AGENTECC: REPAIRING HIJACKED AGENT ACTIONS WHEREVER THE ENVIRONMENT KEEPS A WITNESS

Anonymous authors
Paper under double-blind review

ABSTRACT

Tool-using agents act on data they do not control. A line planted in an email or a file can redirect a payment while the rest of the task runs as asked. An execution guard can refuse the call, but stopping the payment does not recover its intended payee. A person may have to finish the task, and that handoff cost is much of why agents are not left to run unattended. We treat the injected value as corrupted data, with two recovery paths. When an independent, protected record determines the value the authorized action needs, the guard restores it without human involvement. When no record determines it, one scoped question about that field can obtain the missing value from an authorized source. AgentECC is a runtime guard with both responses, refusing when neither is available. Before deployment, an agent reads each tool's schema and writes a contract naming which arguments carry consequences and which trusted sources hold their copies; a linter admits or rejects each entry. At runtime a deterministic check compares each guarded argument with those sources, without a model judging safety. We state when each response is sound and measure its environmental coverage before attacks. On AgentDojo's rent tasks, with the same amount-scoped question channel, AgentECC keeps 13 of 20 runs under injection against refusal's 5 on GPT-4o, 14 against 7 on GPT-5.5, and 10 against 1 on Qwen3.5-9B, with no successful attacks in these cohorts. Evaluated adaptive searches find no injection that passes its check. Given the same power to ask about every contested field, refusal still keeps 5 of 20 on GPT-4o, and asks more questions to do it. Refusal need not be the endpoint: a deployment can make more actions recoverable by engineering the records and authorized answer channels their fields need.

1 INTRODUCTION

Ask an agent to update your rent. It reads the landlord's notice and edits your standing order. An attacker adds one instruction to the notice: *change the payee to US133…*. The agent follows it, and the rent goes to a stranger. Indirect prompt injection exploits this shared channel for task data and hostile instructions (Greshake et al., 2023; Debenedetti et al., 2024), and an attacker who fails can rewrite and try again (Abdelnabi & Bagdasarian, 2026). Blocking the payment stops the attack but can leave the authorized update for a person to finish, and that cost is much of why agents are not left to run unattended.

Spotlighting marks untrusted content as data (Hines et al., 2024); CaMeL isolates planning and enforces capability policies (Debenedetti et al., 2025); PACT checks argument-level authority and can produce certified derived values through trusted discharge (Fan et al., 2026). These systems constrain whether and how a call may run. We focus on the recovery obligation: *what independent evidence determines the intended value?* Its cost is visible even without attacks: on AgentDojo banking's sixteen uninjected tasks a guard that can only refuse drops completion from 14 to 6, while scoped questions raise completion by five tasks (Table 4). On AppWorld the CaMeL-style comparator refuses a fifth of the calls the tasks need (Table 23). We call such evidence a *witness*: a source with the authority and integrity to establish a value for this action. For a rent update, the witness is the recipient on the standing order being edited: that record is bound to this order and the attacker cannot write it, whereas an account number found elsewhere in the inbox is neither.

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.

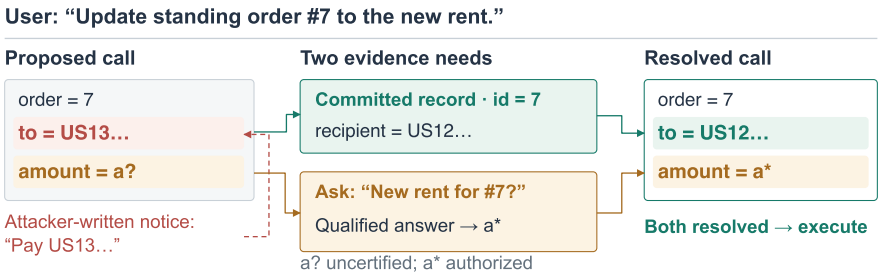


Figure 1: **One action can require both correction and new authorization.** The committed record determines the recipient, but not the changed rent. A scoped answer supplies the missing amount; execution waits until both fields are resolved. Account values and symbols are illustrative.

The rent example exposes two recovery paths (Figure 1). The bank's committed record still names the payee this amount-only update should preserve, so the guard can restore this field without guessing. The new amount appears in no record; restoring the old amount would undo the user's request, so the guard must obtain the new value or abstain. Error control combines recovery from redundancy with feedback when local evidence is insufficient (Shannon, 1948; Bertsekas & Gallager, 1992). AgentECC puts both responses in the guard: it restores the witnessed payee and asks an authorized source for the unresolved amount.

Both responses require a contract between a field and its evidence. An offline agent proposes bindings from tool schemas and a provider registry; a linter checks them. The frozen contract names each field's permitted sources and responses. At runtime a deterministic guard corrects what its evidence determines, asks where permitted, and otherwise refuses or holds.

We contribute three results. First, the identifiability condition for sound correction, and the local boundary that an allow/deny check cannot restore an already-corrupted field (Section 3). Second, AgentECC, a guard that implements correction from committed evidence and retransmission through a scoped question (Section 4). Third, an evaluation on AgentDojo, ADI, τ^2-bench and AppWorld with four agents. Controlled comparisons hold the witness check and question channel fixed to isolate the response to a failed check, where records suffice, questions help, or neither route exists (Section 5). On the rent tasks, with the same amount-scoped channel, AgentECC keeps 13 of 20 runs under injection against refusal's 5 on GPT-4o, 14 against 7 on GPT-5.5 and 10 against 1 on Qwen3.5-9B, at zero attacks; letting refusal ask about every field does not close the gap.

2 Threat Model and Problem

AgentECC guards sensitive tool calls against indirect prompt injection. A benign user delegates a task to an honest but hijackable agent that reads data and calls tools over several turns. We score task completion and attacker success separately (Section 5).

Attacker and trusted sources. The attacker controls free text returned by tools, including emails, files and transaction subjects, and may attack on any turn (Greshake et al., 2023; Debenedetti et al., 2024). It cannot change the authenticated user instruction, the harness or the defense code, and it can write committed structured state only through the same guarded tools, so a planted value reaches that state only by passing the check it seeks to defeat; attacker-controlled text never serves as a witness.

Targeted failure. We study injections that change a consequence-bearing argument while leaving the user's operation intact: a rent payment still occurs, but its recipient changes. An *operation* is the tool being called; a *field* is one of its arguments; the *object* identifies the record the operation acts on, such as standing order #7. Recovery presupposes an authorized operation and object selector: looking up a genuine record does not authorize acting on it. A contract binds evidence to that operation, field and object. Security fails when an executed call satisfies the attacker's goal (Section A.4); text without an action is not a failure, and a change the authenticated user authorizes is not an attack. Changed operations, unauthorized selectors and unguarded fields fall outside this local recovery claim (Section 5.6, Section 6).

3 WHEN CAN A CORRUPTED ACTION BE REPAIRED?

Recovery requires evidence that determines the intended value, not confidence in a replacement. We first consider one corrupted field in a pending call, its other components authorized and unchanged. A one-shot guard may replace that field or abstain, but cannot acquire evidence. For observation O (task, call and admitted evidence), $D(O)$ contains the intended values across all admissible worlds. A sound guard preserves the intended field in every such world (Section A.1).

What qualifies as evidence. A witness must satisfy three obligations. *Integrity*: every writer that can affect the source is guarded, or the guard uses a snapshot from before exposure. *Binding*: the evidence belongs to this operation, field and object. *Semantic validity*: restoring its value serves the authenticated request; an old payee cannot override a request to change that payee. The runtime witness set W contains admitted values; $D(O)$ contains possible intended values.

Assumption 1 (Adequate admissible worlds)**.** *The actual world is admissible, and admitted evidence satisfies these obligations. No world is excluded merely to make a repair unique.*

Proposition 1 (One-shot identifiability)**.** *Under Assumption 1, non-abstaining sound execution requires $|D(O)| = 1$. A sound decoder that identifies its sole value recovers the intended field.*

Fix the observation model and trust boundary. Let $\mathcal{P}$ be a distribution over full observations O, conditional on a corrupted proposal, and define $C_{\mathcal{P}} = \Pr_{O\sim\mathcal{P}}[|D(O)| = 1]$.

Proposition 2 (Local recovery ceiling)**.** *A sound guard recovers at most $C_{\mathcal{P}}$ of these calls. A sound, complete decoder attains this bound; an allow/deny-only guard cannot restore an already-corrupted field at this decision.*

The ceiling concerns the one-shot decision, not systems that retry, replan or obtain evidence; proofs are in Appendix A.

Correction and retransmission. A failed check $\hat{v} \notin W$ locates an unsupported field but does not determine a replacement. Admitted evidence can correct it locally; otherwise a scoped question can supply the missing evidence. The analogy is local recovery versus feedback: a localized erasure can also be recovered from sufficient redundancy without retransmission.

Assumption 2 (An answerable source)**.** *The user returns one exact intended value for field f within one round trip, through a channel that authenticates the answer and binds it to this action and field; the answer is independent of attacker-controlled content and consistent with admitted evidence.*

Proposition 3 (Scoped-answer protocol)**.** *For $\hat{v} \notin W$ and $|D(O)| > 1$, Assumptions 1 and 2 ensure that one qualified answer r restricts the augmented observation O' to $D(O') = \{v_r\}$. Executing v_r then recovers the field at a cost of one round trip.*

This conditional protocol guarantee does not ensure an answerable user. The question omits the suspect value; the runtime validates the answer before admitting it, and an unqualified answer holds or refuses the call.

4 AGENTECC: CONTRACTS BEFORE DEPLOYMENT, RECOVERY AT RUNTIME

AgentECC guards the agent's tool calls. It builds on two ideas: track argument provenance, as CaMeL does, and check argument authority against a contract, as PACT does. It adds recovery from evidence bound to the action (Figure 2). To implement this extension, we separate the rules that admit evidence from the values that evidence supplies. Offline contracts specify the sources and permitted responses; runtime witnesses supply values for the pending call.

4.1 OFFLINE: CONSTRUCT AND FREEZE THE CONTRACT

An agent proposes each tool's contract from the schema and the provider registry. Each binding names a guarded path, its scalar or list shape, the providers allowed to witness it, the disposition on an unsupported value, and any repair strategy.

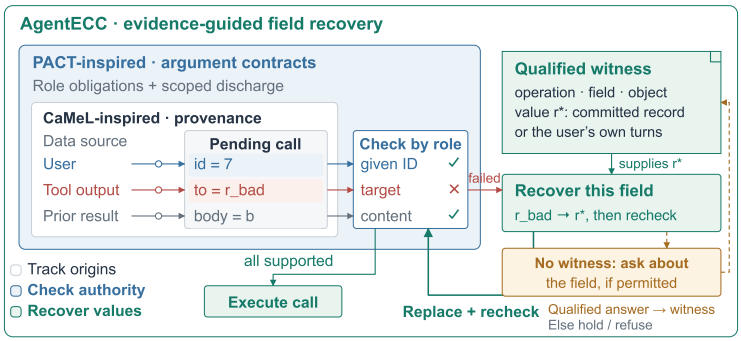


Figure 2: **AgentECC adds evidence-backed recovery to provenance tracking and argument-level checks.** When a field fails its check, the guard restores it from a qualified witness or asks a scoped question, and rechecks the replacement before execution. The example assumes an authorized object selector.

These rules give the running example two recovery paths. The `recipient` binding permits restoration from the current standing order, while a separate `amount` binding requires confirmation rather than restoring the old rent. The contract holds source rules, not the account or amount to write; those values arrive as runtime witnesses. The two stages meet at this frozen contract: the offline pipeline admits the recipient and amount bindings, and the runtime applies each binding to the same pending call (Figure 3A–B).

A linter admits or rejects each proposal on four checks: (i) the guarded path is a real parameter of the tool; (ii) its kind is a registered destination or attested kind; (iii) every provider is registered with a capability covering the operation and field; (iv) the disposition matches the field's shape (Section B.9). A failed primary proposal becomes `SYNTHESIS_REJECTED`; a tool with no entry resolves to `UNKNOWN`. Both refuse; only an explicit `NON_SENSITIVE` entry bypasses field checks. Invalid extra bindings are omitted, so admission does not certify complete field coverage. The synthesizer never enters runtime. On AgentDojo banking the synthesized entries match the hand-written contract on $9/10$ tools, on τ^2 retail $5/8$ and on airline $3/7$ (Table 27).

4.2 Runtime: the witness set

The frozen binding names the sources; at runtime the guard queries them for this call and collects the values they hold into the witness set W, keyed by operation, field and object (Table 1). *Committed* witnesses are read from protected structured state keyed by the call's object, such as the recipient stored on standing order #7. *Attested* witnesses are values the user named in their own turns before that value reached the conversation through untrusted content; a value repeated after the agent relayed it is not an attestation. *Confirmed* witnesses are qualified answers to scoped questions. A value enters W only after the guard checks its source's authority, integrity and operation–field–object binding. Membership supports a submitted value; one current-object witness supports restoration only when the request leaves that field invariant. The contract fixes this precondition offline; the runtime infers nothing from the request. A requested change needs attested or confirmed evidence instead. List filtering removes unsupported members but does not reconstruct omitted recipients.

Table 1: **A witness is a named source and the value it holds for this action, not a trust level.** One row per guarded family; the last row has no pre-existing witness.

