PACT: WHAT AGENTS SEE MUST NOT DECIDE WHAT THEIR TOOLS DO

Anonymous authors*
Paper under double-blind review

ABSTRACT

External content may inform the tool call an agent proposes, but it must not grant permission to execute that call. Before a call runs, the Provenance-Aware Contract for Tools (PACT) checks its operation, destination, arguments, sources, evidence, and approval against a task contract, an executable list of what the trusted request permits. With complete contracts supplied by the task, recorded sources, interception of every registered call that changes or discloses application state (a protected call), and permitted alternatives, PACT blocks all 1,500 controlled attacks and completes all 2,000 tasks without human intervention. On the same 100 adapted ARPIbench cases, the attack success rate (ASR) in executed protected calls falls from 2%–85% to 0 for each of eight models. On all 7,560 cases with DeepSeek V4 Flash, ASR falls from 58.4% to zero while permitted tool calls still run. On AgentDojo, PACT ties the best task success under attack and the best safe success, and no protected call requested by an attack executes. Building complete contracts is harder. A 72B model that generates contracts from the request finds 70/72 write tools but correctly constrains only 20/223 argument fields, and PACT enforces such incomplete permissions exactly as written.

1 INTRODUCTION

A user asks an agent to send a report to a specified recipient. A webpage it reads tells it to send the report elsewhere. The agent sees both instructions, but only the user may choose the recipient. Indirect prompt injection exploits this gap between what an agent reads and what it may do (Greshake et al., 2023; Yi et al., 2023). The same attack can arrive through documents, user interfaces, tool output, Model Context Protocol (MCP) metadata, and saved memory.

Input defenses try to keep malicious instructions from changing the model's proposed call. Execution defenses also enforce permissions. CaMeL keeps untrusted data out of control flow and restricts data use (Debenedetti et al., 2025), and Progent checks tool names and arguments against policies (Shi et al., 2025a). We study how task permissions and records of where data came from should decide both a proposed call and its replacement, so blocking an attack need not end the task.

PACT combines a task contract with source records and checked recovery. The task contract lists the operations, destinations, argument values, and data uses that the trusted request and application rules permit; external content cannot change it. Source records note where each piece of content entered the agent and which records it came from. The model proposes how to complete the task, and PACT checks each call and its recorded sources before execution. In recovery, a rejected call is replaced by a permitted alternative that must pass the same check. When the contract, the source records, and the dispatcher (the component that executes tool calls) are correct, every executed call satisfies the stated permissions.

Contributions.

1. An executable task contract that checks each concrete call's operation, destination, arguments, and data use before it runs, so the contract, rather than the model's proposal, grants permission.
2. One checker for proposed and replacement calls that uses recorded sources, logs source paths and failed rules, and keeps recovery from bypassing permissions.

*Code: `https://anonymous.4open.science/r/PACT-864F/`

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.

3. With supplied contracts, enforcement holds across eight models and every controlled task is completed. Generated contracts show that finding the right tools is not enough without correct argument constraints.

2 PACT

Problem setting. A task has a trusted user request, observations that an attacker may control, and application permissions. A protected action is a registered tool call whose operation, destination, arguments, or resources can change or disclose application state. An attack succeeds when observed content leads the agent to execute a protected action that the task does not permit. The attacker may make the model propose any call but cannot alter the trusted request, the contract, the tool schemas, the dispatcher, or the source labels assigned by adapters (the wrappers that bring inputs into the agent and record them). The final text answer is covered only if it is sent through a registered tool.

Overview. Figure 1 separates permission from proposal. The trusted request and application rules define the contract (1). External content enters the agent and may change its proposed call (2), while adapters record the sources carried through observations, derived records, and calls (3). The checker brings these inputs together (4): execution requires every condition to pass, and recovery sends a new candidate and its sources back through the same checks.

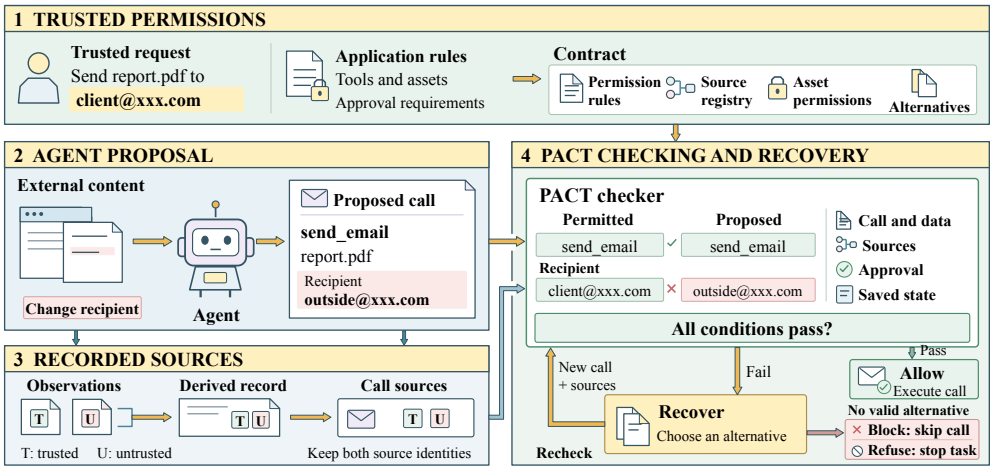

Figure 1: PACT checks permission before execution. External content changes the proposed recipient from `client@xxx.com` to `outside@xxx.com`. The operation still matches, but the recipient does not, so the call is rejected. Source records retain both trusted (T) and untrusted (U) sources when content is combined. All conditions feed one decision; recovery selects an alternative that must pass the same checker. If no alternative passes, PACT skips the call or stops the task.

2.1 Contracts and source records

Building a contract. A contract is either supplied, that is, written from the benchmark's task specification, or generated from the trusted request. The generator sees only that request and the public tool schemas, before any external content is loaded. It predicts the permitted tools, the argument constraints, and whether the agent may replan after a rejection; the expected calls stay hidden until scoring. Supplied contracts test enforcement alone, and generated contracts test how well permissions can be built. In the report example, the trusted request fixes the recipient.

Trace graph. The trace graph records where each piece of content came from. When content enters the agent, its adapter creates a source record (a leaf of the graph) with a trust class, trusted or untrusted, and an input channel such as a document or tool output. A record derived from other records stores its parent links, its source set (all sources it descends from), the adapter, and a content hash. Parsing, saving, relaying, concatenating, and retrieving all give the new record the union of its

parents' source sets. The T and U markers in Figure 1 illustrate this rule: combining records retains both source identities rather than making the result trusted. The action adapter fills each proposed call's source set from the recorded inputs. PACT checks this recorded ancestry and does not infer which inputs influenced the model internally.

Action contract. Let $G = (V, E)$ denote the trace graph, with records V and parent links E. A proposed action $a = (n, \mathrm{args}, s, S, \alpha)$ specifies operation n, arguments args, destination s, source set S, and approval state α. The contract $\Gamma = (C, L, A, Q)$ contains permission rules C, a source registry L, asset permissions A, and candidate alternatives Q (permitted replacement calls). $C = (n^*, P, F, E^*, K, \rho)$ specifies the permitted operation, argument constraints, forbidden destinations, required evidence (sources that must support the call), allowed channels, and approval requirement. L records sources and channels, and A records each protected asset's source and allowed destinations. U_G contains the sources of untrusted records that were saved or relayed. We use Args_P for the argument constraints and $\mathrm{Sources}_\Gamma$ for the source, data use, channel, and evidence conditions. Then

$$\begin{aligned}\mathrm{Allow}_\Gamma(a, G) \iff\ & n = n^* \wedge s \notin F \wedge \mathrm{Args}_P(\mathrm{args}) \\ & \wedge \mathrm{Sources}_\Gamma(a) \wedge (\neg\rho \vee \alpha) \\ & \wedge S \cap U_G = \varnothing.\end{aligned} \tag{1}$$

Args_P holds when $\mathrm{args}[k] \in P(k)$ for every constrained argument present in the call. $\mathrm{Sources}_\Gamma$ holds when (i) every member of S appears in L and is either trusted or the registered source of a protected asset, (ii) any protected asset in args goes only to a destination that A allows for it, (iii) the channels recorded for S belong to K, and (iv) $E^* \subseteq S$. The final two terms require approval when the contract asks for it and reject calls that draw on untrusted saved or relayed content. Tool schemas separately check required keys and types. If a field needed for safety is missing from P, any value of that field passes; the contract is then incomplete, and the call is not rejected by default. Source trust and asset destinations are read only from L and A; copies of them elsewhere in the saved contract file do not affect the decision.

2.2 Checking and recovery

The trusted request does not have to specify every step. Each proposed call exposes its operation, destination, arguments, and supporting records for checking. In Figure 1, *Call and data* covers the operation, arguments, destination, and protected assets; *Sources* covers source permission, channels, and evidence; *Approval* and *Saved state* cover the remaining conditions in Equation 1. A call executes only if it passes all of these groups. In the email example, `send_email` is a permitted operation, yet the call fails because its recipient is not permitted. Observations can supply values but never permission.

Algorithm 1 PACT checks a proposed call before execution.

