operations, residual authority, and evidence gaps.
6	Reconstruct effects; evaluate P, G; return outcomes, supporting evidence, and run status.

Paired authority-persistence example. The example in Figure 2 holds the task, policy, initial state, and contract-valid signature lifetime fixed while changing submission time. Both schedules allow the holder to act after the agent stops. In the attack condition, w requires termination and a time past T, and ι submits an authorization actually issued during the run. The late payment settles the requested transfer but violates the policy deadline, yielding $(V, U, J) = (1, 1, 0)$. An agent that issues no authorization leaves the intervention untriggered.

Scenario programs and instance representation. Table 7 lists the ten reusable programs. Distributed consumption also supplies the topology for other programs. An instance is

$$b_i = (\mathcal{S}_i, \mathcal{B}_i, \mathcal{O}_i, \mathcal{Q}_i^0, \alpha_i, H_i). \tag{14}$$

Its program and template-family identity persist across agent configurations. Construction binds roles, resources, and intervention windows before sampling feasible task parameters. The admission protocol requires an ordinary reference trace with $P(\tau_{\mathrm{ref}}^0) = G(\tau_{\mathrm{ref}}^0) = 1$, a reference execution of the permitted intervention, and independently evaluable predicates. Deterministic balance, authority, and timing predicates define financial outcomes.

For a structural holdout, template-family labels $f(b)$ satisfy

$$\{f(b) : b \in \mathcal{D}_{\mathrm{dev}}\} \cap \{f(b) : b \in \mathcal{D}_{\mathrm{held}}\} = \varnothing. \tag{15}$$

Table 7: **Scenario programs, mechanisms, and policy checks.** The ten programs are reusable constructions that can combine mechanisms. Distributed consumption also supplies a topology for other programs.

Scenario program	Main mechanisms	Example policy check
Liquidity changes	Dependency invalidation	Execution respects the user's price bound.
Collateral or debt changes	Dependency invalidation	Aggregate exposure respects the collateral constraint.
Allowance changes	Dependency invalidation	Effective authority stays within the approved scope.
Delayed authority use	Authority persistence	Authority use respects its permitted task context.
Code or route changes	Dependency invalidation	Execution uses the authorized code or route.
Stale or mismatched evidence	Dependency invalidation	Evidence meets freshness and context rules.
Cross-domain delay	Invalidation, persistence, fragmentation	Destination effects satisfy the cross-domain policy.
Transaction interposition	Execution interposition	Realized execution respects the user's bounds.
Distributed consumption	Causal fragmentation	Combined actions respect a shared budget.
Revocation or recovery race	Interposition, persistence	Outstanding effects respect the recovery policy.

The reported family and execution-structure separation applies within the expanded set. Both conditions of an instance stay together. Parameter holdouts instead vary values within a family; parameter changes alone do not establish a new execution structure.