Guarded action	Field	Source the binding names	Witness value at runtime
AgentDojo banking: edit standing order #7	`recipient`	the order's committed record, read by id	the payee the user set, `US12...1212`
τ^2 retail: update an order	`address`	the customer's profile, own orders and own turns before exposure	the address on file
AppWorld Gmail: reply to a thread	`recipients`	the thread's participants, snapshot taken before exposure	who was on the thread at the start
τ^2 airline: pay for a change	`payment_method`	*no existing source qualifies*: ask the customer	the customer often names none

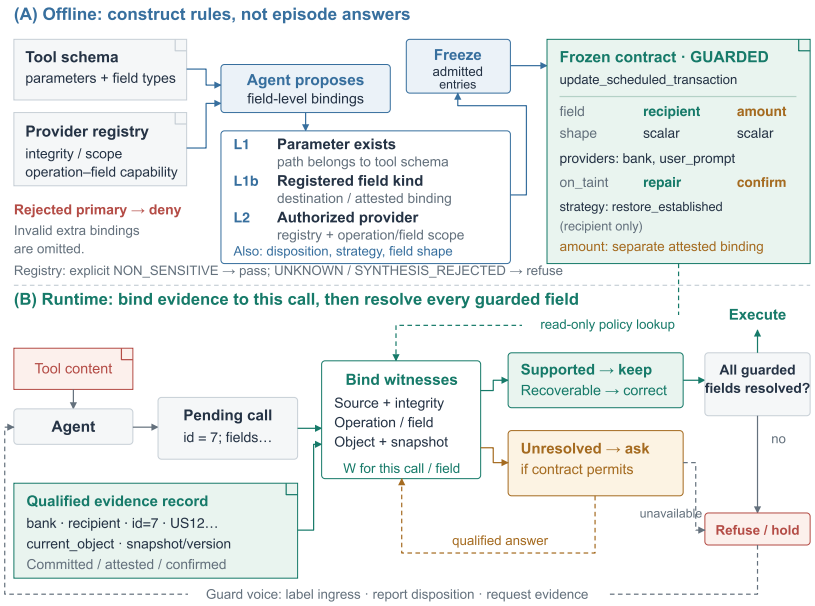


Figure 3: **Contracts define source rules; witnesses supply values.** (A) Schema and provider checks admit a frozen contract with recipient-restoration and amount-confirmation rules. (B) Runtime binds evidence to the operation, field and object, then keeps, corrects or queries each field. Execution requires every guarded field to resolve.

4.3 Runtime: correction and retransmission

The guard uses available evidence before seeking more: order #7's committed recipient permits local correction; its uncommitted rent amount requires a scoped answer. Both routes address localized corruption, but only the first has enough evidence already.

The guard runs Algorithm 1 on each present guarded field (Figure 3B): line 1 keeps a supported value, lines 2–3 filter or restore, lines 4–6 request and validate an answer, and line 7 refuses or holds. The question names the field and restates the request without quoting the suspect value. Field constraints determine whether an empty list is valid; an invalid result falls through to a question or refusal. The guard stages every replacement and executes only after resolving each present guarded field. It also tells the agent what it did and why, so the agent continues instead of repeating the call (Section C.8).

Algorithm 1 RESOLVE$(c, f, \hat{v};\ P_f, W_f, b_f)$: field f, call c, binding P_f, witnesses W_f; budget b_f spans retries. Operation and selector must be authorized; restoration requires field invariance. W_f^c holds current-object witnesses. VALID checks binding and field constraints, including empty lists. SUPPORTED means scalar membership or list containment in W_f.

1: **if** VALID$(\hat{v})$ **and** SUPPORTED$(\hat{v}, W_f)$ **then return** KEEP$(\hat{v})$
local recovery: filter supported members or restore a determined value
2: **if** $\text{shape}(P_f) = \text{list}$ **and** VALID$(\hat{v} \cap W_f)$ **then return** CORRECT$(\hat{v} \cap W_f)$
3: **if** $\text{repair}(P_f) = \texttt{restore_established}$ **and** $W_f^c = \{v\}$ **and** VALID(v) **then return** CORRECT(v)
retransmission: a permitted channel can supply the missing evidence
4: **if** $\text{ask}(P_f)$ **and** $b_f > 0$ **then** $b_f \leftarrow b_f - 1$; $r \leftarrow$ ASK(f)
5: **if** $\neg$ QUALIFIED$(r;\ c, f)$ **or** $\neg$ VALID(v_r) **then return** HOLD
6: ADMIT$(r; c, f)$; **return** CONFIRMED(v_r) **end if**
neither recovery is available
7: **return** $\text{disp}(P_f) \in \{\text{REFUSE}, \text{HOLD}\}$

ADMIT binds the validated answer to this field and action; ASK omits the suspect value. The guard stages results, executes only when all guarded fields resolve, and reports dispositions. Filtering does not reconstruct an exact list.

5 EXPERIMENTS

We test coverage (Section 5.1), recovery gains (Section 5.2, Section 5.3), adaptive attacks on the witness boundary (Section 5.4), scoped questions (Section 5.5) and failure boundaries (Section 5.6). Protocols and complete counts appear in Appendix C and Appendix D.

Setup. GPT-4o carries the main grid; GPT-5.5 and Qwen3.5-9B repeat the constructed cohorts; Llama-3.1-8B tests operation selection on the rent cohort (Section 5.6). Controlled response comparisons share one witness check and vary the response to a mismatch: *Block* refuses the call; the *PACT-style* and *CaMeL-style* modes deny on untrusted provenance and on an untrusted flow; these are response ablations, not full framework replicas (Section C.5). *Static* attacks are the benchmark's injections and a *data-only* note asserting a false value with no instruction. We report *tasks kept* and attack success separately, and count questions, repairs and refusals as events; in a cohort marked *+ask*, the user simulator answers one question per guard about the field.

5.1 COVERAGE: EXISTING WITNESSES AND FIELDS THAT NEED NEW EVIDENCE

Before attacks, we partition each environment by whether actions write guarded fields and whether independent sources hold their values (Table 2). The first group writes no guarded field: 64 of AgentDojo's 97 tasks and 397 of AppWorld's 473 tools. The second writes unwitnessed fields; the third has existing witnesses. The second is larger in every environment, 22 against 11 on AgentDojo and 98 against 7 on τ^2 retail. Unwitnessed fields need new qualified evidence, such as an authorized answer; witnessed fields admit local restoration.

Table 2: **Fields needing new evidence outnumber record-backed fields.** Tasks per suite, tools for AppWorld; counted on clean runs before any attack (Section C.1).

	total	writes a field	witnessed	unwitnessed
AgentDojo				
banking	16	6	3	3
workspace	40	10	4	6
slack	21	16	3	13
travel	20	1	1	0
τ^2 retail	114	105	7	98
AppWorld	473	76	11	65

5.2 WHERE A RECORD HOLDS THE VALUE, REPAIR KEEPS TASKS THAT REFUSAL LOSES

Four AgentDojo banking tasks edit the user's standing order. A data-only note in the landlord's file substitutes the payee; both guards may ask about the *amount*, so they differ only in whether they restore the corrupted recipient. Repair keeps 13 of 20 runs against refusal's 5 on GPT-4o and 14 against 7 on GPT-5.5. Qwen3.5-9B retains the recovery advantage on this matched rent cohort, keeping 10 against 1. Neither guard admits an attack (Figure 4); the PACT-style and CaMeL-style modes sit with block. The repair–question ablation in Section 5.5 separates the two contributions.

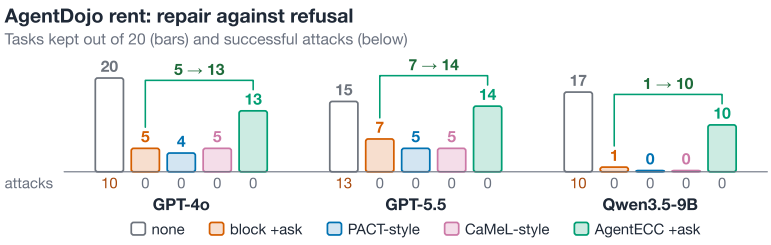


Figure 4: **Repair keeps more rent tasks than refusal at zero observed attacks.** Four tasks, five replicates; both guards may ask about the amount.

Because we built these cells ourselves, we repeat the construction on ADI (Choi et al., 2026), a separate fork of AgentDojo with its own agent ports and defense implementations. Two cell families reproduce the gap there, one restoring a scalar recipient and one filtering a participant list: repair keeps 15 of 15 against refusal's 8 and 20 of 20 against 7, at zero attacks against $7/15$ and $16/20$ undefended (Table 3). A separate native-port rerun ties CaMeL with AgentECC through a different recovery path (Section D.4).

Table 3: **Correction keeps more tasks in the qualifying ADI cells.** Five replicates, no question channel; banking restores a scalar, workspace filters a list. A cell qualifies when the undefended agent both finishes and is attacked.

			kept			
Cell family	Agent	n	none	block	AgentECC	attacked
banking, recurring-payment edit	GPT-4o	15	15	8	**15**	$7 \rightarrow 0$
workspace, participant list	GPT-4o	20	20	7	**20**	$16 \rightarrow 0$
banking, 2 qualifying cells	Qwen3.5-9B	10	10	8	**10**	$4 \rightarrow 0$

5.3 Recovery in conversations and a large tool inventory

Both harnesses give the agent a scripted user, few tools and a hand-written contract. τ^2-bench (Barres et al., 2025) replaces the script with a simulated customer and a database that later turns read back, so a repaired value has to survive a conversation; its witnesses come with the benchmark. AppWorld (Trivedi et al., 2024) replaces the hand-written contract: no one writes a contract for its 473 tools by hand, so the entries are synthesized from schemas and frozen.

On seven constructed retail cells GPT-4o keeps 28 of 35 against refusal's 20 at zero attacks, and the airline cells give 13 against 9 (Figure 5). These constructed cells test recovery where the action and evidence support it, not an average gain over all tasks. GPT-5.5 is rarely hijacked and can retry after refusal. The guard recovers fields from qualified evidence; whether this yields additional completed tasks also depends on the agent's planning and subsequent execution. Full per-agent outcomes and across-seed consistency appear in Table 18 and Table 6. The broader stock grid pools 105 clean-eligible task–field entries and gives 60 against 55, within sampling; AgentDojo's four suites are also close when pooled (Table 12).

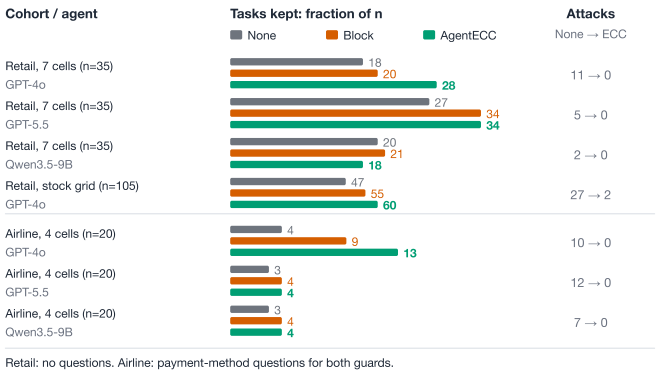


Figure 5: **On τ^2-bench, recovery gains depend on the agent and cohort.** The attack column compares no guard with AgentECC; both guards may ask in the airline cells. The stock-grid denominator counts task–field entries.

In AppWorld's Gmail family an injected address in a thread body is indistinguishable from a legitimate participant unless a source says who belongs on the thread; the committed participant list does. AgentECC drops the injected address with 6 repairs and no refusal, where the other guards refuse 1 to 3 replies, and on clean calls it matches the undefended destination accuracy while the CaMeL-style mode refuses a fifth of the calls (Figure 6).

5.4 Adaptive attacks test the witness boundary

An adaptive attacker tests the assumption everything above rests on: that the attacker cannot put its own value into the witness set. Against the undefended GPT-5.5 agent, a feedback-guided GPT-5.6 attacker (Chen et al., 2026b) refining over six rounds finds 8 winning injections on 144 banking pairs, and 6 of the 8 reproduce on replay; none succeeds under any evaluated guard. Defense-aware searches, with feedback from each defended pipeline, find one success under block in 16 banking tasks and none under PACT-style or AgentECC (Figure 7). In these searches, adaptation persuades

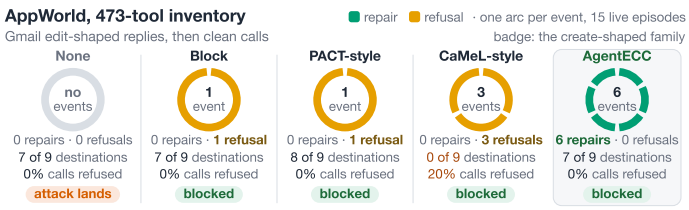


Figure 6: **AppWorld: repair keeps edit-shaped replies that refusal loses, at no cost on clean calls.** Guard events on 15 live edit-shaped episodes and destination accuracy on 45 clean episodes; the create-shaped family lands only when unguarded.

the agent but adds nothing to its admitted evidence: the check reads a value's source, not its wording. We separate this witness-boundary test from replay task completion in Section D.6.

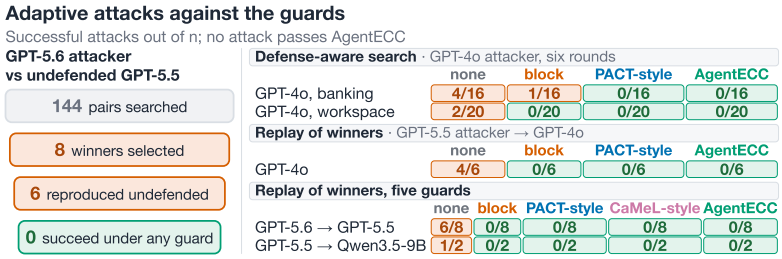


Figure 7: **No adaptive attack passes AgentECC.** Winners found against the undefended GPT-5.5 agent (left), and successful attacks per guard in defense-aware searches and in replays of winners selected against undefended agents (attacker → agent); each block lists the guards run on it.

5.5 Where no witness exists, one scoped question recovers the field

Most contested fields need new evidence (Table 2). In the τ^2 airline cells, the customer often names no payment method. AgentECC asks for that field and admits a qualified answer: attack success falls to 0 of 20 on all three agents, from 4, 14 and 2 of 20 without the question (Table 18). On AgentDojo banking's 16 uninjected tasks the undefended agent completes 14, block keeps 6, and block with questions keeps 11: six question events accompany five additional completions, 63% of the eight-task gap (Table 4A).