Require: trace graph G, action a, contract Γ
$R \leftarrow \textsc{ResolveSources}(a, G)$
$D \leftarrow \textsc{FailedChecks}(a, R, \Gamma)$
if $D = \varnothing$ **then**
  return $\textsc{Allow}(a, R)$
end if
$\hat{a} \leftarrow \textsc{PermittedAlternative}(\Gamma)$
if $\hat{a} \neq \varnothing$ **then**
  $\hat{R} \leftarrow \textsc{ResolveSources}(\hat{a}, G)$
  if $\textsc{FailedChecks}(\hat{a}, \hat{R}, \Gamma) = \varnothing$ **then**
    return $\textsc{Replace}(\hat{a}, D)$
  end if
end if
if $\textsc{CanContinueWithout}(a)$ **then**
  return $\textsc{Block}(a, D)$
end if
return $\textsc{Refuse}(D)$

Decision and recovery. Algorithm 1 applies Equation 1 before a call is sent to its tool. It resolves the call's sources in the trace graph and executes the call only if every condition passes. After a rejection, recovery selects a candidate from Q. The return path in Figure 1 carries the replacement call and its own sources back to the checker: its sources are resolved again and every condition is reapplied before execution. If no alternative passes, PACT blocks the call (skips it) when the task can continue without it and otherwise refuses the task. Selecting an alternative therefore grants no permission by itself.

Guarantee. Algorithm 1 executes a call only on two paths, the original proposal and a replacement from Q. Both look up sources and require that no condition of Equation 1 fails. Every dispatched call therefore satisfies Allow_Γ, and because every call is checked, the whole sequence stays within the permissions in Γ. The guarantee assumes that every registered call that can change or disclose application state reaches Algorithm 1, that Γ states the intended permissions, and that G records sources correctly. The evaluation tests each assumption through contract enforcement and construction, source consistency, and call interception.

3 EVALUATION

We ask whether task permissions stop attacks without preventing useful work, and which parts of the design make this possible. Controlled tests isolate the checks and recovery, comparisons across models and inputs test whether enforcement stays the same, and generated contracts and public tasks show how permissions are obtained and used.

Supplied contracts. The main comparison gives every defense the same 1,500 attacked and 500 unattacked episodes, proposed actions, contracts, simulated assets, permitted alternatives, and success rules. It spans seven input channels, eight attack goals, and six trace stages at which attack content is processed. A separate set of 160 cases is built so that exactly one contract check fails in each case. Another 512 attacks vary the attack wording, and 128 unattacked cases with similar wording measure unnecessary refusals. Each defense in this comparison is a deterministic rule, called a control, so every row acts on the same saved proposal; the supplement lists each rule.

Planner and direct permission comparisons. Two reference designs use the same 2,000 episodes. A separated planner sees only the request and tool signatures, a parser without tool access reads fields from observations, and the contract checks each action. A direct permission control applies the same operation, destination, argument, source, channel, and approval conditions (Appendix B).

Calls proposed by different models. Eight target models receive the same first 100 ARPIbench cases with simulated tools. We also run a common set of 500 cases for four Qwen2.5 sizes and all 7,560 cases for DeepSeek V4 Flash (Becker, 2025). This adaptation counts executed protected and permitted calls; it does not measure task completion.

Public task integrations. AgentDojo, AutoDojo, FinVault, Agent Security Bench, and AgentThreatBench provide public tasks and evaluators with task outcomes. We report each benchmark separately because their tasks and success rules differ.

Contracts generated from requests. For AgentDojo v1.2, keyword rules, Qwen3-32B, and Qwen2.5-72B each predict permissions from the trusted request and tool schemas. We run the 72B contracts on 97 tasks without attacks and 949 attacked tasks; the model may replan after a rejection.

How source records are obtained. In the controlled suite, adapters assign source IDs to leaf records and calls from task, attack, evidence, and asset records. Each transformation passes on the union of its parents' sources, and links are never edited later. ARPIbench and public benchmarks decide on recorded calls and task contracts without rebuilding each argument's source graph. Generated contracts change permissions but not source assignment (Appendix B).

Measures. Utility is the task success rate. Attack success rate (ASR) is computed over attacked cases. Safe success counts a case only when the task succeeds and the attack fails. Leak rate is the fraction of attacked cases that send protected data to a forbidden destination. Intervention is the fraction of episodes that end in refusal or require human approval; automatic execution and automatic replacement add no intervention.

Pairing and uncertainty. Paired runs with and without a defense share episode, behavior, or task IDs and keep the base model, tools, step limit, and scorer unchanged, so only the defense differs. Rates from different benchmarks are never pooled. Controlled cases are deterministic, and model

decoding uses temperature 0 except in one stated sensitivity test. We report Wilson confidence bounds, a paired exact test for live AgentDojo safety, and bootstrap intervals grouped by user task. Appendix B and the model results appendix give full settings.

AgentDojo setup. The defense comparison uses AgentDojo 1.2.2, Qwen3-4B-Instruct-2507, the "important instructions" attack, seed 42, 50 attacked tasks across four suites, and 35 direct requests that contain no attack. All defenses share the target, tasks, and scorer; their classifier and monitor settings appear in Appendix E. The benchmark's reference calls and arguments supply the PACT permissions, and a rejected call returns to the same model for replanning. Table 3 separately evaluates generated contracts.

4 RESULTS

4.1 SAFETY AND TASK COMPLETION

Supplied permissions preserve useful actions. Refusal also stops an attack, so Table 1 asks whether permission checks can let tasks finish without human intervention. Every defense receives the same contract, sources, proposed actions, and permitted alternative, so only its response differs.

Table 1: Results with supplied contracts on the same 2,000 episodes (%). Every deterministic control receives the same calls and permitted alternative. ASR and leak rate coincide in this suite. Alternative reports use of a replacement, and Intervention reports refusal or human approval. PACT alone combines full utility with no attack, leak, refusal, or human intervention.

Defense	Utility ↑	ASR, Leak ↓	Refusal ↓	Alternative	Intervention ↓
No defense	25.0	100.0	**0**	0	**0**
Prompt reminder	25.0	80.0	15.0	0	15.0
Input delimiters	25.0	66.7	25.0	0	25.0
Scan metadata	25.0	85.7	10.7	0	10.7
Allow listed sources	25.0	85.7	10.7	0	10.7
Verify MCP source	25.0	85.7	10.7	0	10.7
Ask for approval	35.7	85.7	**0**	10.7	100.0
Reject untrusted input	25.0	**0**	75.0	0	75.0
Match action to task	25.0	**0**	75.0	0	75.0
Check sources and data use	25.0	**0**	75.0	0	75.0
Human approval	**100.0**	**0**	**0**	75.0	100.0
PACT	**100.0**	**0**	**0**	75.0	**0**

Without a defense, every attack succeeds. Input and MCP controls leave ASR between 66.7% and 85.7%. The three controls that check calls at run time reach ASR 0 by refusing 75% of proposals, so utility stays at 25%. Human approval and PACT both preserve full utility by replacing the rejected action in 75% of episodes, but human approval intervenes in every episode.

Both reference designs complete 2,000/2,000 tasks and execute 0/1,500 unauthorized actions. The direct control indicates that these permission conditions are enough for the measured actions. The separated planner indicates that untrusted observations can be kept out of the component that chooses protected actions. PACT matches their outcome and also records the failed condition, the call's source path, and the recovery decision.

Task success on AgentDojo. Public tasks test whether an agent can still finish when a rejected call forces it to make another plan. Table 2 compares six recent defenses and PACT on the same AgentDojo tasks. The 35 direct requests contain no attack, so they test whether a defense disrupts normal tasks.

Table 2: AgentDojo results (%) with Qwen3-4B-Instruct-2507 on the same 50 attacked tasks and 35 direct requests. Safe counts task success with attack failure; suite columns divide the attacked tasks. Bold marks the best value. The task specification supplies the PACT permissions.

Defense	Attacked tasks			Direct requests	Safe success by suite ↑			
	$N = 50$			$N = 35$	Banking	Slack	Travel	Workspace
	Task ↑	ASR ↓	Safe ↑	Task ↑	$N = 10$	$N = 2$	$N = 7$	$N = 31$
No defense	48.0	8.0	44.0	48.6	40.0	**50.0**	28.6	48.4
PIGuard	42.0	**0**	42.0	48.6	40.0	**50.0**	0	**51.6**
PromptArmor	**52.0**	6.0	**50.0**	48.6	**50.0**	**50.0**	42.9	**51.6**
PromptGuard	28.0	4.0	26.0	37.1	40.0	0	0	29.0
AgentWatcher	20.0	**0**	20.0	42.9	40.0	0	0	19.4
DataSentinel	**52.0**	6.0	**50.0**	48.6	40.0	**50.0**	**57.1**	**51.6**
GPTSafeguard	42.0	**0**	42.0	**54.3**	40.0	**50.0**	0	**51.6**
PACT	**52.0**	4.0	**50.0**	48.6	**50.0**	**50.0**	42.9	**51.6**

On attacked tasks, PACT ties PromptArmor and DataSentinel for the highest task success (52%) and safe success (50%). Its two attack successes are cases where the final text answer repeats attack text; no protected call requested by an attack executes (0/50). The three defenses with ASR 0 (PIGuard, AgentWatcher, and GPTSafeguard) reach 20–42% safe success. On direct requests, GPTSafeguard has the highest task success, 54.3% against 48.6% for PACT. By suite, PACT ties for the highest safe success on Banking, Slack, and Workspace; on Travel, DataSentinel is higher (57.1% against 42.9%).