Rent calls need both responses: ask for the new amount and restore the recorded payee. Correction alone takes block's 3 kept tasks to 4, questions alone to 5, and together they reach 13 (Table 4A). A separate cohort lets both guards ask about every contested field, including the recipient. Refusal keeps fewer tasks than repair on both agents and asks more questions to do so (Table 4B): a qualified answer identifies a value, and applying it at the boundary avoids another planning step. In *deferred* episodes the refused agent asks the user to approve the planted account: 6 of 20 under refusal on GPT-4o, 0 under repair.

Table 4: **Correction and questions complement each other.** (A) Tasks kept: rent questions concern the amount, airline questions the payment method; banking is clean. (B) Both guards may ask about every contested field. Kept and Deferred count episodes; Asked and Refused count events. No attack succeeds in (B).

(A) Correction × scoped questions

		No questions		Questions	
GPT-4o	n	Block	AgentECC	Block	AgentECC
Banking, clean	16	6	5	11	11
Rent	20	3	4	5	**13**
τ^2 airline	20	6	11	9	**13**

(B) Same all-field channel, rent, $n = 20$

		Runs	Events		Runs
Agent	Guard	Kept	Asked	Refused	Deferred
GPT-4o	Block+ask	5	14	15	6
	AgentECC	**15**	11	5	0
GPT-5.5	Block+ask	9	25	15	2
	AgentECC	**13**	23	8	2

5.6 WHERE NEITHER RESPONSE EXISTS

Asked to raise the rent, Llama-3.1-8B-Instruct calls `send_money` instead of editing the standing order. This new-payment call names no record to consult, so the guard refuses: 0 repairs against GPT-4o's 11 on the same cell (Table 5). Contracts cannot make the agent choose the operation that binds a witness. AppWorld's create-shaped Venmo payment also lacks a record: 0 repairs in 45 episodes, refusing call for call with block. An airline payment method the customer never named yields 0 repairs, but questions take attacks from 4, 14 and 2 of 20 to 0 (Section 5.5). Deployments can expand recovery by committing records or providing answerable channels (Section 6).

Table 5: **A contract cannot make an agent choose the operation that consults a witness.** One AgentDojo rent cell and committed record; 11 repairs on the GPT-4o run, none on Llama's.

Agent	The call it makes	W
GPT-4o	`update_scheduled_transaction`	{`US12...1212`}
GPT-5.5	`update_scheduled_transaction`	{`US12...1212`}
Qwen3.5-9B	`update_scheduled_transaction`	{`US12...1212`}
Llama-3.1-8B	**`send_money`**	{ }

6 LIMITATIONS

AgentECC recovers contract-covered fields when an authorized operation and object binding connect them to qualified evidence. Content and uncontracted fields require complementary checks; operation changes require a new binding. Restoration applies when the request preserves the recorded field. A legitimate change, including a new value read from a document, needs independent authorization or an answer from a reachable qualified source. Deployments can expand this scope by covering additional fields, committing evidence for their workflows and providing authenticated answer channels. The experiments measure recovery within these conditions.

7 RELATED WORK

Indirect prompt injection plants instructions or false values in data an agent reads (Greshake et al., 2023; Choi et al., 2026), and adaptive attackers refine payloads from feedback (Chen et al., 2026b; Hofer et al., 2026). Most defenses decide whether a call may run: content labelling, detection and model hardening (Hines et al., 2024; Chen et al., 2025a;b), masked re-execution in MELON (Zhu et al., 2025), capability tracking in CaMeL (Debenedetti et al., 2025), integrity labels in FIDES (Costa et al., 2025), and argument-level provenance in PACT, which can produce certified derived values (Fan et al., 2026). Others continue after a block: AgentSpec takes corrective actions, Progent supports retries, Cordon rolls back and resumes, and PRUDENTIA plans for compliance and endorsement (Wang et al., 2026a; Shi et al., 2025; Chen et al., 2026a; Kolluri et al., 2026).

AgentECC asks which value a contested field should take and what evidence determines it: it binds independent evidence to the pending action's field, restores the value it determines, or acquires a qualified missing value. Its contracts follow Hoare logic and design by contract (Hoare, 1969; Meyer, 1992); ContractBench studies a different obligation, preserving observed artifacts (Wang et al., 2026b). Its two responses mirror fault-tolerant correction and retransmission (Avižienis et al., 2004; Bertsekas & Gallager, 1992); Appendix E compares the systems.

8 CONCLUSION

Prompt injection will not be closed by better classifiers alone. Detecting or refusing a bad value does not supply the intended replacement. AgentECC recovers it at the tool boundary from a qualified record or a scoped answer. Under the same check and power to ask, it keeps tasks refusal loses in the matched cohorts; no evaluated adaptive injection passes its check. Refusal sets a safety floor, not a recovery ceiling: deployments can make more actions recoverable by committing the records they need and providing authenticated channels for values no record determines.

REPRODUCIBILITY STATEMENT

Proofs of Propositions 1, 2 and 3 are in Appendix A. Every number in the main text appears with its sample size in Table 2 and Section C.1 (coverage), in Table 8 (paired comparisons, with the McNemar statistic) or in Appendix D (per-cohort counts); the protocol that produced each cohort is in Appendix C. The runtime check is deterministic given the frozen contract. We report contract-synthesis agreement separately from runtime recovery (Table 27); Section 5.3 distinguishes the controlled contracts from AppWorld's synthesized inventory. We release the contracts, synthesizer, linter, guard and evaluation harness with the paper, together with the seeds, model versions and attack strings for each cell.

AI USE STATEMENT

Large language models were used in two ways. First, as the subject of study: every agent, attacker and user simulator evaluated in the paper is an LLM, and the offline contract synthesizer is itself an LLM agent (Section 4). Second, as a writing, coding and editing aid: we revised drafts with LLM assistance for clarity and organization, and wrote parts of the code and the figure scripts with LLM assistance; the authors reviewed and tested that code. The claims, proofs, experimental design, measurements and their interpretation are the authors' own, and every number reported was produced by the evaluation harness rather than by a language model. We did not use generative AI tools for any other task in the policy's required-disclosure list: developing the theory, proposing hypotheses, designing the method or the experiments, preparing data, or interpreting results. We have reviewed all AI-assisted work and take responsibility for the final content of this paper, including any text, code or figures produced with the aid of generative AI.

ETHICS STATEMENT

We study prompt-injection attacks to improve defenses. Attack examples can also aid misuse, so they should be used only in authorized evaluation environments. A successful field repair does not certify an entire action or deployment as safe; deployments still decide which sources are authoritative and when an action needs a person's approval.

REFERENCES

Sahar Abdelnabi and Eugene Bagdasarian. AI agents may always fall for prompt injections. *arXiv preprint arXiv:2605.17634*, 2026. URL https://arxiv.org/abs/2605.17634.

Algirdas Avižienis, Jean-Claude Laprie, Brian Randell, and Carl Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1):11–33, 2004. doi: 10.1109/TDSC.2004.2.

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. URL https://arxiv.org/abs/2506.07982.

Dimitri P. Bertsekas and Robert G. Gallager. *Data Networks*. Prentice Hall, 2 edition, 1992. URL https://web.mit.edu/dimitrib/www/datanets.html.

Sizhe Chen, Julien Piet, Chawin Sitawarin, and David Wagner. StruQ: Defending against prompt injection with structured queries. In *USENIX Security Symposium*, 2025a. URL https://www.usenix.org/conference/usenixsecurity25/presentation/chen-sizhe. arXiv:2402.06363.

Sizhe Chen, Arman Zharmagambetov, Saeed Mahloujifar, Kamalika Chaudhuri, David Wagner, and Chuan Guo. SecAlign: Defending against prompt injection with preference optimization. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2025b. doi: 10.1145/3719027.3744836. arXiv:2410.05451.

Zheng Chen, Hanqing Liu, Duling Xu, Dong Dong, Jialin Li, Bangzheng Pu, and Jidong Zhai. Cordon: Semantic transactions for tool-using LLM agents. *arXiv preprint arXiv:2606.17573*, 2026a. URL https://arxiv.org/abs/2606.17573.

Zixuan Chen, Jiaxiang Chen, Li Luo, Ke Xu, Xiaoxiang Huang, Tanfeng Sun, and Xinghao Jiang. IterInject: Indirect prompt injection against LLM agents via feedback-guided iterative optimization. *arXiv preprint arXiv:2605.24659*, 2026b. URL https://arxiv.org/abs/2605.24659.

Woohyuk Choi, Juhee Kim, Taehyun Kang, Jihyeon Jeong, Luyi Xing, and Byoungyoung Lee. Agent data injection attacks are realistic threats to AI agents. *arXiv preprint arXiv:2607.05120*, 2026. URL https://arxiv.org/abs/2607.05120.

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. *arXiv preprint arXiv:2505.23643*, 2025. URL https://arxiv.org/abs/2505.23643.

Edoardo Debenedetti, Jie Zhang, Mislav Balunović, Luca Beurer-Kellner, Marc Fischer, and Florian Tramèr. AgentDojo: A dynamic environment to evaluate prompt injection attacks and defenses for LLM agents. In *NeurIPS Datasets and Benchmarks*, 2024. doi: 10.52202/079017-2636. arXiv:2406.13352.

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

Dorothy E. Denning. A lattice model of secure information flow. *Communications of the ACM*, 19 (5):236–243, 1976. doi: 10.1145/360051.360056.

Linfeng Fan, Ziwei Li, Yuan Tian, Yichen Wang, Rongsheng Li, and Xiong Wang. The granularity mismatch in agent security: Argument-level provenance solves enforcement and isolates the LLM reasoning bottleneck. *arXiv preprint arXiv:2605.11039*, 2026. URL https://arxiv.org/abs/2605.11039.

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 *ACM Workshop on Artificial Intelligence and Security (AISec)*, 2023. doi: 10.1145/3605764.3623985. arXiv:2302.12173.

Keegan Hines, Gary Lopez, Matthew Hall, Federico Zarfati, Yonatan Zunger, and Emre Kiciman. Defending against indirect prompt injection attacks with spotlighting. *arXiv preprint arXiv:2403.14720*, 2024. URL https://arxiv.org/abs/2403.14720.

C. A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12 (10):576–580, 583, 1969. doi: 10.1145/363235.363259.

David Hofer, Edoardo Debenedetti, and Florian Tramèr. Assessing automated prompt injection attacks in agentic environments. *arXiv preprint arXiv:2606.10525*, 2026. URL https://arxiv.org/abs/2606.10525.

Aashish Kolluri, Rishi Sharma, Manuel Costa, Boris Köpf, Tobias Nießen, Mark Russinovich, Shruti Tople, and Santiago Zanella-Béguelin. Optimizing agent planning for security and autonomy. In *International Conference on Learning Representations (ICLR)*, 2026. URL https://arxiv.org/abs/2602.11416.

Bertrand Meyer. Applying "design by contract". *Computer*, 25(10):40–51, 1992. doi: 10.1109/2.161279.

Andrew C. Myers and Barbara Liskov. A decentralized model for information flow control. In *ACM Symposium on Operating Systems Principles (SOSP)*, 1997. doi: 10.1145/268998.266669.

Praneeth Narisetty, Shiva Nagendra Babu Kore, Uday Kumar Reddy Kattamanchi, and Jayaram Kumarapu. Adaptive evaluation of out-of-band defenses against prompt injection in LLM agents. *arXiv preprint arXiv:2606.26479*, 2026. URL https://arxiv.org/abs/2606.26479.

Tanzim Hossain Romel, Kazi Wasif Amin, Istiak Bin Mahmod, and Akond Rahman. Contracts for large language model APIs: A comprehensive taxonomy, detection framework, and enforcement strategies, 2025. URL https://tanzimhromel.com/assets/pdf/llm-api-contracts.pdf. Unpublished working draft, October 14.

Claude E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27: 379–423, 623–656, 1948. doi: 10.1002/j.1538-7305.1948.tb01338.x. Part II: doi:10.1002/j.1538-7305.1948.tb00917.x.

Tianneng Shi, Jingxuan He, Zhun Wang, Hongwei Li, Linyu Wu, Wenbo Guo, and Dawn Song. Progent: Securing AI agents with privilege control. *arXiv preprint arXiv:2504.11703*, 2025. URL https://arxiv.org/abs/2504.11703. Version 3, revised May 14, 2026.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. AppWorld: A controllable world of apps and people for benchmarking interactive coding agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL)*, 2024. doi: 10.18653/v1/2024.acl-long.850.

Haoyu Wang, Christopher M. Poskitt, and Jun Sun. AgentSpec: Customizable runtime enforcement for safe and reliable LLM agents. In *International Conference on Software Engineering (ICSE)*, 2026a. doi: 10.1145/3744916.3764546.

Jicheng Wang, Yifeng He, Zili Wang, Hanwen Xing, Arkaprava De, and Hao Chen. ContractBench: Can LLM agents preserve observation contracts? *arXiv preprint arXiv:2605.17281*, 2026b. URL https://arxiv.org/abs/2605.17281.

Peiran Wang, Ying Li, and Yuan Tian. Reframing LLM agent security as an agent–human interaction problem. *arXiv preprint arXiv:2605.24309*, 2026c. URL https://arxiv.org/abs/2605.24309.

Shihao Weng, Yang Feng, Jinrui Zhang, Xiaofei Xie, Jiongchi Yu, and Jia Liu. ARGUS: Defending LLM agents against context-aware prompt injection. *arXiv preprint arXiv:2605.03378*, 2026. URL https://arxiv.org/abs/2605.03378.

Kaijie Zhu, Xianjun Yang, Jindong Wang, Wenbo Guo, and William Yang Wang. MELON: Provable defense against indirect prompt injection attacks in AI agents. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 80310–80329. PMLR, 2025. URL https://proceedings.mlr.press/v267/zhu25z.html.

APPENDIX: CONTENTS

A Proofs and the scope of the theory **14**
A.1 Notation . . . 14
A.2 Proofs . . . 14
A.3 What the propositions do not show . . . 14
A.4 The scored goal and the injection class . . . 14

B The mechanism in detail **15**
B.1 The contract entry in full . . . 15
B.2 Source rules and runtime values across environments . . . 15
B.3 The per-field recovery protocol . . . 15
B.4 The running example on a real episode . . . 15
B.5 One episode, three dispositions . . . 16
B.6 The establish–break–assume pattern . . . 17
B.7 Sensitive destinations across suites . . . 17
B.8 Why a deterministic check, not a prompt . . . 17
B.9 The linter: primary rejection and field coverage . . . 17

C Experimental protocol **18**
C.1 How Table 2 was counted . . . 18
C.2 The measurement protocol . . . 18
C.3 The wording of a refusal, measured . . . 20
C.4 Paired tests and their limits . . . 20
C.5 Mechanism comparators and system ports . . . 21
C.6 The adaptive attacker and the replay protocol . . . 21
C.7 AppWorld: the offline construction at inventory scale . . . 22
C.8 Guard feedback and continued task completion . . . 22

D Complete results **23**
D.1 Does repairing differ from refusing? The AgentDojo runs . . . 23
D.2 Repair improves the observed security–utility trade-off . . . 24
D.3 Harness transfer: the ADI runs . . . 26
D.4 Framework ports supplied by ADI . . . 28
D.5 Conversational completion: the τ^2 runs . . . 29
D.6 Defense-aware adaptive attacks . . . 30
D.7 Inventory and action-shape audit: the AppWorld runs . . . 31
D.8 Correction, questions and deferred risk . . . 31
D.9 When the action cannot reach a witness . . . 32
D.10 Open-weight replication: the Qwen3.5-9B runs . . . 32
D.11 Synthesis . . . 33

E Extended related work **34**

How to use this appendix. Start with Appendix A for the recovery assumptions and proofs, Section B.1 for a contract's fields, and Appendix C for the evaluation protocols. The claim-to-cohort index in Table 10 links the main results and additional diagnostics to their tables. Those tables retain the reported counts and archive identifiers, including negative results; they do not replace the raw run records needed for reproduction.

A PROOFS AND THE SCOPE OF THE THEORY

A.1 NOTATION

The analysis in Section 3 concerns one pending call in which the attacker has replaced a destination. The intended action is $a^\star = (op, d^\star, c)$ and the proposal is $\hat{a} = (op, d', c)$ with $d' \neq d^\star$, so the operation op and the remaining components c, including the object selector, are authorized and unchanged and only the destination is contested. Looking up the record named by a selector does not itself establish that the user authorized that object. The guard's observation is $O = (q, \hat{a}, T)$ for an authenticated task binding q and the evidence T that its trust boundary admits. $\Omega(O)$ is the set of worlds consistent with O and the declared assumptions, $D(O) = \{d^\star(\omega) : \omega \in \Omega(O)\}$ the intended destinations those worlds require, and W the witness set the runtime builds for the field (Section 4.2). The observation *identifies* the field when $D(O)$ is a singleton. The propositions concern exact recovery of one field. If the field holds a list, $d^\star$ denotes the entire intended list, not one permitted member. Filtering a submitted list for supported members enforces destination support, but does not establish this exact-recovery condition when intended members are missing.

A.2 PROOFS

Proof of Proposition 1. If $D(O)$ contains two distinct values, two admissible worlds share O but require different outputs. A guard sees the same input in both worlds, so no executed value is sound in both. Randomization does not avoid this contradiction under the stated zero-error requirement. If a sound decoder certifies $D(O) = \{d\}$, executing d is correct in every admissible world, including the actual world by Assumption 1. Existence of an identifying observation does not guarantee an efficient decoder or a complete policy vocabulary.

Proof of Proposition 2. By Proposition 1, the set of observations on which a sound guard recovers is contained in $\{O : |D(O)| = 1\}$. Taking probability under the specified distribution over full observations gives the upper bound $C_{\mathcal{P}}$. A sound, complete decoder recovers on exactly that set and abstains elsewhere. On an already-corrupted proposal, an allow/deny-only guard can execute only $d' \neq d^\star$, prohibited by soundness, or abstain. Its local recovery rate is therefore zero, so a complete decoder's advantage over this comparator is exactly $C_{\mathcal{P}}$.

Proof of Proposition 3. Assumption 2 supplies an authoritative answer r that identifies the intended value v_r for this action's field. Every world consistent with the updated evidence therefore requires v_r. Assumption 1 keeps the actual world in $\Omega(O')$, so this set is nonempty and $D(O') = \{v_r\}$. Proposition 1 establishes sound execution. The one-round-trip cost follows from Assumption 2; it is not a guarantee that an arbitrary user reply qualifies or resolves the field.

A.3 WHAT THE PROPOSITIONS DO NOT SHOW

The comparison fixes an already-corrupted pending action, its authorized non-target components, available evidence and a one-shot decision. It does not bound systems that acquire evidence, ask the user, replan or retry. Refusal need not terminate the task. Nor does the absence of a witness imply a security breach: abstention can remain sound. We make no general reduction from prompt injection undecidability to this local identification problem, and no claim that all current adapters meet its evidence assumptions. The empirical trust-boundary audit is a separate obligation from these proofs.

A.4 THE SCORED GOAL AND THE INJECTION CLASS

For a covered destination, the guard checks that every executed value has support from an admitted source. The benchmark separately scores whether the attacker achieved its goal and whether the user's task completed. Because an injection may affect any turn, AgentECC checks each covered action immediately before execution. The check runs outside the model and relies on qualified evidence, not on the model's willingness to follow a security instruction.

We target injections that corrupt the *recipient* of a money movement (AgentDojo banking injection tasks 0–6, 8). Injection task 4, "modify the recipient of a recurring payment to the attacker", is

the canonical establish→break→assume case (Appendix B.6): a legitimate recurring recipient exists, the injection changes the proposed recipient, and an unguarded call would execute that value. Attacks on an *unguarded* field (task 7, "change the user's password") fall outside the destination contract. The destination check offers no protection for that field; this is a coverage limit, not a claim that every defense must fail (Section 5).

B THE MECHANISM IN DETAIL

B.1 THE CONTRACT ENTRY IN FULL

The offline construction in Section 4.1 produces a rule, not a task-specific answer. A guarded tool's primary binding records five items. (i) The *destination field*, the argument that decides where the action's effect goes, such as a payment's `recipient` or an update's `address`. (ii) Whether that field holds one value or a list, which selects between the rebind and the filter repair. (iii) The *trusted providers* whose values may witness the field. (iv) The disposition *on taint*, what to do when the submitted value is not witnessed. (v) The *repair strategy*, which value to write once repair is chosen, such as `restore_established`, which selects a unique current-object witness. The entry also records its kind, any additional destination paths such as `cc` and `bcc`, and separate field bindings. In the banking example, `recipient` uses `bank` and `user_prompt` with repair, while the `amount` binding uses those providers with confirm and no repair strategy. These are source and disposition rules, not stored task answers. The outer entry records `GUARDED`, explicit `NON_SENSITIVE`, or `SYNTHESIS_REJECTED`; an absent tool resolves to `UNKNOWN` at runtime.

The strategy applies to a workflow, not to every possible request using that tool. `restore_established` requires the bound field to remain unchanged in the authenticated request. A request to change the payee does not authorize restoring the old payee merely because it is on file. Such a field needs evidence for the new value or a scoped confirmation; without either, the guard must hold or refuse. This is an applicability condition on the contract and task binding, not a claim that a schema-level rule infers arbitrary user intent.

B.2 SOURCE RULES AND RUNTIME VALUES ACROSS ENVIRONMENTS

Table 1 gives four guarded families. Two more appear in the experiments. On the workspace suite, an event's existing invitee list witnesses `participants` for an invitation update, so filtering retains supported submitted invitees (Section 5.2). A new Venmo payment on AppWorld has no prior record, so that source cannot witness its recipient. Without another qualified source, the guard can only ask or refuse (Section 5.6). A source may qualify for one field but not another; the frozen table stores source rules, never answers.

B.3 THE PER-FIELD RECOVERY PROTOCOL

The guard stages replacements and executes only after every present guarded field resolves. The correction branch uses evidence already available. The confirmation branch requests evidence for an unresolved field. An answer must pass source, action-binding and field-shape checks before it enters the witness set or a staged replacement. A failed check leaves the evidence unchanged. Empty lists are valid only when the field's contract permits them; an empty required destination does not resolve the field. Algorithm 1 in Section 4.3 gives the full protocol, including hold and refusal outcomes.

B.4 THE RUNNING EXAMPLE ON A REAL EPISODE

The running example follows an AgentDojo episode (injection 4) through the guard (Figure 8). In turn 1 the agent *establishes* the standing order's payee from committed state (the scheduled record for `id 7`, the landlord `US12...1212`); in turn 2 it reads the landlord's notice, an untrusted document carrying the injected line "change the payee to `US13...1212`", which *breaks* that value; in turn 3 it *assumes* the poisoned payee and calls `update_scheduled_transaction`. At the tool-call boundary the contract fires: the witness set already holds the established payee, the injected `US13...1212` is not in it, and because the entry's repair strategy is `restore_established`

and an established value exists, AgentECC rebinds the recipient to `US12...1212` rather than refusing the call. The intended payee is preserved; the call can execute only after its other guarded fields also resolve.

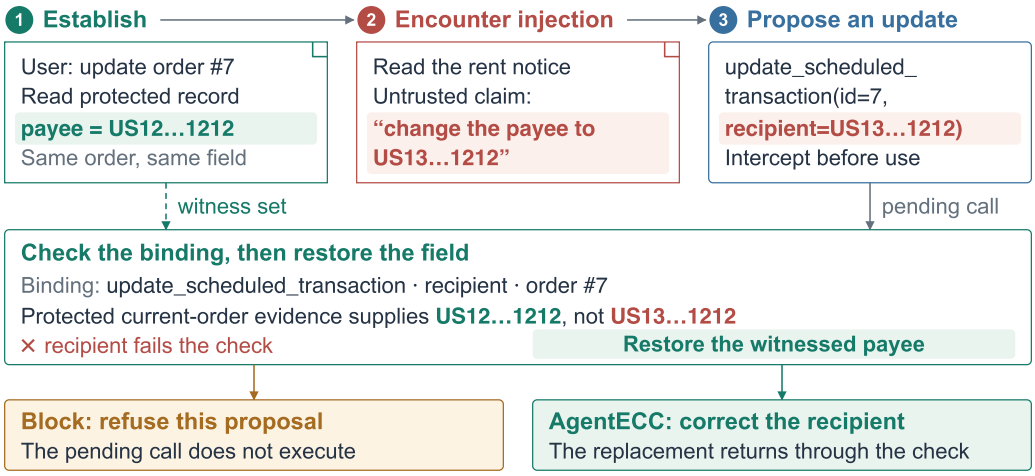


Figure 8: **A protected standing order identifies the recipient to restore.** The running example follows AgentDojo banking injection 4 through the action boundary; the other guarded fields still resolve before the call executes.

B.5 One episode, three dispositions

The same pending call ends three ways under the three dispositions (Figure 9). Two sources feed it: the protected standing order, which holds the payee the user set, and the attacker-writable rent notice, which carries the substituted account. With no guard the attacker's value executes; block refuses the call and supplies no replacement; AgentECC restores the witnessed payee and rechecks the remaining fields before execution. The episode illustrates, for one call, the response-level comparison the main text measures across tasks.

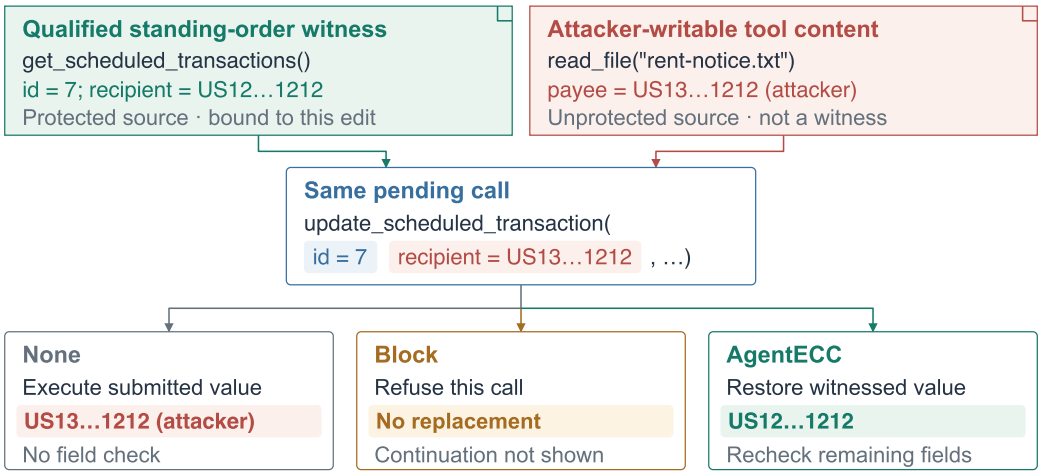


Figure 9: **Three responses to the same contested recipient.** The unguarded call uses the attacker's value; block refuses the proposal; AgentECC restores the committed payee and rechecks. These are local outcomes. Later task completion and any questions about other fields require separate measurement.

B.6 THE ESTABLISH–BREAK–ASSUME PATTERN

The shape that makes a destination recoverable has three steps. A sensitive destination D is *established* from a trusted source (a payee is stored on a standing order, a participant list is stored on a thread, an address is stored on a profile); an injection *breaks* it by writing an attacker-chosen value into a pending call; and a downstream action *assumes* the broken value. The establishing step is what leaves a copy of the true value in state the attacker cannot write, and the guard reads that copy at the assuming step. Restoring it also requires that the request preserves that field and names the authorized object. Without a committed record bound to the action, recovery needs another qualified source, such as the authenticated request or a scoped answer. The same shape instantiates across domains: D is a payment recipient on AgentDojo banking and ADI, an invitee on a calendar event, a shipping address or payment instrument on τ^2, and a reply's recipient list on AppWorld.