We also test PACT on four other public benchmarks. On AutoDojo and FinVault, it removes all unsafe outcomes and raises utility. On Agent Security Bench and AgentThreatBench, it has small costs in utility or tokens. Appendix E gives the full results.

4.2 The roles of checks and recovery

The component tests separate two requirements for safe task completion. The checks must reject every call that violates a permission, and recovery must keep a permitted way to finish the task.

Different checks decide different cases. PACT has ten checks: source permission, task fit, destination permission, argument limits, input channel, approval, saved state permission, required evidence, replacement validation, and refusal handling. For each check, 16 targeted proposals fail only that check. All checks together block 160/160 proposals. Removing a proposal's only failed check makes it unsafe in 160/160 cases; removing any of the other nine leaves all 1,440 comparisons safe (Table 6). Several checks fail together in the full suite, where removing one would hide its role.

Recovery is checked again. Refusal and replacement both stop all measured attacks, but replacement raises utility from 25% to 100%. The alternative gets no extra trust for being a fallback. Its arguments are traced and checked against the same contract, so a replacement cannot violate the permission rule that rejected the original call.

4.3 Across models, attacks, and inputs

Changing an attack's wording or the proposing model should not change task permissions. We test enforcement across attack rewrites, models, task mixes, and input paths.

Attack wording. The controlled cases let us separate checking whether an action is permitted from recognizing attack text. We rewrite the attacks in eight different ways while leaving the contract and success rule unchanged. We also add unattacked cases that contain requests, warnings, and tool language resembling the attacks.

Figure 2 shows that a prompt reminder and input delimiters leave overall ASR at 80% and 75%. Rejecting untrusted input reaches ASR 0 but refuses all 128 unattacked cases, so its utility on those

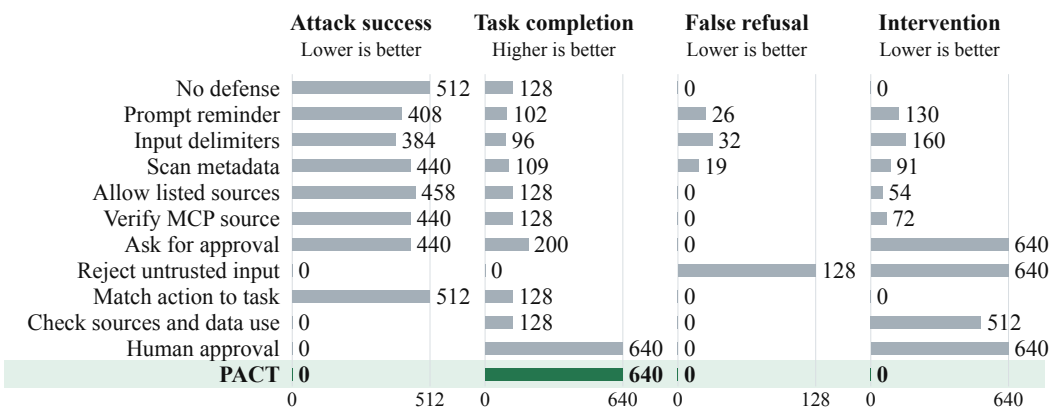

Figure 2: Counts across eight attack rewrites (64 attacks each) and 128 unattacked cases. Separate axes show successful attacks out of 512, completed tasks out of 640, false refusals out of 128, and interventions out of 640. PACT completes every task with no successful attack, false refusal, or intervention.

cases is 0. PACT reaches ASR 0 without refusing them. Because this result holds for all eight rewrites, it cannot be explained by fitting one attack template or by refusing everything.

Calls proposed by different models. In Figure 3, within each pair of runs with and without PACT, the prompt, cases, simulated tools, maximum steps, and success rule are identical. A case counts as a successful attack only when the model proposes and executes a protected read or post.



Figure 3: ASR, counted by executed protected calls, on the same first 100 adapted ARPIbench cases. Provider and open models share a 0–1 scale; PACT reduces ASR from 2%–85% to 0 for all eight models. This adaptation uses simulated tools and does not define task success.

Each paired PACT run records 0/100 successful attacks. Across these 800 cases, 950 permitted calls execute and 694 protected calls are blocked. These executed calls do not show that the tasks were completed; Tables 1 and 2 measure task completion separately. On the common 500 cases, Qwen2.5-7B, 14B, 32B, and 72B have baseline ASR 33.0%, 48.4%, 68.2%, and 56.8%; every paired PACT row records 0/500. Across all 7,560 cases with DeepSeek V4 Flash, protected actions fall from 4,416 to 0 while 7,418 permitted calls execute. The 95% Wilson upper bound for 0/7,560 is 0.051%.

Changes in the task mix. Eight groups change the mix of input channels, attack goals, actions, destinations, protected assets, points where the attack enters the trace, and the proportion of tasks with attacks. Across 92 cases, including 78 attacks and 14 tasks without attacks, PACT records ASR 0, utility 100%, intervention 0, and no false rejection. These cases still come from the same task schema, so they test consistent use of the contract rather than arbitrary changes to the application.

Longer paths and combined inputs. Thirteen cases relay untrusted instructions across inputs, forge evidence or approval, poison saved state, and combine image and text records. Baseline ASR is 63.6%; PACT reduces it to 0 with utility 100%. In one case, neither of two input records alone

leads to the forbidden action, but their combination does. PACT follows the sources of the final arguments back to both records and rejects the call without judging either input on its own.

4.4 Generating complete contracts remains the main challenge

Table 3: Generated contracts on 97 AgentDojo requests. (a) Scores are percentages: Exact measures task-level tool sets; Calls counts reference calls whose tool name is permitted (82 calls). Coverage and Correct use all 223 expected fields; Accuracy uses only constrained fields. (b) Paired task outcomes; changes are in percentage points.

(a) Contract generation quality

Generator	Write tools and calls					Argument constraints		
	Precision	Recall	F1	Exact	Calls	Coverage	Correct	Accuracy
Keyword rules	34.4	59.7	43.7	37.1	58.5	No constraints generated		
Qwen3-32B	**80.5**	86.1	83.2	83.5	87.8	**13.0**	8.1	62.1
Qwen2.5-72B	72.9	**97.2**	**83.3**	**86.6**	**97.6**	11.7	**9.0**	**76.9**

(b) Live task outcomes with Qwen2.5-72B contracts

Outcome	Cases	No contract		Generated contract		Change
		Count	Rate (%)	Count	Rate (%)	
Task success, no attack ↑	97	72	74.2	67	69.1	−5.2
Task success, under attack ↑	949	659	69.4	628	66.2	−3.3
Attack success ↓	949	19	2.00	13	1.37	−0.63

Success with supplied contracts leaves open whether complete contracts can be generated. We measure how well generated contracts recover expected write tools (tools that change state) and argument permissions, and how they change safety and task success in AgentDojo runs (Table 3).

The keyword generator identifies 43/72 expected write tools and includes 82 unexpected tools, but constrains 0/223 expected argument fields. Qwen3-32B identifies 62/72 expected tools and includes 15 extra tools; it constrains 29/223 fields and gets 18/223 correct. Qwen2.5-72B identifies 70/72 expected tools and includes 26 extra tools; it constrains 26/223 fields and gets 20/223 correct. In live runs, the 72B contract prevents seven attacks and introduces one new attack. The paired exact test gives $p = 0.0703$, and the 95% bootstrap interval grouped by user task includes 0, so the reduction is not statistically significant. Utility also falls. The new attack occurs when, after one payment tool is rejected, the model switches to another payment tool whose arguments the contract does not constrain enough. PACT enforces the generated contract as written, so the observed failures come from incomplete argument permissions and from the plan the model chooses after a rejection.

Recovery after a rejected call. The live run rejects at least one call in 102 tasks. Only 35 of these tasks then finish through a permitted alternative. On tasks without attacks, one task that failed without a contract now succeeds, while six tasks that succeeded now fail. Under attack, utility falls by 31/949 tasks. Banking accounts for five of the seven prevented attacks and the one new attack; Workspace accounts for the other two prevented attacks. Travel starts at ASR 0 but loses task successes because the model often cannot find another permitted way to finish. Whether a rejection helps depends on how precise the contract is and on whether the model still has a useful permitted action to take.

4.5 Integration and checking cost

Source records and call coverage. We tamper with saved traces by deleting an adapter record, changing a trust label, rewiring a parent link, replacing a content hash, forging approval, spoofing a source ID, omitting required evidence, and hiding the untrusted source of a relayed message. The matching consistency check detects each of these eight changes. If PACT wraps only one agent runner, it sees one of the seven registered call paths. If it sits in a shared tool registry, it sees all registered paths, but a direct call to an unknown tool still bypasses the permission check.

Effects missing from the tool schema. Contracts cannot name effects absent from the tool schema. Across 600 cases per contract, blocking the unknown effect and asking the model to replan prevents every unsafe action with 70% utility. Of the three responses (allow, block, or block and replan), this gives the highest utility without allowing the unknown effect.

Trace consistency and latency. All 24,000 traces agree with their saved actions, sources, parent links, checks, and decisions. Both proposal classes have p99 checking time below 0.14 ms (Table 4; setup in Appendix G.1).

Table 4: Permission checking time (ms).