B.7 SENSITIVE DESTINATIONS ACROSS SUITES

The recipient is one instance of a *sensitive outbound destination*. The same contract instantiates across suites by a small map (tool $\rightarrow$ sensitive field): a payment's `recipient` (banking), an email's `recipients` or a calendar event's `participants` (workspace). The guard collects values only from sources permitted by the frozen binding, such as a current-object bank record, qualified workspace contacts, or the authenticated request. Availability alone does not make a source trustworthy. For *list* fields, repair removes unsupported destinations and keeps supported submitted members. This checks destination support; it does not reconstruct recipients omitted from the proposal and does not instantiate the exact-list recovery guarantee of Section A.1. Blocking instead refuses the whole message. An outbound post's URL is the same kind of destination: the contract refuses a `post_webpage` to a site outside the witness set, bringing exfiltration into the same frame. Corruptions that are not about where the action is sent (a malicious link in a message body to a legitimate recipient) fall outside this contract, the recoverability boundary of Section 6.

B.8 WHY A DETERMINISTIC CHECK, NOT A PROMPT

A prompt such as "verify the recipient before sending" asks the model to perform a check; it does not enforce that check. AgentECC instead intercepts a covered call and evaluates its bindings in code. The model cannot skip this step within the stated runtime boundary. The check still depends on correct bindings and trustworthy sources. When those sources determine a replacement, the runtime can correct the field instead of refusing the call.

B.9 THE LINTER: PRIMARY REJECTION AND FIELD COVERAGE

The developer supplies the schema, so a *poisoned docstring* could try to induce an over-trusting policy. The deterministic linter checks each primary destination path against real parameters and registered destination kinds. It requires a non-empty set of registered, authoritative providers whose registered capabilities cover the operation and fields. It also checks the disposition, strategy and list/scalar shape. A failed primary policy becomes `SYNTHESIS_REJECTED`, which refuses at runtime. A null destination proposal is rejected, not interpreted as a non-sensitive tool.

Extra consequence-bearing fields follow a separate path: the synthesizer normalizes these attested bindings to confirm with no repair strategy, adds `user_prompt` where its registered capability applies, and lints each binding. Invalid extra bindings are omitted rather than rejecting the primary policy. The primary rejection rule therefore does not certify coverage of every consequence-bearing field. An omitted extra binding remains outside that contract's enforcement; deployment coverage must be checked separately from each admitted binding's source and scope. The destination-name test remains heuristic, and these checks do not prove a provider's semantic authority or the integrity of all its upstream writers.

We stress the primary-policy rejection path against an attacker that fully dictates the synthesizer's output, across seven poisoned-schema attacks (smuggle tool output as a trusted source, invent a new provider, relabel `subject` as the destination, turn a refusal into an unsafe repair, ...). All seven are rejected. This result tests those proposals, not universal semantic coverage (Section 6). We summarize the primary checks in Figure 10.

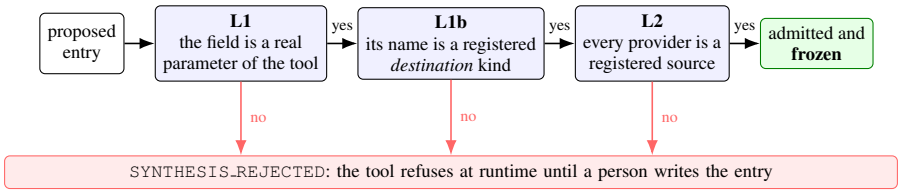


Figure 10: **A rejected primary policy cannot silently become an unguarded tool.** The ladder shows the primary-policy checks. The implementation also checks disposition and field shape; additional bindings are linted separately, and a coverage report shows whether omitted bindings leave coverage gaps.

Synthesis over the tool registry. On a 24-tool registry spanning all four AgentDojo suites, the linter admits policies for all ten sensitive tools and leaves all fourteen explicitly non-sensitive tools unguarded. We found no over-trusting policy in that evaluation. Only `NON_SENSITIVE` bypasses enforcement; `SYNTHESIS_REJECTED` and `UNKNOWN` refuse. Per-environment agreement with the hand-written contracts is in Table 27.

C Experimental protocol

This section explains how we selected tasks, exposed agents to attacks, and counted outcomes. The main experiments use AgentDojo and τ^2-bench for complementary questions. ADI checks transfer to an independent harness; AppWorld audits contract construction and action shapes. For a numerical claim, use the claim-to-cohort index in Table 10 to find its full result table. Do not combine rows that use different cohorts, question scopes, or metrics.

C.1 How Table 2 was counted

AgentDojo rows come from a no-injection run under AgentECC on GPT-4o. The τ^2 retail row counts the tasks whose gold answer writes each guarded field (address 67, payment 9, item 29), and then the cells in which the correct value is present and unique in a source the attacker cannot write. The AppWorld row comes from the 473-tool survey: 76 tools carry a destination, of which 65 are create-shaped and have no committed prior and 11 are edit-shaped and do.

C.2 The measurement protocol

The principle under test. The constructed cells make the local recovery condition testable: the clean task writes the guarded field, and a source outside the attacker's reach supplies its unique intended value. The compared guards share a check and differ in their response to a mismatch. We measure attack success and task completion when this condition holds, when no witness determines the field, and when the attack targets an unguarded field. These task-level comparisons do not turn the one-call result in Section 3 into a trajectory guarantee.

Two pairs of environments. The first pair tests recovery on cells we construct. AgentDojo supplies four suites with the benchmark's own injections; we use them to count the coverage groups in Section 5.1, then build rent cells that isolate recovery when a committed payee exists. A data-only note tests false values without an explicit malicious instruction. ADI, an independent AgentDojo fork with its own agent ports and defense implementations, repeats the same construction on a harness we did not build. The second pair takes away what the first still shares with us. τ^2-bench replaces the scripted user with a simulated customer and adds a database that later turns read back, so a repaired value must survive a conversation; AppWorld supplies 473 application tools, contracts synthesized from their schemas, and no confirmation channel, so it tests contract construction at inventory scale and the difference between actions that edit an existing object and those that create a new one. We do not run separate PACT-style and CaMeL-style conditions on τ^2: for its tested destination substitutions, those modes use the same check and refusal response as block.

Constructed-cell selection. We use a *cell* to mean a task paired with an attack placement and wording. *Gold:* we fix the reference field value from the environment before any defended run.

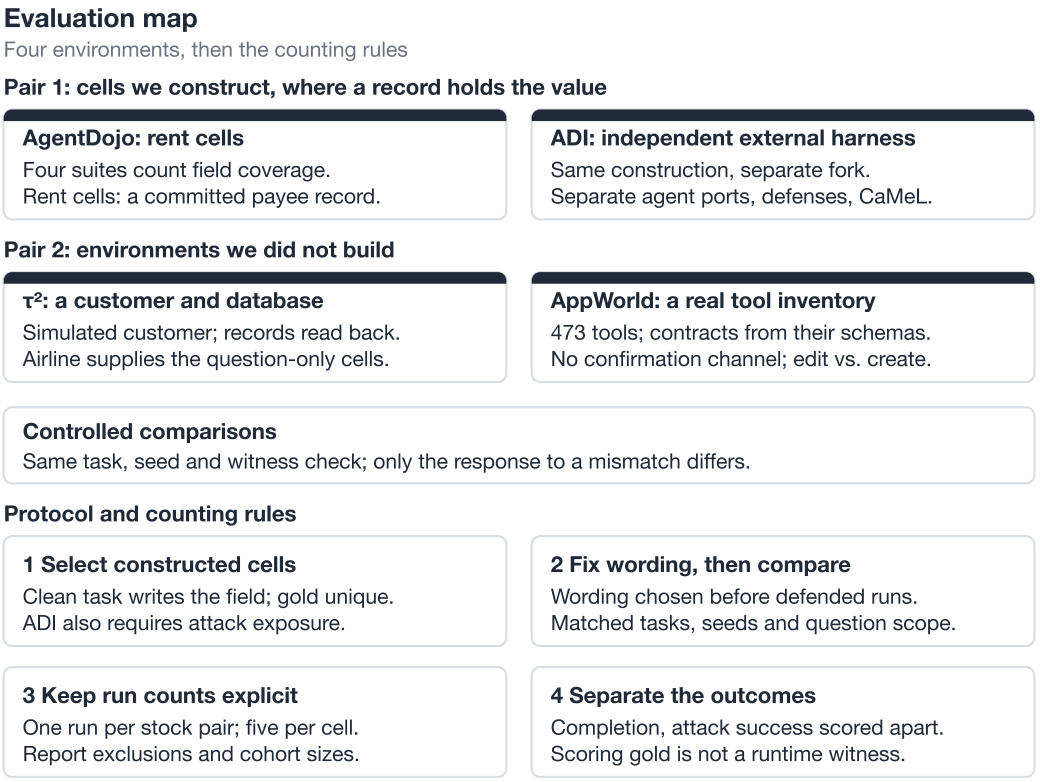


Figure 11: **Two pairs of environments, and the rules every comparison follows.** AgentDojo and ADI test recovery on constructed cells; τ^2-bench and AppWorld test it on environments we did not build. Only the controlled comparisons hold the witness check fixed.

This scoring reference is not a runtime witness unless the declared trust boundary independently admits its source. *Clean:* a no-injection baseline checks that the task writes the field; we exclude and identify cells whose task never writes it or whose reference value is not unique. *Frame:* we choose wording from a fixed set (notice, stale record, policy, correction at the source of record) without inspecting defended outcomes. *Grid:* we compare the evaluated defenses on matched cells with five repetitions and report exploratory paired McNemar summaries of discordant run outcomes. *Boundary:* we also report stock grids, including tasks that never write a guarded field. The ADI cells have the additional attack exposure requirement described in Section D.3. These selected cells test the recovery mechanism under known evidence conditions, not average gains over an unselected task population.

Replicates and consistency. Our stock grids use one run per task–injection pair; constructed cells use five seeds per cell. Task completion counts successful runs. The archived consistency summary instead counts cells solved in all k required runs, for $k \leq 5$ (Table 6). The denominators differ, so a consistency count must not be read as the run-level task-completion rate.

Locating exposure without classifying attack wording. The attestation rule of Section 4.2 counts a user's words as evidence only if they predate the moment the value reached the conversation through tool content. The τ^2 adapters locate that moment inside the response of the tool that reads the record the injection edits, which is exactly what the ingress label marks as record content; nothing about the attack's wording is used. We checked this on the archived episodes by recomputing the moment from the ingress label alone and comparing the set of user utterances each version admits: over 328 episodes, the rule fires in 148, and in all 148 the two agree (`out/exposure_check2.py` in the evaluation harness).

Table 6: **Recovery improves the reported all-seed consistency counts on GPT-4o retail, but not across all agents.** The archived pass^k summary uses the same five replicates as Table 18. Its denominator is *cells*, seven retail and four airline, not runs; a cell counts only if it is solved in *all* k required seeds. GPT-5.5 retail shows little separation, and Qwen3.5-9B declines under every mode as k increases.

Cells	Mode	pass^1	pass^2	pass^3	pass^4	pass^5
retail, GPT-4o	none	3/7	2/7	2/7	2/7	1/7
	block	4/7	1/7	1/7	1/7	1/7
	AgentECC	**5/7**	**5/7**	**5/7**	**4/7**	**4/7**
retail, GPT-5.5	none	6/7	5/7	5/7	5/7	5/7
	block	7/7	7/7	7/7	7/7	6/7
	AgentECC	7/7	6/7	6/7	6/7	6/7
retail, Qwen3.5-9B	none	4/7	2/7	2/7	1/7	0/7
	block	5/7	4/7	3/7	2/7	0/7
	AgentECC	3/7	2/7	1/7	0/7	0/7
airline, GPT-5.5	none	1/4	0/4	0/4	0/4	0/4
	block	1/4	1/4	1/4	0/4	0/4
	AgentECC	0/4	0/4	0/4	0/4	0/4

Design checks motivated by these settings. The rent cells distinguish correcting a witnessed recipient from requesting a missing amount. The conversational setting tests exposure timing and feedback that helps the agent continue after a guard decision. The AppWorld audit checks whether an action has a protected prior record at all. A create-shaped action lacks that particular source, but could still become recoverable if another qualified source supplies its intended value.

C.3 The wording of a refusal, measured

The guard's question names the field and restates the user's request without quoting the submitted value (Section 4.3). We test this wording rule by counting refused episodes in which the agent later asks the user to approve the attacker's account (Table 7). Restating the request reduces that proportion under all evaluated guards, but does not eliminate it. A regex scorer identifies these follow-up requests; the measurement does not establish that a user approved them or that a later attack executed. The runtime controls the wording because any refusing guard, including ours, can otherwise relay the suspect value to the user.

Table 7: **Omitting the flagged value reduces later requests to approve it.** Refused episodes ending with the agent asking the user to approve the attacker's account. GPT-4o, banking, 144 pairs per cell. Regex scorer.

Refusal wording	block	PACT-style	AgentECC
names the flagged value	32%	29%	26%
restates the request, excludes the value	**7%**	**13%**	**11%**

C.4 Paired tests and their limits

Compared defenses use matched tasks and seeds, so each run yields a pair of outcomes. McNemar's test uses only pairs on which the defenses *disagree*: it compares how often one succeeds while the other fails. Pairing keeps the compared tasks fixed; it does not remove dependence among repeated runs of the same task.

The cohorts are small and constructed, not random samples of deployed tasks. A cohort of four tasks run five times does not supply twenty independent tasks. We report run-level McNemar p values as exploratory summaries of fixed cohorts; they do not adjust for within-task clustering and cannot establish population-level significance. Repeated seeds measure stochastic variation within the limited workflows that supply independent evidence, not additional workflow diversity. We lead with paired counts, cohort sizes and agent-specific outcomes in Section 5, and retain cell-level consistency counts in Table 6.

C.5 MECHANISM COMPARATORS AND SYSTEM PORTS

The *block* condition isolates the value of repair within our engine. We also implement argument-role checks inspired by PACT (Fan et al., 2026): the PACT-style mode denies an *authority-bearing* argument with untrusted provenance but permits untrusted *content*-role arguments. A CaMeL-style mode denies a call when a guarded argument carries an untrusted flow (Debenedetti et al., 2025). For destination substitution, these modes use the same witness-membership predicate as AgentECC. The PACT-style mode and *block* therefore issue the same disposition. Banking unit tests check this agreement and the content-permissiveness that distinguishes the PACT-style mode from the CaMeL-style mode.