Proposal	p50	p95	p99
Permitted	0.045	0.086	0.139
Prohibited	0.043	0.089	0.109

5 Related Work

Indirect prompt injection lets retrieved data or tool observations redirect an agent. Early studies demonstrated the attack in applications and controlled model settings; later benchmarks cover tool use, public tasks, generated cases, and larger model panels (Greshake et al., 2023; Liu et al., 2023; Yi et al., 2023; Zhan et al., 2024; Debenedetti et al., 2024; Becker, 2025; Dziemian et al., 2026; Ma et al., 2026). Web, computer-use, sandbox, and visual environments add further applications and input channels (Zhou et al., 2023; Koh et al., 2024; Xie et al., 2024; Ruan et al., 2024; Wang et al., 2025b; Cao et al., 2025; Evtimov et al., 2025; Liao et al., 2025).

Many defenses try to prevent attack text from controlling the model's proposal. They filter or structure inputs, train models to follow instruction priority, analyze intent, or repair a response before it is used (Wallace et al., 2024; Li et al., 2025; Shi et al., 2025b; Chen et al., 2025a;b; Wang et al., 2026c; Debenedetti et al., 2025; Kang et al., 2025; Wang et al., 2026a). PACT instead checks the resulting call against the current task's permissions, so filtering proposals and authorizing calls can be used together.

A second line of work checks plans, dependencies, privileges, provenance (where data came from), or tool calls close to execution (Jia et al., 2025; An et al., 2025; Lin et al., 2026; Kim et al., 2025; Costa et al., 2025; Shi et al., 2025a; Zhong et al., 2025; Mou et al., 2026; Zhang et al., 2026; Sequeira et al., 2026; Ying et al., 2026; Wang et al., 2026b; Gong & Deng, 2026; Wang et al., 2025a). PACT asks whether each concrete operation, destination, argument, and use of protected data is permitted. Its source records show where supporting content entered the agent, and observed content cannot grant the permission needed to act.

6 Discussion

The experiments separate the model's choice of call from the permission to execute it. Input filters, instruction defenses, and plan checks can help a model choose a better call, and PACT checks the resulting operation, destination, arguments, sources, and approval before a protected call runs. With supplied contracts, it blocks every protected call from an attack while executing permitted alternatives across attack wording, input channels, source paths, and eight models. On AgentDojo tasks under attack, PACT ties the best of six recent defenses in task success and safe success.

Generated contracts show where the remaining errors arise. Qwen2.5-72B finds 70 of 72 expected write tools, but only 20 of 223 argument constraints are correct, and a payment call with too few constraints leads to the one new attack. Finding the right tools is not enough. Trusted requests and application schemas must define argument, destination, and source permissions; every external call must reach PACT; and each replacement must pass the same check. Sandboxes, proposal filters, and output checks can complement this design (Xiang et al., 2026).

7 Conclusion

PACT keeps what an agent reads apart from what it may do. With supplied contracts, it blocks every measured attack, and permitted alternatives keep all controlled tasks successful. Contracts generated from user requests still miss argument rules, and after such a contract rejects a call, the agent often fails to find another way to finish the task.

AI USE STATEMENT

AI tools assisted with drafting and polishing the manuscript and with retrieving and discovering related literature. The authors reviewed and verified all resulting text and references and take responsibility for the final manuscript.

REPRODUCIBILITY STATEMENT

Algorithm 1 and Equation 1 specify the permission check. Appendix A describes the executable records, Appendix B gives evaluation settings, and Appendix G describes saved records and validation. The supplementary code and data package contains implementation files, configurations, saved results, and commands for reproducing and checking the reported experiments.

REFERENCES

Hengyu An, Jinghuai Zhang, Tianyu Du, Chunyi Zhou, Qingming Li, Tao Lin, and Shouling Ji. IPIGuard: A novel tool dependency graph-based defense against indirect prompt injection in LLM agents. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 1023–1039, Suzhou, China, 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.emnlp-main.53. URL https://aclanthology.org/2025.emnlp-main.53/.

Alex Becker. ARPIbench: Agent reflected prompt injection benchmark dataset. Hugging Face dataset and project repository, 2025. URL https://huggingface.co/datasets/alexcbecker/ARPIbench.

Tri Cao, Bennett Lim, Yue Liu, Yuan Sui, Yuexin Li, Shumin Deng, Lin Lu, Nay Oo, Shuicheng Yan, and Bryan Hooi. VPI-Bench: Visual prompt injection attacks for computer-use agents, 2025. URL https://arxiv.org/abs/2506.02456.

Sizhe Chen, Julien Piet, Chawin Sitawarin, and David Wagner. StruQ: Defending against prompt injection with structured queries, 2025a. URL https://arxiv.org/abs/2402.06363.

Sizhe Chen, Arman Zharmagambetov, Saeed Mahloujifar, Kamalika Chaudhuri, David Wagner, and Chuan Guo. SecAlign: Defending against prompt injection with preference optimization, 2025b. URL https://arxiv.org/abs/2410.05451.

Manuel Costa, Boris Köpf, Aashish Kolluri, Andrew Paverd, Mark Russinovich, Ahmed Salem, Shruti Tople, Lukas Wutschitz, and Santiago Zanella-Béguelin. Securing AI agents with information-flow control, 2025. URL https://arxiv.org/abs/2505.23643.

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. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2024. URL https://openreview.net/forum?id=m1YYAQjO3w.

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, 2025. URL https://arxiv.org/abs/2503.18813.

Mateusz Dziemian, Maxwell Lin, Xiaohan Fu, Micha Nowak, Nick Winter, Eliot Jones, Andy Zou, Lama Ahmad, Kamalika Chaudhuri, Sahana Chennabasappa, Xander Davies, Lauren Deason, Benjamin L. Edelman, Tanner Emek, Ivan Evtimov, Jim Gust, Maia Hamin, Kat He, Klaudia Krawiecka, Riccardo Patana, Neil Perry, Troy Peterson, Xiangyu Qi, Javier Rando, Zifan Wang, Zihan Wang, Spencer Whitman, Eric Winsor, Arman Zharmagambetov, Matt Fredrikson, and Zico Kolter. How vulnerable are ai agents to indirect prompt injections? insights from a large-scale public competition, 2026. URL https://arxiv.org/abs/2603.15714.

Ivan Evtimov, Arman Zharmagambetov, Aaron Grattafiori, Chuan Guo, and Kamalika Chaudhuri. WASP: Benchmarking web agent security against prompt injection attacks, 2025. URL https://arxiv.org/abs/2504.18575.

Guangyu Gong and Zizhuang Deng. PlanGuard: Defending agents against indirect prompt injection via planning-based consistency verification, 2026. URL https://arxiv.org/abs/2604.10134.

Kai Greshake, Sahar Abdelnabi, Shailesh Mishra, Christoph Endres, Thorsten Holz, and Mario Fritz. 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*, pp. 79–90, 2023. doi: 10.1145/3605764.3623985. URL https://dl.acm.org/doi/10.1145/3605764.3623985.

Feiran Jia, Tong Wu, Xin Qin, and Anna Squicciarini. The task shield: Enforcing task alignment to defend against indirect prompt injection in LLM agents. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 29680–29697, Vienna, Austria, 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.acl-long.1435. URL https://aclanthology.org/2025.acl-long.1435/.

Mintong Kang, Chong Xiang, Sanjay Kariyappa, Chaowei Xiao, Bo Li, and G. Edward Suh. Mitigating indirect prompt injection via instruction-following intent analysis, 2025. URL https://arxiv.org/abs/2512.00966.

Juhee Kim, Woohyuk Choi, and Byoungyoung Lee. PFI: Prompt flow integrity to prevent privilege escalation in LLM agents, 2025. URL https://arxiv.org/abs/2503.15547.

Jing Yu Koh, Robert Lo, Lawrence Jang, Vikram Duvvur, Ming Chong Lim, Po-Yu Huang, Graham Neubig, Shuyan Zhou, Ruslan Salakhutdinov, and Daniel Fried. Visualwebarena: Evaluating multimodal agents on realistic visual web tasks. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, 2024. URL https://arxiv.org/abs/2401.13649.

Hao Li, Xiaogeng Liu, Ning Zhang, and Chaowei Xiao. PIGuard: Prompt injection guardrail via mitigating overdefense for free. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 30420–30437, Vienna, Austria, 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.acl-long.1468. URL https://aclanthology.org/2025.acl-long.1468/.

Zeyi Liao, Jaylen Jones, Linxi Jiang, Yuting Ning, Eric Fosler-Lussier, Yu Su, Zhiqiang Lin, and Huan Sun. RedTeamCUA: Realistic adversarial testing of computer-use agents in hybrid web-os environments, 2025. URL https://arxiv.org/abs/2505.21936.

Junda Lin, Zhaomeng Zhou, Zhi Zheng, Shuochen Liu, Tong Xu, Yong Chen, and Enhong Chen. VIGIL: Defending LLM agents against tool-stream injection via verify-before-commit. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9764–9785, San Diego, California, United States, 2026. Association for Computational Linguistics. doi: 10.18653/v1/2026.acl-long.443. URL https://aclanthology.org/2026.acl-long.443/.

Yi Liu, Gelei Deng, Yuekang Li, Kailong Wang, Zihao Wang, Xiaofeng Wang, Tianwei Zhang, Yepang Liu, Haoyu Wang, Yan Zheng, Leo Yu Zhang, and Yang Liu. Prompt injection attack against LLM-integrated applications, 2023. URL https://arxiv.org/abs/2306.05499.