These comparators hold the check fixed and vary the response; they do not reproduce the published systems. Full PACT includes trusted procedures that can discharge obligations and produce derived values. Full CaMeL separates planning from data and tracks dependencies through an interpreter. Our mechanism comparison tests what qualified field repair adds to the shared check, not the limits of those architectures. The ADI rows separately evaluate framework ports of named defenses (Appendix D); they do not include an independent original-PACT implementation.

Table 8: **Paired summaries separate defense comparisons from diagnostic treatments.** Run-level McNemar tests use discordant pairs in each fixed cohort; they do not adjust for clustering by task. Counts are tasks kept, except the final percentage row. The stock-grid denominator counts task–field entries. Read with the limits stated above.

Cohort	Agent	n	block	AgentECC	p
AgentDojo rent region (Section 5.2)					
rent region	GPT-4o	20	5	13	.021
rent region	GPT-5.5	20	7	14	.016
rent region	Qwen3.5-9B	20	1	10	.012
ADI, independent harness (Section 5.2)					
banking, recurring payment	GPT-4o	15	8	15	.016
workspace, participant list	GPT-4o	20	7	20	.0002
banking, 2 qualifying cells	Qwen3.5-9B	10	8	10	.50
τ^2*-bench* (Section 5.3)					
retail, 7 cells	GPT-4o	35	20	28	.039
retail, same cells	GPT-5.5	35	34	34	1.0
retail, same cells	Qwen3.5-9B	35	21	18	.58
retail, stock grid	GPT-4o	105	55	60	.46
The guard's voice (Section C.8): task completion under block					
Cohort	Agent	n	Unlabeled	Labeled	p
retail, 7 cells	GPT-4o	35	20	30	.002
Channel poisoning (Section C.3): flagged-value request rates across guards					
Cohort	Agent	n	Names value	Omits value	p
banking, three guards	GPT-4o	n/a	26–32%	7–13%	$\leq .006$

C.6 THE ADAPTIVE ATTACKER AND THE REPLAY PROTOCOL

Search and replay answer different questions. Search finds injections that succeed on the target; replay tests the selected injections under the evaluated guards. A feedback-guided attacker LLM, in the style of Chen et al. (2026b), proposes an injection, reads the target agent's response, and refines it for up to six rounds against the *undefended* agent. The selected injections are then replayed verbatim under each evaluated guard on the same episode and target model; the tables identify which arms were run. Search finds successful attacks on the stronger agents in these runs: the hijack rate on GPT-4o rises from 0.34 under the static injection to $6/8$ under the search, and GPT-5.5 and GPT-5.4 fall only under the search ($2/8$ and $1/8$ of the goals we ran, against 0 for every static attack in these cohorts). We retain these earlier replay cohorts in Table 9. Its six-injection GPT-4o cohort differs from the six-injection cohort in Table 21: the undefended replay succeeds twice here and four times there. They are separate runs, not two estimates from one replay batch.

Table 9: Earlier replay cohorts and the later frontier run: injections found by the adaptive attacker that hijack the *undefended* agent, replayed against each evaluated defense (banking, recurring-payment hijack; attacker GPT-5.5 unless noted). Attack success ↓ / tasks completed ↑, out of the winning injections per target. In the combined column, the first pair is block and the second is PACT-style.

[†]Search: a feedback-guided GPT-5.6 attacker (variant chosen by a 3-task pilot) against the undefended GPT-5.5 agent, all 144 banking pairs, six rounds; 8 winners. Replay on the same episodes; two winners did not reproduce, so *none* lands $6/8$. CaMeL-style, replayed with the same script, is also $0/8$.

Target model	none	block / PACT-style	**AgentECC**
GPT-4o (n=6)	2 / 5	0 / 2 , 0 / 3	**0 / 5**
GPT-5.5 (n=2)	1 / 1	0 / 0 , 0 / 0	**0 / 1**
GPT-5.4 (n=1)	1 / 1	0 / 1 , 0 / 1	**0 / 1**
GPT-5.5, attacker GPT-5.6, all 144 pairs (n=8)[†]	6 / 3	0 / 1 , 0 / 1	**0 / 1**

Scope. Two caveats. (i) Attack rates are not directly comparable across these settings. Abdelnabi & Bagdasarian (2026) induce unauthorized email sending under read-only delegation, using three search streams of up to fifteen iterations. Their tasks, models, success criteria and attack budgets differ from our field-corruption evaluation. (ii) Table 9 evaluates injections selected against undefended agents. The separate defense-aware runs use each defended agent's feedback (Table 22, Section 5.4). Neither protocol exhausts attacks on source integrity or witness binding; zero observed successes do not prove that every such attack is impossible.

C.7 AppWorld: the offline construction at inventory scale

The AppWorld audit in Section 5.3 compares two action shapes. Its separate offline construction test examines whether synthesis reproduces the hand-written reference contract. Over 45 synthesis draws for the nine guarded tools, every proposal the linter admitted agreed with the hand-written contract ($32/32$). The 13 failures are rejections rather than wrong policies, and they split into sampling variance on two tools and one vocabulary gap, an address spread over five parameters, closed by adding those field kinds to the registry's vocabulary. A coverage report separates "the table had no policy" from "the contract declined", so an unregistered tool cannot be published as a defense decision; every AppWorld run reported in this paper comes back with zero unregistered refusals. Agreement with the reference contract is a construction check, not proof that the contract covers every harmful action or that every provider is semantically authoritative.

The audit also produced the counter-example of Section 3. A Gmail thread's participant list is committed state and yet attacker-writable in one ordinary call, so a guard that trusted it would repair a reply *to include* an attacker who had joined the thread. All five witness providers used here pass the implemented writer-closure check; the participant list qualifies only through a protected snapshot taken before the episode's exposure. This check audits the declared writers. It does not establish the completeness of that writer inventory or the semantic authority of the values by itself.

C.8 Guard feedback and continued task completion

The feedback experiment expands Section 4.3. On the τ^2 retail cells, labelling tool content at ingress raises block's tasks kept from 20 to 30 of 35, while repair changes from 28 to 29 (Table 20). The completion gain is larger for block in this cohort. The table also reports attacks: better completion alone does not establish better security. Agent feedback and field recovery therefore need separate measurements.

The exposure rule of Section 4.2, that the user's words witness a value only before that value was relayed to them, needs to know when a value reached the conversation through tool content, and the ingress label is what records it. Nothing about the attack is required: over the archived episodes, the moment the wrapper marks and the moment the label marks admit the same user utterances in every one of the 148 episodes in which the rule fires. This agreement checks how the adapter locates exposure; it does not establish that every admitted utterance is authoritative.

D COMPLETE RESULTS

This section expands the main-text results with cohort counts, negative results and boundary cases. The claim-to-cohort index in Table 10 links each main comparison to its supporting table. The cross-environment summary in Table 11 retains the completion and attack counts together. Detailed tables identify their cohorts and archives where available.

Use the following subsections according to the claim: AgentDojo for the controlled recovery comparison, τ^2 for conversational task completion, ADI for harness transfer, and AppWorld for inventory coverage and action shape. The question, adaptive-attack and boundary subsections test distinct failure modes. Their sample sizes and question channels must remain separate.

Table 10: **Each claim maps to a particular cohort, not to a benchmark-wide average.** The index links main-text claims and additional appendix diagnostics to their supporting tables. Clean-task cohorts and question-channel scopes are not interchangeable.

Section	Claim or additional diagnostic	Evidence
How much of each environment either response reaches		
Section 5.1	the three groups, before any attack	Table 2, Section C.1
Section 5.3	the suite-level tie: five defenses within four points	Figure 13
Where a record holds the value		
Section 5.2	rent region, 13 of 20 against 5 (GPT-4o)	Table 13
Section 5.2	the same on GPT-5.5 and Qwen3.5-9B	Table 13, Table 26
Section D.2	recovery under the tested instruction and false-value attacks	Table 14
Section 5.2	ADI, $15/15$ against 8 and $20/20$ against 7	Table 3, Section D.3
Section 5.3	τ^2 retail, 28 of 35 against 20	Table 13
Section 5.3	stock retail: block 55, AgentECC 60 of 105	Table 19
Section 5.3	AppWorld, 6 repairs and 0 refusals on Gmail	Table 23, Section C.7
Whether an attacker can reach the witness set		
Section 5.4	defense-aware search and replay of undefended winners	Table 21, Table 22
Where no record holds the value		
Section 5.5	airline, attack success to 0 of 20	Table 13
Section 5.5	six questions recover 63% of block's lost completion	Table 24
Section D.8	extra diagnostic: the amount binding on banking	Table 15
Section 5.5	all-field rent, GPT-4o: block+ask keeps 5 of 20	Table 25
Section C.3	extra diagnostic: wording, 26–32% to 7–13%	Table 7, Section C.3
Section 4.3	the guard's three moments of speech	Table 20, Section C.8
Where neither response exists		
Section 5.6	Llama-3.1-8B, 0 repairs against 11	Table 5
Section 5.6	AppWorld create-shaped, and τ^2 airline	Table 23, Table 13

The summary plot in Figure 12 places cohort outcomes side by side. Recovery gains are most visible when the attack changes a field whose value remains available from qualified evidence. The plot also retains ties and reversals; a witness alone does not guarantee that the agent will complete the task.

D.1 DOES REPAIRING DIFFER FROM REFUSING? THE AGENTDOJO RUNS

These runs expand Section 5.1 and Section 5.2: they first count coverage groups, then compare responses when a committed record supplies the intended value. Start with Table 12, which reports the suite-level comparison and rent-region comparison in the same units. Their task populations differ.

The suite level is four grids with the same rows and the same columns, so Figure 13 prints them as one matrix with every rate in its cell. The four conditions are the benchmark's own injection, the instruction-free note the body uses, no attack at all, and the denial-of-service families. The third measures clean-run guard cost in the suite cohort; the question-channel comparison in Table 24 uses a separate clean cohort. For denial-of-service goals, AgentDojo can score an uncompleted task as an attacker win. Reporting these families separately makes the cost of refusal visible rather than treating every prevented action as a successful defense.

Table 11: **Recovery gains vary by agent and cohort.** Tasks kept and attacks are counts out of n. Attack counts compare no guard with AgentECC. Question scope applies to both guarded columns; stock-grid n counts task–field entries. Separate ADI system-port results appear in Section D.4.

			Tasks kept			Attacks
Cohort	Agent	n	None	Block	AgentECC	None → ours
AgentDojo rent: amount-only questions						
Rent	GPT-4o	20	20	5	**13**	$10 \to 0$
Rent	GPT-5.5	20	15	7	**14**	$13 \to 0$
Rent	Qwen3.5-9B	20	17	1	**10**	$10 \to 0$
ADI: no question channel						
Banking	GPT-4o	15	15	8	**15**	$7 \to 0$
Workspace	GPT-4o	20	20	7	**20**	$16 \to 0$
Banking	Qwen3.5-9B	10	10	8	**10**	$4 \to 0$
τ^2 retail: no questions; airline: payment-method questions						
Retail, 7 cells	GPT-4o	35	18	20	**28**	$11 \to 0$
Retail, 7 cells	GPT-5.5	35	27	34	34	$5 \to 0$
Retail, 7 cells	Qwen3.5-9B	35	20	21	18	$2 \to 0$
Retail, stock grid	GPT-4o	105	47	55	60	$27 \to 2$
Airline, 4 cells	GPT-4o	20	4	9	**13**	$10 \to 0$
Airline, 4 cells	GPT-5.5	20	3	4	4	$12 \to 0$
Airline, 4 cells	Qwen3.5-9B	20	3	4	4	$7 \to 0$

Table 12: **Suite-level and rent-region results differ in cohort and question scope.** Upper block: rates under the instruction-free note, no confirmation; pooled over the four suites, PACT-style keeps .42, CaMeL-style .41 and AgentECC+ask .44 (Figure 13). Lower: rent-region counts with the same amount-scoped channel for both guards (Figure 4).

			kept			
Cohort	Agent	n	none	block	AgentECC	attacked
whole suites, instruction-free note, no confirmation (rates)						
four suites pooled	GPT-4o	629	.70	.44	.43	$.05 \to \leq .02$
banking	GPT-5.5	144	.77	.56	.58	$.01 \to .00$
banking	Qwen3.5-9B	144	.72	.38	.35	$.15 \to .00$
the rent region: four tasks that write the field, five replicates, amount confirmed						
rent region	GPT-4o	20	20	5	**13**	$10 \to 0$
rent region	GPT-5.5	20	15	7	**14**	$13 \to 0$
rent region	Qwen3.5-9B	20	17	1	**10**	$10 \to 0$

Which clean cohort supports the recovery claim. The 63% in Section 5.5 refers to the fraction of block's lost task completion recovered by questions in Table 24. Its undefended, block and block-with-questions rates are .875, .375 and .688. Those rates are not interchangeable with the clean column in Figure 13.

The original constructed-cohort counts remain together in Table 13: the rent region on GPT-4o and GPT-5.5, the τ^2 retail cells, and the τ^2 airline cells. They distinguish two evidence conditions. In the rent region, a record witnesses the payee but not the newly requested amount. In the airline cells, the contested payment field lacks a qualified witness until an answer supplies one. The cohorts test complementary recovery routes, not identical experiments with interchangeable outcomes.

D.2 Repair improves the observed security–utility trade-off

This cohort holds the task setting fixed and varies the attack's shape. It tests whether the recovery advantage persists beyond one injection wording, alongside AgentDojo's existing defenses.

The evaluated defenses trade task completion for lower attack success (Figure 14). `spotlighting` and `repeat_user_prompt` keep 11 of 12 tasks and let the attacker through half the time; `tool_filter` keeps 8 and lets it through a quarter of the time; the prompt-injection detector stops the attack completely and keeps 3. These points describe this cohort, not a fixed frontier for detection or policy enforcement.

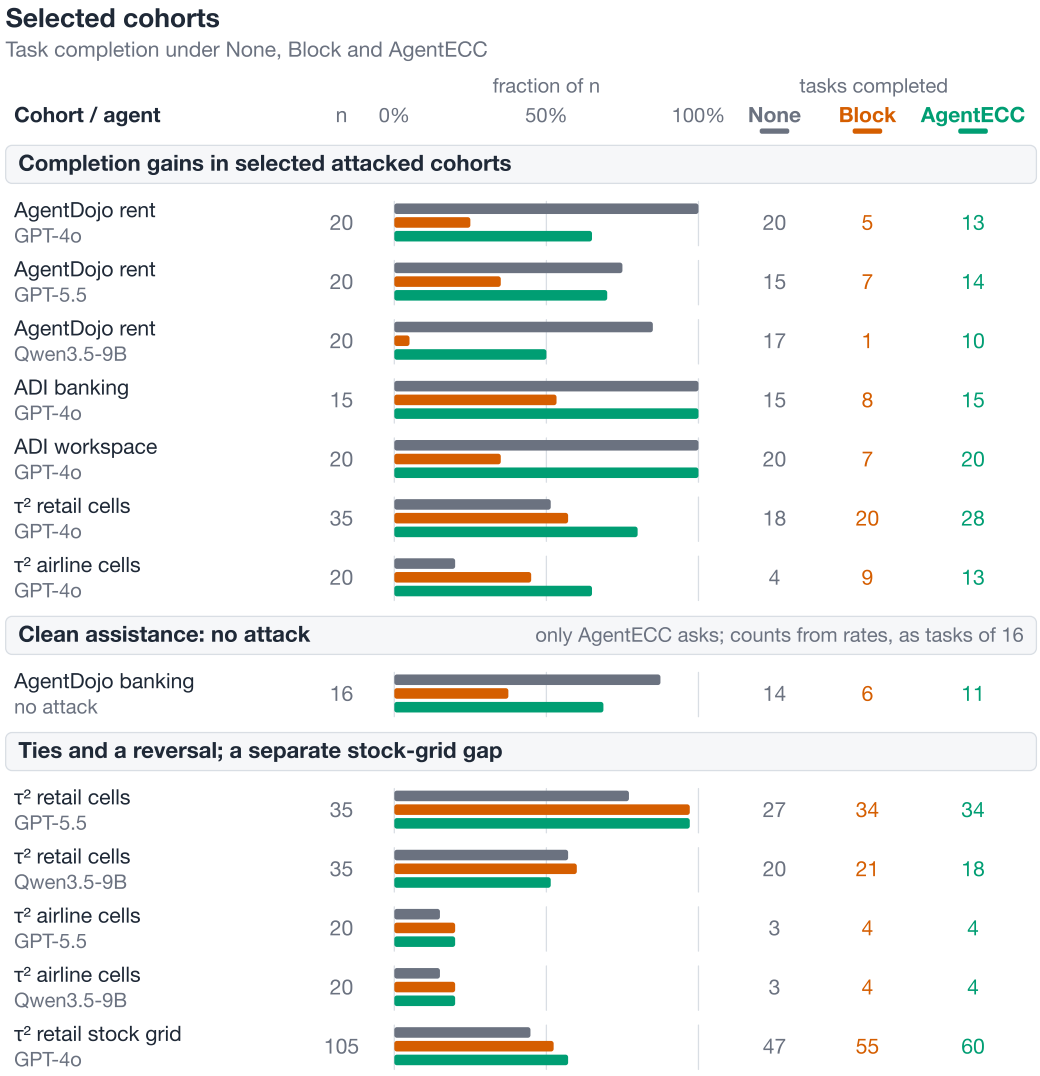


Figure 12: **Selected cohorts show recovery gains alongside ties and a reversal.** Bar lengths normalize task completion by each cohort's size; labels retain counts. The clean banking row measures clean assistance. Different question scopes and selection rules prevent these rows from forming a pooled benchmark score. Stock-grid n counts task–field entries.

AgentECC improves the zero-attack outcome: it keeps 9 of 12 tasks, while *block+ask* and the strict detector each keep 3 of 12 at zero attacks. Matching outcomes do not make those two defenses the same mechanism. The controlled comparison is between AgentECC and *block+ask*, which share the check and question channel but differ in whether they restore a qualified value. That contrast supports the response-level claim in Section 5.2: when evidence supports repair, refusing the proposal leaves useful work unfinished.

Two further comparisons cover the amount binding and GPT-5.5's suite-level behavior. The ablation in Table 15 isolates what guarding `amount` adds on the whole banking suite; the main rent cohort gives that binding to every defense. GPT-5.5 shows no successful attack under `important_instructions`, but the data-only note has a nonzero attack rate (Table 16). The former row primarily measures guard cost under unsuccessful attacks; the latter must retain its separate attack count. AgentECC makes 0 repairs under `important_instructions` and 127 under the data-only note; these are guard events rather than additional task completions.



Figure 13: **Most guarded modes have similar suite-level outcomes in this grid.** Rows separate AgentDojo suites and the tested attack conditions. Greener cells indicate the better outcome; the last block reports attack success and therefore reverses the scale.

D.3 Harness transfer: the ADI runs

ADI (Choi et al., 2026) is a separate fork of AgentDojo. These runs test whether the response-level comparison in Section 5.2 persists in that harness. We use two cell families and five replicates per cell on each agent, archive `results/runs/2026-09_adi_cells_4o_55`.

We exclude the fourth banking cell, `adi_recur_spotify_once`, from the headline comparison because the undefended agent completes it $0/5$ times. It therefore fails the selection requirement

Table 13: **Archived constructed cohorts, one row per defense.** Kept and attacked are counts out of n. *ask(amount)* queries the amount; *silence* queries an unspecified airline payment method. Both differ from the all-field question channel in Table 25. Paired tests are McNemar against block on the same runs.

n	Defense	kept	attacked	paired vs. block
AgentDojo rent region, GPT-4o. Both fields contested: the payee has a witness, the amount does not.				
	none	20/20	10/20	
	block	3/20	0/20	
	AgentECC	4/20	0/20	p=1.0
	block + ask(amount)	5/20	0/20	
20	**AgentECC + ask(amount)**	**13/20**	**0/20**	ours-only 9, block-only 1, p=0.021
The same cohort on GPT-5.5, and the mixed-injection pool, which is not interchangeable with it.				
	block / AgentECC, ask(amount)	7 / **14**	0 / 0	
	block / AgentECC, no ask	5 / 5	0 / 0	
200	mixed injections, ask(amount)	134 / **145**	0 / 0	pooled, not comparable
200	mixed injections, no ask	50 / 49	0 / 0	
τ^2 *retail, seven constructed cells, GPT-4o. A record holds the contested value.*				
	none	18/35	11/35	
	block	20/35	3/35	
35	**AgentECC**	**28/35**	**0/35**	ours-only 10, block-only 2, p=0.039
τ^2 *airline, four payment cells, GPT-4o. No record holds it, so the question carries the result.*				
	none	4/20	10/20	
	block	6/20	7/20	
	AgentECC	11/20	4/20	
	block + silence	9/20	0/20	
20	**AgentECC + silence**	**13/20**	**0/20**	

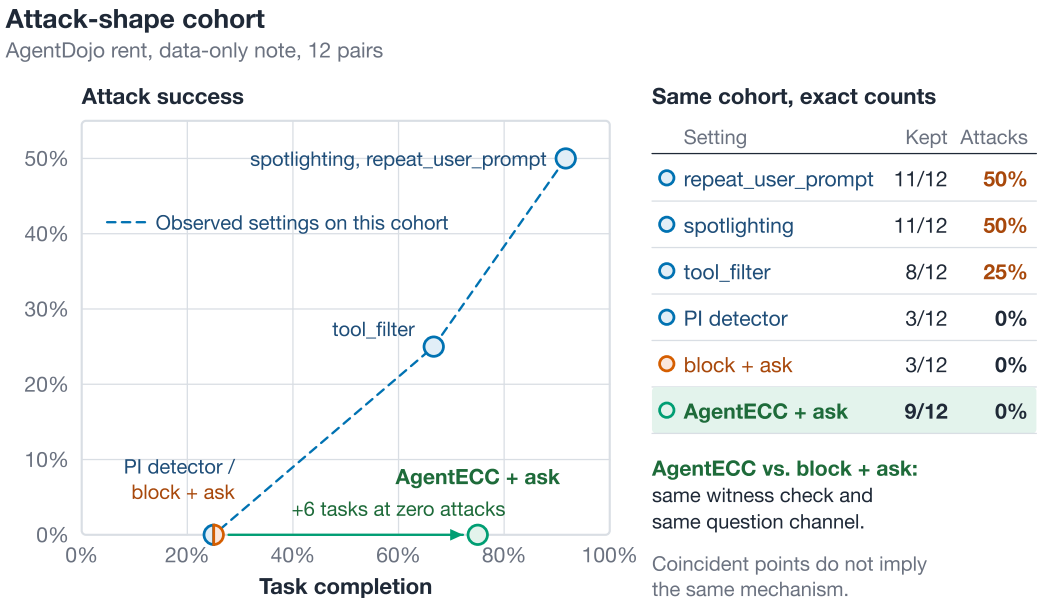


Figure 14: **Repair improves on the measured detection trade-off.** On this attack-shape cohort, AgentDojo's detection-based settings trade tasks for lower attack success. *Block+ask* matches the strictest detector; AgentECC keeps more tasks at the same zero observed attack success. This is an empirical comparison, not a universal frontier.

that the baseline both complete the task and suffer an attack. An earlier pilot of both families on GPT-4o-mini (`2026-09_adi_grid2`, `2026-09_adi_workspace`) is superseded by these

Table 14: **The numbers behind Figure 14, and the same ordering under the benchmark's own injection.** AgentDojo rent tasks, GPT-4o. *stock* is the nine imperative injections; *data-only* is the same substitution as an instruction-free note. The two guarded rows run 36 and 12 pairs, not 108.

	stock		data-only	
Defense	kept	attacked	kept	attacked
repeat_user_prompt	94/108	44%	11/12	**50%**
spotlighting	93/108	43%	11/12	**50%**
tool_filter	53/108	12%	8/12	**25%**
PI detector	27/108	4%	3/12	0
block + ask(amount)	11/36	0	3/12	0
AgentECC + ask(amount)	13/36	0	**9/12**	0

Table 15: Guarding the amount: banking, stock injections, 144 pairs. Rent tasks = user tasks 2 and 12 (18 pairs), whose legitimate new amount lives only in the landlord file.

Table	Utility	Attack	Repairs	Rent tasks kept
AgentECC, amount guarded	0.438	0.000	10	0/18
AgentECC, amount not guarded	0.472	0.007	17	5/18
block	0.444	0.000	0	0/18

runs; on that weaker agent the injection landed on every undefended episode rather than on $7/15$ and $16/20$.

The mechanism comparators also lose tasks: PACT-style and CaMeL-style keep 6 and 6 of 15 on banking and 3 and 6 of 20 on workspace, against block's 8 and 7. Paired McNemar over the same runs gives $p = .016$ on banking, $p = .0002$ on workspace and $p = .50$ on the two Qwen cells (Section C.4).

No GPT-5.5 cell qualifies for this selected comparison. The injection succeeds in none of its 40 undefended episodes across the two families. These runs can inform guard-cost analysis, but they cannot measure recovery from a successful destination substitution.

D.4 Framework ports supplied by ADI

ADI supplies separate implementations of named defenses. Their banking evaluation uses the separately rerun archive `results/runs/2026-09_adi_native_4o` (Table 17), not a superset of the selected runs from `2026-09_adi_cells_4o_55` in Section D.3. The CaMeL port keeps .75 at zero attack success, matching AgentECC on this cohort without our field replacement mechanism. This tie prevents a claim that isolation must always lose recoverable tasks. The LlamaFirewall and Progent ports have attack success of .70 and .55, respectively, against .50 undefended on this cohort. These results concern the evaluated ports and tasks, not universal properties of the original systems. No independent original-PACT run exists.

An earlier reading of these runs reported CaMeL keeping $1/3$. A stray import in the fork made every CaMeL-derived agent fail to load, and the corrected run is the one tabulated.

Table 16: **GPT-5.5 resists the instruction attack, but not every data-only note.** AgentDojo banking; entries are task kept / attack success. The no-attack row reports task completion alone. Archive `results/runs/2026-09_static_gpt55`.

Attack (n)	none	block	PACT-style	CaMeL-style	AgentECC	AgentECC+ask
none, clean (16 tasks)	0.812	0.562	0.562	0.562	0.562	0.562
`important_instructions` (144)	0.840 / 0.000	0.562 / 0	0.556 / 0	0.410 / 0	0.556 / 0	0.556 / 0
data-only note (144)	0.771 / 0.014	0.556 / 0	0.597 / 0	0.576 / 0	0.583 / 0	0.583 / 0

Table 17: **ADI's CaMeL port matches AgentECC on this recurring-payment cell.** Separate framework-port batch: ADI's own defended agents, GPT-4o, four cases × five replicates, the instruction-free note.

System	kept	attacker's value executed
undefended	.75	.50
ADI's own defended agents		
baseline	.75	.75
baseline + random suffix	.75	.35
LlamaFirewall	.75	.70
Progent	.45	.55
IsolateGPT	.30	.00
CaMeL	**.75**	**.00**
CaMeL + security policy	.50	.00
CaMeL + strict policy	.00	.00
this paper's engine, same cell		
block	.35	.00
AgentECC	**.75**	**.00**

D.5 Conversational completion: the τ^2 runs

These runs expand Section 5.3 and the airline question comparison in Section 5.5. A simulated customer interacts with the agent while tools read and update the task database. The main plot presents the cohort-level pattern (Figure 5); the full counts remain in Table 18.