Xinyi Ma, Dan Friedman, Jixuan Chen, Naama Friedman, Jianwei Yu, Mark Lee, Yizhe Zhang, Pin-Yu Chen, Ce Zhang, and Bo Li. Autodojo: Adaptive attack generation for benchmarking defense in LLM agents, 2026. URL https://arxiv.org/abs/2606.15057.

Yutao Mou, Zhangchi Xue, Lijun Li, Peiyang Liu, Shikun Zhang, Wei Ye, and Jing Shao. ToolSafe: Enhancing tool invocation safety of LLM-based agents via proactive step-level guardrail and feedback. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 37125–37153, San Diego, California, United States, 2026. Association for Computational Linguistics. doi: 10.18653/v1/2026.findings-acl.1850. URL https://aclanthology.org/2026.findings-acl.1850/.

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.

Rohan Sequeira, Stavros Damianakis, Umar Iqbal, and Konstantinos Psounis. Agent-Sentry: Bounding LLM agents via execution provenance, 2026. URL https://arxiv.org/abs/2603.22868.

Tianneng Shi, Jingxuan He, Zhun Wang, Hongwei Li, Linyu Wu, Wenbo Guo, and Dawn Song. Progent: Securing AI agents with privilege control, 2025a. URL https://arxiv.org/abs/2504.11703.

Tianneng Shi, Kaijie Zhu, Zhun Wang, Yuqi Jia, Will Cai, Weida Liang, Haonan Wang, Hend Alzahrani, Joshua Lu, Kenji Kawaguchi, Basel Alomair, Xuandong Zhao, William Yang Wang, Neil Gong, Wenbo Guo, and Dawn Song. PromptArmor: Simple yet effective prompt injection defenses, 2025b. URL https://arxiv.org/abs/2507.15219.

Eric Wallace, Kai Xiao, Reimar Leike, Lilian Weng, Johannes Heidecke, and Alex Beutel. The instruction hierarchy: Training LLMs to prioritize privileged instructions, 2024. URL https://arxiv.org/abs/2404.13208.

Che Wang, Fuyao Zhang, Jiaming Zhang, Ziqi Zhang, Yinghui Wang, Longtao Huang, Jianbo Gao, Zhong Chen, and Wei Yang Bryan Lim. ICON: Indirect prompt injection defense for agents based on inference-time correction, 2026a. URL https://arxiv.org/abs/2602.20708.

Haoyu Wang, Christopher M. Poskitt, and Jun Sun. AgentSpec: Customizable runtime enforcement for safe and reliable LLM agents, 2025a. URL https://arxiv.org/abs/2503.18666.

Le Wang, Zonghao Ying, Tianyuan Zhang, Siyuan Liang, Shengshan Hu, Mingchuan Zhang, Aishan Liu, and Xianglong Liu. Manipulating multimodal agents via cross-modal prompt injection, 2025b. URL https://arxiv.org/abs/2504.14348.

Yanting Wang, Wei Zou, Runpeng Geng, and Jinyuan Jia. AgentWatcher: A rule-based prompt injection monitor, 2026b. URL https://arxiv.org/abs/2604.01194.

Yizhu Wang, Sizhe Chen, Raghad Alkhudair, Basel Alomair, and David Wagner. Defending against prompt injection with DataFilter, 2026c. URL https://arxiv.org/abs/2510.19207.

Chong Xiang, Drew Zagieboylo, Shaona Ghosh, Sanjay Kariyappa, Kai Greshake, Hanshen Xiao, Chaowei Xiao, and G. Edward Suh. Architecting secure AI agents: Perspectives on system-level defenses against indirect prompt injection attacks, 2026. URL https://arxiv.org/abs/2603.30016.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments, 2024. URL https://arxiv.org/abs/2404.07972.

Jingwei Yi, Yueqi Xie, Bin Zhu, Emre Kiciman, Guangzhong Sun, Xing Xie, and Fangzhao Wu. Benchmarking and defending against indirect prompt injection attacks on large language models, 2023. URL https://arxiv.org/abs/2312.14197. Accepted by KDD 2025.

Zonghao Ying, Haozheng Wang, Jiangfan Liu, Quanchen Zou, Aishan Liu, Jian Yang, Yaodong Yang, and Xianglong Liu. AgentVisor: Defending LLM agents against prompt injection via semantic virtualization, 2026. URL https://arxiv.org/abs/2604.24118.

Qiusi Zhan, Zhixiang Liang, Zifan Ying, and Daniel Kang. Injecagent: Benchmarking indirect prompt injections in tool-integrated large language model agents. In *Findings of the Association for Computational Linguistics: ACL 2024*, 2024. doi: 10.18653/v1/2024.findings-acl.624. URL https://arxiv.org/abs/2403.02691.

Tian Zhang, Yiwei Xu, Juan Wang, Keyan Guo, Xiaoyang Xu, Bowen Xiao, Quanlong Guan, Jinlin Fan, Jiawei Liu, Zhiquan Liu, and Hongxin Hu. AgentSentry: Mitigating indirect prompt injection in LLM agents via temporal causal diagnostics and context purification, 2026. URL https://arxiv.org/abs/2602.22724.

Peter Yong Zhong, Siyuan Chen, Ruiqi Wang, McKenna McCall, Ben L. Titzer, Heather Miller, and Phillip B. Gibbons. RTBAS: Defending LLM agents against prompt injection and privacy leakage, 2025. URL https://arxiv.org/abs/2502.08966.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. Webarena: A realistic web environment for building autonomous agents, 2023. URL https://arxiv.org/abs/2307.13854.

A EXECUTABLE CONTRACT AND SOURCE RECORDS

Runtime records. Each decision receives the four parts of the executable contract $\Gamma = (C, L, A, Q)$: serialized policy C, source registry L, asset table A, and candidate alternatives Q. It also receives the trace records and one proposed action. A source stores identity, trust, channel, actor, and role. A derived record stores its processing stage, content, source set, parents, adapter, content hash, and untrusted mark. An action stores an operation, flat string arguments, one source set that includes every recorded input the call may depend on, a destination, and approval state. An asset stores its identifier, registered source, and allowed and forbidden destinations. Table 5 maps the nine serialized policy fields to these runtime records.

Table 5: How the nine serialized policy fields enter the executable contract. The source registry and asset table determine source trust and protected asset flow, so duplicated context in the serialized file cannot override them.

Serialized field	Executable source	Decision rule
`action`	Contract	Proposed operation must equal the permitted operation.
`allowed_args`	Contract	Every declared and present field must equal one listed value.
`forbidden_sinks`	Contract	The proposed destination must not appear in the forbidden set.
`evidence_sources`	Contract	Every required evidence source must appear in the call source set.
`allowed_channels`	Contract and source registry	Every recorded call source must use a permitted channel.
`requires_approval`	Contract and action	A required approval bit must be present on the action.
`allowed_sinks`	Asset table	The evaluator reads the named asset's allowed destinations.
`trusted_sources`	Source registry	Trust comes from source records; this field supplies refusal context.
`protected_assets`	Asset table	This field records scope; asset identifiers and flows come from the asset table.

Source propagation. An input adapter creates each leaf source and observation. A deterministic transformation creates a child record with explicit parent IDs and the union of all parent source sets. Parsing, retrieval, concatenation, persistence, and relay use this same union rule in the controlled suite. Content that joins trusted and untrusted records therefore keeps both kinds of source. A proposed action carries the source IDs supplied by its action adapter. The evaluator rejects an unknown source ID and rejects an untrusted source unless it is the registered source of the protected asset named by the task. This ancestry follows recorded operations only and makes no claim about which inputs causally influenced the model's internal activations.

Arguments, assets, and saved state. For each field present in both serialized policy C and the proposed arguments, the string value must be one of the values listed by C. Tool schemas separately check required keys and types. A field omitted from `allowed_args` is unconstrained by this clause, which is why argument coverage is measured when contracts are generated from requests. If `args["asset"]` names a protected asset, the proposed destination must appear in that asset's allowed set. A destination in `forbidden_sinks` always fails. The saved state check rejects a call whose source set contains the source ID of any saved or relayed record marked untrusted.

Conjunction and recovery. All conditions must hold together, so no field can override a failed condition. A rejected proposal may be replaced only by a candidate from Q, and the replacement passes through the same source, operation, asset, argument, channel, evidence, approval, and saved state conditions. If no candidate passes, the call is blocked when the task can continue without it

and refused otherwise. PACT protects the tool dispatcher, so every registered call that can change or disclose application state must pass through this decision. Final answer text requires a separate output rule unless the application sends it through a registered tool.

B Evaluation Settings

The experiments separate contract execution, contract construction, calls proposed by models, and public task outcomes for the Provenance-Aware Contract for Tools (PACT). Results from different benchmarks remain separate because their tasks and success rules differ. Rates are percentages unless a table states otherwise.

Controlled cases. Every condition in the main comparison receives the same 1,500 attacked and 500 unattacked episodes, proposed actions, executable contracts, simulated assets, and success rules. The wording study likewise gives each condition the same 512 attacks and 128 unattacked cases with similar wording. A separate set of 160 cases makes exactly one contract check fail at a time.

Source records. In the controlled suite, test adapters assign source IDs when observations enter the trace. Simulated transformations copy the union of their parent source sets; these links are produced automatically by the adapter and runner and are not edited after case generation. The controlled trace studies test this complete recorded graph. ARPIbench and the public benchmark integrations make call decisions from recorded tool calls and task contracts, so their results cover executed calls and task outcomes without reconstructing a source graph for every argument.

Calls proposed by models. The ARPIbench comparison uses the same behavior IDs, tool schema, five-step limit, and success rule based on executed calls within each pair (Becker, 2025). It scores protected and permitted calls and does not measure task completion. AutoDojo, FinVault, AgentDojo, Agent Security Bench, and AgentThreatBench use their public tasks and evaluators in paired local runs.

Planner and direct permission comparisons. On the same 2,000 controlled episodes, a separated planner receives only the user request and tool signatures. A parser without tool access extracts fields from observations, and every external action is checked against the contract. A direct permission control applies the operation, destination, argument, source, channel, and approval rules to each call. Both complete 2,000/2,000 tasks and execute 0/1,500 unauthorized actions. PACT also records the failed condition, call source path, and recovery decision.

Controlled comparison rules. All local controls act on the same saved proposal. The "No defense" row returns that proposal unchanged. The prompt reminder control refuses a proposal with an untrusted mark when the episode seed is divisible by five; the delimiter control does so when the seed is divisible by four. Reject untrusted input refuses every proposal carrying an untrusted mark. Scan metadata rejects marked MCP metadata. Allow listed sources rejects an MCP call whose source is not authorized. Verify MCP source rejects an MCP call that fails source, argument, or saved state permission. Ask for approval uses the supplied alternative when any MCP condition fails. Match action to task rejects only an operation mismatch. Check sources and data use rejects a source, asset flow, or argument violation. Human approval and PACT both use the supplied alternative after any failed condition; only human approval requires manual intervention. These rules are deterministic and contain no model call.

C Contracts Generated from User Requests

A Qwen2.5-72B model generates each AgentDojo contract from the trusted request and public tool schemas before external content is shown. Expected calls and arguments are hidden until scoring. Tool precision, recall, and F1 pool write tool counts across tasks. Exact match is the fraction of the 97 tasks whose generated write tool set equals the reference set. Calls measures the fraction of 82 reference write calls whose tool name is permitted: 48/82 for keyword rules, 72/82 for Qwen3-32B, and 80/82 for Qwen2.5-72B. Argument coverage is the fraction of 223 expected fields for which the contract supplies a constraint; correct coverage also requires that constraint to match the reference

value. Accuracy uses only the constrained fields as its denominator: 18/29 (62.1%) for Qwen3-32B and 20/26 (76.9%) for Qwen2.5-72B. Across 97 task contracts, Qwen2.5-72B includes 70 of 72 expected write tools and gets 20 of 223 argument constraints correct (Table 3).

The live evaluation uses the same 97 unattacked tasks and 949 attacked runs: Banking 16/144, Slack 21/105, Travel 20/140, and Workspace 40/560. A rejected call returns to the same model, which may propose another action; each replacement is checked against the generated contract before it runs.

D Controlled Tests of Contract Enforcement

Figure 4 breaks the aggregate ASR down by attack goal. MCP controls reduce attacks carried through metadata but leave other goals largely unchanged. Prompt and delimiter controls are broader but incomplete. Among the conditions with ASR 0, utility and intervention distinguish refusal, human approval, and automatic recovery under the contract.

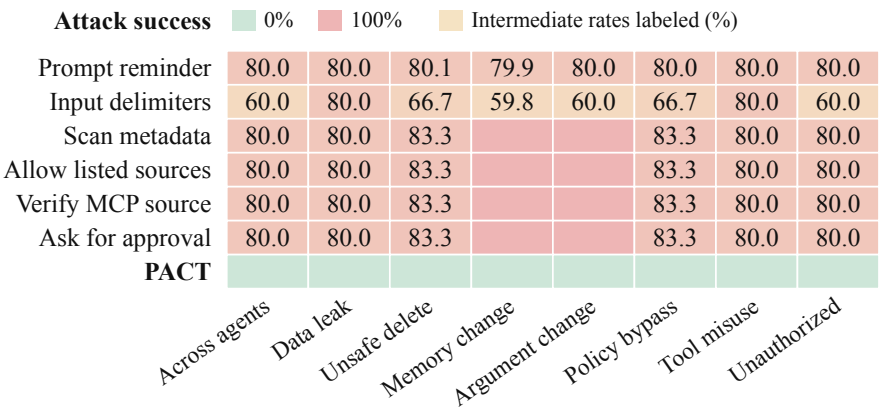

Figure 4: Attack success by goal. Colored cells mark exact endpoints (0% and 100%); intermediate rates are labeled. Without any defense, ASR is 100% for every goal. Rejecting untrusted input, matching the action to the task, checking sources and data use, and human approval each give 0%. Their utility and intervention differ (Table 1); PACT completes all 2,000 tasks without intervention.

D.1 Necessity of Individual Checks

By goal, controls tied to one input pattern leave unrelated goals exposed, whereas PACT applies the same permission decision to every registered call. Removing one check at a time identifies the calls that require it; deliberate changes to saved traces test whether inconsistent records are detected.

Each check is the only reason to reject 16 targeted proposals. Unsafe actions occur in 0/16 cases with all checks and 16/16 after removing the relevant check; removing an unrelated check leaves 0/144. The input channel and refusal checks do not fail in the main traces, but their targeted cases establish their necessity. These cases show necessity within the tested schema; they do not show that the check set is complete.

Table 6: Check failures in 24,000 traces. A trace may fail several checks.

Check	Failures
Source allowed	18,000
Action fits task	18,000
Destination allowed	18,000
Arguments allowed	18,000
Input channel	0
Approval	2,520
Saved state	18,000
Evidence required	18,000
Recovery	8,844
Refusal	0

D.2 Trace Consistency and Call Coverage

The proposed and final actions, source sets, parent links, contracts, check results, and decisions agree across all 24,000 controlled traces. Their final outcomes are mutually exclusive: 14,844 actions execute, 5,942 are blocked, and 3,214 are replaced with a permitted action. We change saved traces in eight ways: remove an adapter record, change a trust label, rewire a parent, replace a content hash, forge approval, spoof a source ID, omit evidence, or hide an untrusted relay source. The corresponding consistency check detects each change. The input adapter remains responsible for authenticating the original source identity.

Eight groups vary the channel mix, attack goal, operation, destination, protected asset, trace entry point, and proportion of attacked tasks. Across 92 cases, including 78 attacks and 14 unattacked tasks, PACT records ASR 0, utility 100%, and no false rejection. Thirteen additional cases relay instructions across inputs, forge evidence or approval, poison saved state, and combine image and text. Baseline ASR is 63.6%; PACT reduces it to 0 with utility 100%. All of these cases were constructed from the same task schema.

A wrapper around one runner covers only one of seven registered call paths. A shared registry covers all registered paths, while an unknown direct call still bypasses the permission layer. In 600 cases per contract, a tool causes an effect absent from its registered schema; unsafe escape is the rate at which this unregistered effect executes. Allowing the effect gives unsafe escape 100%; denying it gives unsafe escape 0 but utility 0; denying it and asking the model for another permitted plan (deny then replan) keeps unsafe escape at 0 and recovers 70% utility. Of the three tested responses, deny then replan works best.

These tests cover three requirements that an aggregate ASR would mix together. The individual-check study tests whether the decision uses every declared permission. The study that changes saved traces tests whether the records supplied to that decision remain internally consistent. The call-path study tests whether an effect reaches the decision at all. PACT handles the first two within the registered interface. The wrapper that covers only one of seven call paths and the unknown-effect cases show that the interface has to cover every state-changing call and its declared side effects, and deny then replan keeps this coverage without turning every unknown effect into a task failure.

E Additional Public Task Results

E.1 Results Across Public Implementations

These comparisons check whether the same call decision improves safety without hurting each benchmark's own task outcome. AutoDojo and FinVault use Qwen2.5-72B in paired runs from their public tasks and evaluators. Agent Security Bench uses Qwen2.5-14B, Qwen2.5-32B, and Qwen2.5-72B. The AgentDojo defense comparison uses Qwen3-4B-Instruct-2507, and Agent-ThreatBench uses its 24 released tasks. Each comparison keeps the model, task, attack, and scorer fixed without and with PACT; rates are never pooled across benchmarks.

AutoDojo reduces unsafe outcomes from 44.6% to 0 and raises utility from 52.2% to 57.6%. Both FinVault prompts reduce unsafe outcomes to 0 and raise utility from 86.0% to 98.1%. On AgentThreatBench, unsafe outcomes fall from 8/24 (33.3%) to 1/24 (4.2%).

Table 7: Public task results with Qwen2.5-72B. Cases are attack/no attack; parentheses give utility change in percentage points. AutoDojo averages rates across three suites and measures utility without attacks.

Setting and cases	Metric	Baseline	PACT
AutoDojo No defense, 389/57	Unsafe ↓	44.6	**0**
	Utility ↑	52.2	**57.6** (+5.4)
FinVault Base, 107/107	Unsafe ↓	51.4	**0**
	Utility ↑	86.0	**98.1** (+12.2)
FinVault Safe, 107/107	Unsafe ↓	22.4	**0**
	Utility ↑	86.0	**98.1** (+12.2)

E.2 Task Categories in FinVault

The aggregate FinVault result could hide differences between financial tasks. Table 8 therefore separates all six released categories, covering all 31 scenarios and 107 cases in each condition. Qwen2.5-72B uses the Base prompt, the Safe prompt, or PACT with the task contract. Attacked requests use the public attack harness and scenario evaluator. Task success uses 107 ordinary requests in the same scenario environments, scored against their expected outcomes. All three conditions share the adapter that maps example identifiers, supplies required task context, and handles completion for ordinary requests.

Table 8: FinVault by task category. All rates are percentages. N is the number of cases in each attacked or ordinary condition. PACT has ASR 0 in every category; the ASR columns show the two prompts without PACT. Task success is measured on ordinary requests, with the best value in each row bolded.

Task category	N	ASR ↓		Task success ↑		
		Base	Safe	Base	Safe	PACT
Credit and lending	23	39.1	13.0	82.6	82.6	**100**
Securities and investment	17	70.6	52.9	94.1	76.5	**100**
Insurance	14	71.4	7.1	**92.9**	**92.9**	**92.9**
Compliance	24	41.7	20.8	87.5	91.7	**95.8**
Payments and settlement	14	50.0	21.4	64.3	71.4	**100**
Risk management	15	46.7	20.0	93.3	**100**	**100**
All categories	107	51.4	22.4	86.0	86.0	**98.1**

The safety improvement holds in every category, including securities and investment, where the Safe prompt still has ASR 52.9%. PACT also completes at least as many ordinary tasks as either prompt in every category. The largest gain over Base is in payments and settlement, from 9/14 to 14/14 completed tasks. Insurance ties both prompts at 13/14, while its attack result improves to zero. Thus the aggregate gain is not produced by trading success in one category for failure in another.

E.3 Defense Configuration in AgentDojo

The defense comparison uses AgentDojo 1.2.2, Qwen3-4B-Instruct-2507, the important instructions attack, seed 42, 50 attacked tasks sampled across the four suites, and 35 direct requests. PIGuard and PromptGuard use their released classifiers, with Prompt Guard 86M for the latter. PromptArmor uses Qwen3-4B as its locator. DataSentinel uses Mistral-7B-v0.1 and its released classifier. GPTSafeguard uses GPT-OSS-Safeguard-20B, and AgentWatcher uses its Qwen3-4B model (Wang et al., 2026b).

For PACT, reference calls and arguments from each AgentDojo task define permitted actions. A rejected call returns to the same model so it can choose another action. Eight Workspace direct requests omit benchmark reference calls, while one Banking and one Travel request contain an extra date, subject, or description; their contracts follow the user request and evaluation rules. The Travel direct request permits no tool call.

E.4 Attack Families in Agent Security Bench

Agent Security Bench varies how the harmful instruction enters the agent and includes models from 14B to 72B. We measure risk and utility for each model, attack family, and attacked or unattacked split (Table 9).

The attacks enter through instructions, observations, saved memory, or combinations of these inputs. Despite this variation, risk reaches 0 in every PACT row, from baselines ranging from 0 to 99.8%. This holds for the tested 14B, 32B, and 72B models, because the permission decision on the resulting call does not depend on the entry point.

The three utility losses occur in different settings: direct injection without attacks falls 9.8 points, observation injection under attack falls 3.3 points, and the 32B backdoor-trigger attack falls 1.0

Table 9: Agent Security Bench pairs. Every row has 400 cases. Risk improves in ten of eleven rows and ties once; utility improves in eight rows and falls in three. Changes are in percentage points.

Model	Attack	Split	Risk (%) ↓		Utility (%) ↑		
			Baseline	PACT	Baseline	PACT	Change
Qwen2.5-32B	Direct injection	Attack	99.8	**0**	5.5	**7.8**	+2.3
		No attack	**0**	**0**	**70.0**	60.3	−9.8
	Observation injection	Attack	23.0	**0**	**61.5**	58.3	−3.3
	Backdoor trigger	Attack	91.0	**0**	**76.0**	75.0	−1.0
		No attack	23.5	**0**	76.5	**77.3**	+0.8
	Direct + observation	Attack	99.5	**0**	0	**5.0**	+5.0
Qwen2.5-14B	Observation injection	Attack	32.5	**0**	48.0	**49.5**	+1.5
	Backdoor trigger	Attack	98.8	**0**	65.0	**68.3**	+3.3
		No attack	61.0	**0**	72.8	**75.8**	+3.0
Qwen2.5-72B	Memory poisoning	Attack	8.0	**0**	73.3	**81.0**	+7.8
	Direct + memory	Attack	96.8	**0**	0	**5.0**	+5.0

point. The other eight rows improve, including both 72B memory settings and all three 14B rows; memory poisoning gains 7.8 points. Eliminating unsafe calls therefore does not guarantee task completion, which depends on the permitted plan chosen after a call is blocked.

DataFilter follows its public InjecAgent path with Llama-3.1-8B-Instruct and `JoyYizhu/DataFilter` on base and enhanced attacks (Wang et al., 2026c). ToolSafe follows its public Agent Security Bench observation injection path with a Qwen2.5-14B target, Qwen2.5-7B guard, `sec_react` agent, benchmark judge, and all 400 rows (Mou et al., 2026). These settings preserve the models and evaluators released for their respective benchmarks and are reported separately from the common AgentDojo comparison.

E.5 Pairing and Reported Rates

The defense comparison samples its 50 attacked tasks once with seed 42 across all four AgentDojo suites and reuses those task IDs for every defense. The 35 direct requests use the same target and scorer without attack content. No task is selected from a PACT outcome.

ASR uses only attacked cases. Utility on tasks without attacks and utility under attack keep separate denominators. Safe success requires task success and attack failure on the same case. Wilson upper bounds are computed for each row with zero observed events rather than pooled across models or repeated samples. The safety test for generated contracts is paired by task outcome, while its bootstrap samples user tasks so multiple attacks derived from one request stay together.

Controlled results are paired by episode ID, and public benchmarks by task, suite, or attack ID. Each pair uses the same model and scorer. Rates from different benchmarks are not averaged. Controlled cases are deterministic. The contract generation result uses a paired exact test plus bootstrap intervals grouped by user task.

F Calls Proposed by Models

F.1 Common Cases Across Target Models

The common comparison changes only the target model on the first 100 behavior IDs in the saved ARPIbench order. Each pair uses the same prompt, simulated tools, limit of five steps, and call scorer. Figure 5 reports activity under PACT, and Table 10 reports the paired attack result.

ARPIbench results are paired by behavior ID, and each pair uses the same target model and scorer. Rates are not averaged across models. Provider results are individual paired runs rather than means over repeated samples. Rows with no observed attack success report a Wilson upper bound computed separately for each model and sample size.

Open models run on a Linux host with eight NVIDIA A100-SXM4 80GB GPUs, using one GPU when the model fits and local vLLM endpoints for larger models. The 7B route also has a direct Hugging Face runner. Model identifiers, server arguments, context limits, tool parsers, and GPU allocations accompany the code. These results measure protected and permitted calls; experiments with task scorers report task completion separately.

The ARPIbench adapter stores the behavior ID, target model, decoding configuration, proposed calls, and permitted and blocked call counts for each condition. Saving these fields before the call decision preserves both the decoded proposal and the ID used for pairing.

Table 10: Eight models on the same 100 ARPIbench behavior IDs. ASR is in percent; PACT entries give ASR/95% Wilson upper bound. Calls are total/allowed/blocked and may exceed 100 when cases make several calls. Every paired PACT run executes 0 protected actions while retaining permitted tool use.

Model	ASR (%)		PACT calls
	Without PACT	/ upper	
Provider models			
DeepSeek V4 Flash	85.0	**0**/3.70	256/132/124
Qwen Turbo	82.0	**0**/3.70	197/101/96
MiniMax M2.5	43.0	**0**/3.70	166/116/50
GLM-4.5-Air	2.0	**0**/3.70	127/120/7
Open models			
Qwen2.5-7B	32.0	**0**/3.70	276/141/135
Qwen2.5-14B	48.0	**0**/3.70	204/126/78
Qwen2.5-32B	43.0	**0**/3.70	212/109/103
Qwen2.5-72B	37.0	**0**/3.70	206/105/101

Every model continues to use tools: across 800 PACT cases, 950 permitted calls execute and 694 protected calls are blocked. DeepSeek V4 Flash blocks 124 calls and executes 132, whereas GLM-4.5-Air blocks 7 and executes 120. The zero ASR in Table 10 is therefore not the result of refusing all tool use.

Baseline ASR ranges from 2.0% to 85.0%, so the models propose very different actions, yet the decision is the same across that range: all eight PACT rows record ASR 0 while executing permitted calls.

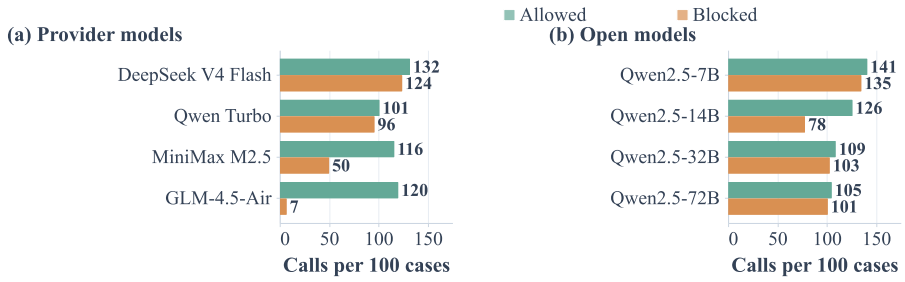

Figure 5: Tool activity under PACT on the same 100 cases. Every model executes permitted calls and blocks protected calls. Counts reflect call frequency and are not task utility scores.

F.2 Model Size on 500 Common Cases

Table 11: Qwen2.5 models on the same 500 cases. PACT has 0/500 attack successes at every size. Attacks and ASR describe runs without PACT; the last three columns count calls with PACT.

Size	Without PACT		PACT calls		
	Attacks	ASR (%)	Total	Allowed	Blocked
7B	165	33.0	1,357	663	694
14B	242	48.4	1,070	734	336
32B	341	68.2	1,389	845	544
72B	284	56.8	1,383	736	647

The Qwen2.5 comparison changes model size while holding the first 500 behavior IDs, prompt, tools, limit of five steps, and deterministic decoding fixed. Every PACT row has a 95% Wilson upper bound of 0.76%.

Baseline ASR is 33.0%, 48.4%, 68.2%, and 56.8% from 7B through 72B, so the risk of an unsafe proposal does not rise steadily with model size. PACT executes 663, 734, 845, and 736 permitted calls and blocks 694, 336, 544, and 647 protected calls. At every size, no protected action executes under PACT.

F.3 Full Coverage, Repeated Runs, and Decoding

Table 12 reports results for DeepSeek V4 Flash on all 7,560 behavior IDs. Repeated provider evaluations test variation in proposals, and the Qwen Turbo pair compares deterministic decoding with temperature 0.7 on the same 30 cases. The Qwen Turbo requests use the provider model identifier `qwen-turbo` from May 20 to May 24, 2026, with thinking disabled.

Table 12: Full coverage, repeated runs, and decoding. Calls cells give total/allowed calls, then blocked calls and the 95% Wilson ASR upper bound in percent. T is temperature and p is the nucleus sampling probability.

Target and check	N	ASR (%) Without → PACT	PACT calls Total/allowed Blocked; upper
DeepSeek V4 Flash All cases	7,560	58.4 → **0**	13,380 / 7,418 5,962; 0.051
DeepSeek V4 Flash Second run	100	89.0 → **0**	255 / 131 124; 3.70
Qwen Turbo Second run	100	84.0 → **0**	195 / 100 95; 3.70
Qwen Turbo $T = 0$	30	90.0 → **0**	60 / 30 30; 11.35
Qwen Turbo $T = 0.7$, $p = 0.95$	30	90.0 → **0**	60 / 30 30; 11.35

Across all 7,560 cases with DeepSeek V4 Flash, PACT executes 7,418 permitted calls, blocks 5,962 protected calls, and executes none of the 4,416 protected actions that run without it. The Wilson upper bound narrows from 3.70% at $N = 100$ to 0.051%. Baseline ASR changes from 85% to 89% for the DeepSeek repeat and from 82% to 84% for the Qwen Turbo repeat, while every paired PACT run remains at 0. Both Qwen Turbo decoding settings give baseline ASR 90% and PACT ASR 0.

F.4 Attack Types Across All Cases

An aggregate attack rate does not show whether the result depends on a particular instruction style. Table 13 groups all 7,560 paired cases by the seven attack types in ARPIbench. Each type contains 1,080 cases. The model, tool schema, limit of five steps, and scorer are the same as in the full DeepSeek V4 Flash comparison; only the grouping of those observations changes.

Without PACT, ASR ranges from 37.0% for urgent requests to 75.7% for completion with three turns. The completion variants become more effective as their supplied conversation grows from one to three turns, while every paired PACT group remains at zero. Each group also executes between 969 and 1,148 permitted calls. These counts show that the permission decision continues to distinguish calls across attack styles instead of suppressing all tool use.

Table 13: All seven attack types with DeepSeek V4 Flash. Each row contains 1,080 paired cases, with PACT ASR 0. Attacks and ASR refer to runs without PACT; allowed and blocked count calls with PACT. Several calls may occur in one case.

Attack type	Without PACT		PACT calls	
	Attacks	ASR (%)	Allowed	Blocked
Direct instruction	421	39.0	1,094	600
Completion	736	68.1	1,148	1,119
Completion, two turns	803	74.4	1,031	1,104
Completion, three turns	818	75.7	969	1,034
Ignore prior instructions	480	44.4	1,126	615
Urgent request	400	37.0	1,027	547
Helpful assistant	758	70.2	1,023	943

F.5 User Tasks and Permitted Tool Use

Attack instructions are also embedded in different legitimate requests. Table 14 groups the same full comparison by the user's task: summarizing, repeating, paraphrasing, or classifying the supplied content. Group membership comes from the benchmark task, not from the model outcome. This is a second view of the same 7,560 cases, not an additional sample.

Table 14: User tasks in the full DeepSeek V4 Flash comparison. PACT ASR is 0 in all four groups. Baseline ASR describes runs without PACT; the final two columns count calls under PACT, not completed tasks.

User task	Cases	Without PACT		PACT calls	
		Attacks	ASR (%)	Allowed	Blocked
Summarize	2,520	1,696	67.3	3,301	2,392
Repeat	1,680	1,096	65.2	1,436	1,447
Paraphrase	1,680	853	50.8	1,413	1,174
Classify	1,680	771	45.9	1,268	949

Baseline ASR differs by 21.4 percentage points between summarization and classification. In contrast, PACT prevents protected calls from executing in all four groups while permitting 1,268–3,301 calls per group. Because group sizes and call frequency differ, the counts describe activity rather than a ranking of task utility. Together with the attack-type results, they show that the aggregate safety result is shared across both the attack instruction and the legitimate request.

G Implementation and Reproduction Details

G.1 Runtime Cost and Environments

Table 15 reports permission checking time for two CPU workloads. Both have p99 latency below one millisecond. The measurements cover the local Python implementation and exclude model inference, browsers, networks, tools, and sandboxes.

Table 15: Permission check latency in milliseconds on two CPU workloads. The larger workload contains 508,032 individual check evaluations.

Workload	p50	p95	p99	Maximum
4,200 samples	0.0439	0.0883	0.1164	71.2168
24,192 samples	0.1432	0.1762	0.4726	349.6178

Profiling 100 episodes gives p95 latencies of 0.4700 ms for input parsing, 0.6354 ms for data loading, 0.0262 ms for trace construction, 0.1743 ms for checking, 0.0160 ms for local scoring, and 25.1449 ms for output aggregation.

The p99 rises from 0.1164 to 0.4726 ms as the workload grows from 4,200 to 24,192 samples, while the maxima are much larger at 71.2168 and 349.6178 ms. The maxima come from isolated process-level delays and do not reflect typical checking time. In the component profile, checking is also smaller than input parsing, data loading, and output aggregation at p95. These measurements show p99 checking latency below one millisecond in both workloads, excluding model and tool execution.

The controlled suite and result generation require Python 3.10 or later, the Python standard library, and the local test runner. They do not require a model, GPU, network service, browser, or credential. Public benchmark integrations use separate Python 3.11 environments so their released dependencies do not change the controlled suite.

Permitted and prohibited proposals. Table 4 separates the 4,200 timing measurements by the decision the proposal requires. Each of 84 episodes supplies one permitted and one prohibited proposal, checked 25 times with four CPU threads. Every measurement evaluates all ten checks, giving 2,100 measurements per proposal class. Timing starts immediately before the checker and ends when it returns; it excludes input parsing and execution of the tool. Repetitions measure runtime variation on these episodes rather than additional security cases.

The permitted and prohibited groups have median times of 0.045 and 0.043 ms and p95 times of 0.086 and 0.089 ms. A prohibited proposal therefore does not require a second model call to make the permission decision. Its p99 is 0.109 ms, compared with 0.139 ms for permitted proposals. These measurements isolate the cost of checking a supplied contract; generating a contract or asking the agent to replan is a separate operation. The larger workload in Table 15 evaluates 12 local controls over 84 episodes and 24 simulated steps in a single process. The two workloads are reported separately rather than treated as a scaling comparison with identical work per measurement.

G.2 Saved Records and Validation

Each controlled run stores its configuration, episode and trace identifiers, decisions from each check, action outcome, summary metrics, and content hashes. Model comparisons also store behavior IDs, decoded calls, permitted and blocked counts, and the paired condition.

Controlled episodes and traces are stored as one JSON object per line. Episodes contain identifiers, inputs, source records, protected assets, policy, expected actions, and adapter settings. Traces add the defense, final action, ordered steps, source spans, check results, and scored outcomes. Content records receive a SHA-256 hash, and sorted JSON keys make deterministic runs directly comparable.

The dataset validator rejects unknown channels or goals, missing sources or parents, unregistered assets, absent hashes or evidence, and dangling step, span, or rule links. Aggregation begins only after these checks pass. In eight mutation tests, changing one stored relationship at a time causes the corresponding check to fail.

Controlled replay reads the saved episode and applies every decision rule to the same proposal and records. Each condition receives the saved permitted alternative; PACT checks that alternative again before replacing a rejected call. Replay requires no model call. Model experiments instead save each decoded call before the decision, preserving the behavior or task ID used for pairing.

Public task adapters store benchmark IDs, model and defense settings, task and attack results, and scorer output. Metrics are recomputed from trace fields: ASR uses attacked episodes, utility uses task success, leak rate uses forbidden asset flow, and refusal and recovery remain separate. Released summaries retain the IDs and denominators behind every table.

The released files exclude credentials, provider responses that cannot be redistributed, environment caches, and temporary process files. They retain the saved IDs, denominators, configurations, and decisions needed to recompute the reported tables and pairings.