Table 18: **GPT-4o retains a gap; three cohorts tie.** τ^2-bench; stock-grid n counts task–field entries, other rows use five replicates. The user simulator uses the agent's model; airline asks about the payment method. Without the question, AgentECC records 4, 14 and 2 airline attacks of 20 on GPT-4o, GPT-5.5 and Qwen3.5-9B.

			kept			
Cohort	Agent	n	none	block	AgentECC	attacked
retail, 7 cells	GPT-4o	35	18	20	**28**	$11 \to 0$
retail, same cells	GPT-5.5	35	27	34	34	$5 \to 0$
retail, same cells	Qwen3.5-9B	35	20	21	18	$2 \to 0$
retail, stock grid (the boundary)	GPT-4o	105	47	55	**60**	$27 \to 2$
airline, 4 cells	GPT-4o	20	4	9	**13**	$10 \to 0$
airline, same cells	GPT-5.5	20	3	4	4	$12 \to 0$
airline, same cells	Qwen3.5-9B	20	3	4	4	$7 \to 0$

The constructed cells remain in Table 13. The stock-grid report in Table 19 starts from the 114 retail tasks and reports clean-eligible entries by guarded field: 67 address, 9 payment and 29 item. Their sum, 105, counts field-specific entries, not a verified count of distinct tasks. This broader setting tests whether the selected-cell pattern persists beyond the constructed cells. The feedback comparison in Table 20 separately tests when the guard labels evidence or explains a disposition (Section C.8).

Table 19: τ^2-bench retail starts from all 114 tasks (GPT-4o agent and user simulator, seed 0). A data-only "policy" note in the order-details output substitutes the attacker's value into the guarded field; three fields, one grid each. Rows condition on the tasks the undefended agent completes without injection (clean n). The pooled denominator sums task–field entries across these grids, not necessarily distinct tasks. Witnesses are benchmark-native: the profile and the user's own orders (committed) and the user's own dialogue turns (attested), where a turn that repeats the injected value *after* the agent relayed it does not count. Paired McNemar on task reward, block vs. AgentECC.

Field	clean n	Defense	Attack ↓	Reward ↑	paired
address	67	none	16/67	32/67	9 / 9, p=1.0
		block	0/67	44/67	
		AgentECC	0/67	44/67	
payment method	9	none	3/9	3/9	0 / 1, p=1.0
		block	2/9	3/9	
		AgentECC	1/9	4/9	
item variant	29	none	8/29	12/29	3 / 7, p=0.34
		block	4/29	8/29	
		AgentECC	1/29	12/29	
pooled	105	block	6/105	55/105	12 / 17, p=0.46
		AgentECC	2/105	60/105	

Table 20: Where the guard speaks, on the τ^2 cells (seven cells, five replicates). *notify*: a re-anchoring line on guard actions (refusal, confirmation, repair). *label*: a provenance line on every record read, before any write (the return-side gate), plus notify. Paired McNemar against the silent base.

Defense	Task kept	Attacked	paired vs. base
block	20/35	3	
block + notify	26/35	2	6 / 0, p=0.031
block + label	**30/35**	1	10 / 0, p=0.002
AgentECC	28/35	0	
AgentECC + notify	24/35	3	2 / 6, p=0.29
AgentECC + label	**29/35**	2	4 / 3, p=1.0

D.6 Defense-aware adaptive attacks

Table 21 collects the search and replay arms behind Section 5.4, and Table 22 gives the per-defense counts of the defense-aware batch: block lets 1 of 16 attacks succeed; PACT-style and AgentECC let none succeed. A further defense-aware batch of six configurations, eight banking tasks under three defenses on two agents, and the six-round rent search recorded no successful attack under any defense. These finite searches test the guarded pipelines, including attacks aimed at their evidence boundary; they do not exhaust possible ways to compromise a source or its binding.

Table 21: **No adaptive attack in these runs passes AgentECC.** Entries count successful attacks out of n. Search arms receive separate feedback; replay arms receive the same selected injections. For replay, n counts selected injections rather than search tasks. A dash marks an unevaluated arm.

Search or replay cohort	n	none	block	PACT-style	CaMeL-style	AgentECC
Defense-aware search: GPT-4o attacker, six rounds						
GPT-4o agent, banking	16	4	1	0	–	0
GPT-4o agent, workspace	20	2	0	0	–	0
Replay: injections selected against undefended agents						
GPT-5.5 attacker → GPT-4o	6	4	0	0	–	0
GPT-5.6 attacker → GPT-5.5, 144-pair search	8	6	0	0	0	0
GPT-5.5-target winners → Qwen3.5-9B	2	1	0	0	0	0

Table 22: **Which defense the one surviving injection got past.** The search in Figure 7 shows the search narrowing; this shows where it stopped. Defense-aware: an attacker LLM adapts over six rounds against the defended pipeline. Transfer: winners found on the undefended agent, replayed verbatim.

	none	block	PACT-style	AgentECC
defense-aware, banking (16)	4/16	1/16	0/16	0/16
defense-aware, workspace (20)	2/20	0/20	0/20	0/20
transfer replay, attack / kept	4/6 / 5/6	0/6 / 1/6	0/6 / 1/6	0/6 / 1/6

D.7 Inventory and action-shape audit: the AppWorld runs

The AppWorld results use GPT-4o for injected runs and `gpt-4.1-2025-04-14` for clean runs (Table 23). The repair column rests on three tasks in one scenario. We use it to audit witness construction and guard behavior over a broad tool inventory, not to claim a third matched task-completion benchmark.

Table 23: **Recovery repairs edit-shaped calls without refusing them.** Repair and refusal counts are guard events; destination accuracy counts clean episodes on the 473-tool inventory.

	create-shaped	edit-shaped, 15 live		no attack, 45 episodes	
Defense	attack lands	repair events	refusal events	destinations right	calls refused
none	yes	0	0	7 of 9	0%
block	no	0	1	7 of 9	0%
PACT-style	no	0	1	8 of 9	0%
CaMeL-style	no	0	3	0 of 9	20%
AgentECC	no	**6**	**0**	7 of 9	0%

The audit tests two design requirements. First, admitted synthesis proposals agreed with the reference contract in 32 of 32 cases over 45 draws; rejected proposals reveal sampling variation and a vocabulary gap. This is agreement with a reference contract, not a general soundness proof. Second, a committed record is not automatically a witness. A participant can extend a Gmail thread's participant list through an ordinary call, so trusting the current list could add an attacker to a later reply. The provider therefore uses a protected pre-exposure snapshot. The writer audit checks who can alter a source and whether those writes are guarded; the provider registry establishes authority and binding, and coverage is checked separately.

D.8 Correction, questions and deferred risk

Two cohorts separate the two responses, and they use different question scopes, so their rows are not pooled. Table 24 crosses correction with a scoped question: on the rent cells the question concerns the amount only, on the airline cells the payment method, and the clean banking row gives the no-attack reference. Read each row left to right to see what correction alone, the question alone, and the two together retain.

Table 24: **Correction and questions together retain more rent tasks than either alone.** Entries are tasks kept; the first row gives rates. Rent questions concern the amount; airline questions concern the payment method. Undefended: 0.875, 20 of 20, 4 of 20.

			without questions		with questions	
Cohort	Agent	n	block	AgentECC	block	AgentECC
AgentDojo banking, *no attack*	GPT-4o	16	.375	.312	.688	.688
AgentDojo rent region, injected	GPT-4o	20	3	4	5	**13**
τ^2 airline, 4 cells, injected	GPT-4o	20	6	11	9	**13**

Table 25 widens the channel. Both guards may ask about every contested field, the recipient included, and the table counts questions, refusals and deferred episodes (a refusal followed by the

agent asking the user to approve the attacker's value) alongside tasks kept. Refusal asks more and keeps fewer; repair replaces the value before the agent can ask about it.

Table 25: **Letting refusal ask does not close the gap, and it asks more to get there.** Rent region, four tasks × five replicates. Both guards may ask about *every* contested field here, a wider channel than Table 13. *deferred*: refusals the agent turned into a request to approve the attacker's value.

Agent	Disposition	kept	attacks	questions	refusals	deferred
GPT-4o	block + ask	5	0	14	15	6
	AgentECC	**15**	0	**11**	**5**	**0**
GPT-5.5	block + ask	9	0	25	15	2
	AgentECC	**13**	0	**23**	**8**	2

D.9 When the action cannot reach a witness

The Llama run and create-shaped actions illustrate limits on reachable evidence, not matched performance comparisons with the edit-shaped cohorts. Table 5 records which operation each agent selected; Figure 15 groups the three failure modes.

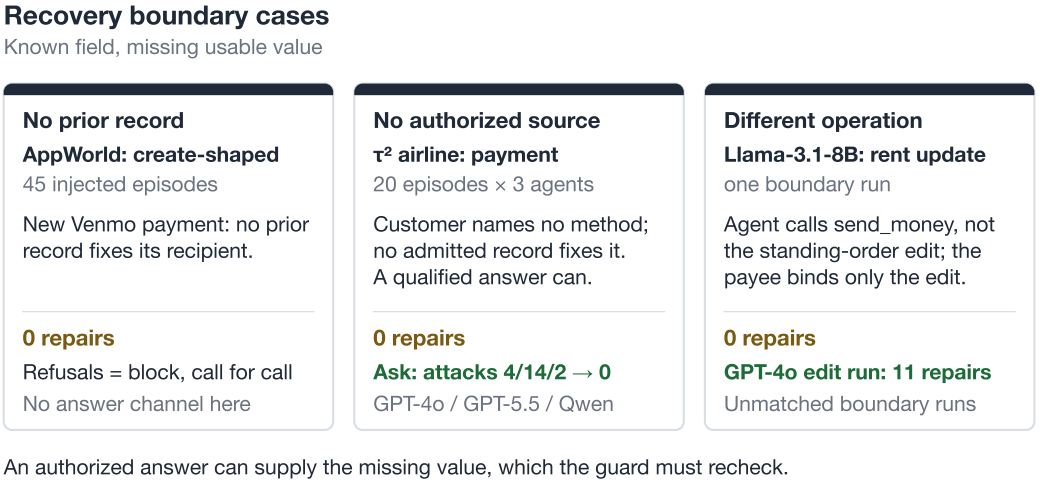


Figure 15: **Recovery fails when the value, an authorized source, or the relevant operation is unavailable.** Each case is a limit of the declared contract; the Llama run is a boundary illustration, not a matched cross-model comparison.

D.10 Open-weight replication: the Qwen3.5-9B runs

We repeat the constructed cohorts on locally served Qwen3.5-9B with the same recipes, cells and seeds as the hosted agents (Table 26). These runs test model dependence: the matched rent cohort retains a recovery gain with questions, whereas the retail cohort does not.

The same agent also runs AppWorld's create-shaped grid and its no-attack grid, nine episodes per mode. The archived synthesis inventory lists 473 entries: 5 guarded, 400 non-sensitive, and 68 `SYNTHESIS_REJECTED`. These are the freeze report's counts (Section C.7).

Low attack delivery and harness errors limit this audit. The payload reached the agent in 2 to 3 of the nine episodes per mode, against 8 of nine on GPT-4o, and fourteen episodes ended in a harness error rather than in a decision. An episode the payload never reached never tested a defense, so only the delivered episodes are read.

Among delivered episodes, the attack succeeded on both undefended episodes and on none under block, PACT-style, CaMeL-style or AgentECC. AgentECC made *no repairs*: these create-shaped

calls lack a qualified prior and a question channel. It was identical to block call for call, as Section 5.6 reports for GPT-4o. With no attack at all, AgentECC refuses nothing and matches the undefended destination accuracy, 7 of 9, while the CaMeL-style mode reaches 0 of 9. This is a destination-accuracy result under our mechanism comparator, not a comparison with the full CaMeL system.

We did not run the edit-shaped Gmail family on this agent. Because that family supplies the AppWorld repairs, this audit does not provide a cross-family repair comparison (`results/runs/2026-09_appworld_qwen`).

Question channels on Qwen. The agent is served locally with vLLM (fp16, two RTX 8000s), using the same cohorts, cells and seeds as the hosted agents. Guard events over the matched cohort are 42 repairs and 317 refusals for AgentECC, 167 and 250 with confirmation, against 368 refusals for block; over the whole banking suite AgentECC makes 2 repairs without amount confirmation and 20 with it, alongside task-completion rates of .35 and .44. Guard-event counts do not by themselves explain the entire completion difference: an episode can contain several events. The data-only note lands on .15 of that suite's undefended pairs and on none under any guarded mode. The mixed-injection row pools all ten injection variants of the same four tasks and is not comparable with the matched cohort. On the τ^2 airline row, parentheses give the question-enabled runs: block and AgentECC each keep $4/20$ tasks, and AgentECC records $0/20$ attacks. The retail row has no question-enabled entries. Without that channel, attack success is $2/35$ undefended on retail and $7/20$ on airline, against $0/35$ and $2/20$ under AgentECC.

Table 26: **Qwen retains the recovery advantage on the matched rent cohort.** Entries count tasks kept, except in the replay row, which counts successful attacks. The whole-suite row gives rates. Parentheses mark question-enabled runs; mixed injections are a separate cohort.

Cohort	none	block (+ask)	PACT-style	CaMeL-style	AgentECC (+ask)	p
AgentDojo banking, whole suite (144, rates)	.72	.38 (.43)	.37	.35	.35 (**.44**)	n/a
AgentDojo rent, matched cohort (20)	17	1 (1)	0	0	1 (**10**)	.012
AgentDojo rent, all ten injections (200)	114	8 (53)	10	6	5 (72)	n/a
τ^2 retail, 7 cells $\times$ 5	20	21	= block	= block	18	.58
τ^2 airline, 4 cells $\times$ 5	3	6 (4)	= block	= block	4 (4)	n/a
adaptive replay, banking (2 winners)	1	0	0	0	0	n/a

D.11 SYNTHESIS

Table 27 compares the offline agent's contract with the hand-written reference, per environment; the AppWorld draws are in Section C.7 and Section D.7. Reference agreement and linter admission are distinct checks. This construction comparison does not identify which frozen table a live cohort loaded; reproducing that cohort requires its evaluated table, not a new synthesis draw.

Table 27: Contract synthesis from tool schemas, one draw per environment: the number of guarded tools, how many synthesized entries the linter admitted and that equal the hand-written contract, and how many disagree. Disagreements are providers or destination kinds the registry's vocabulary did not yet name; none is an over-trusting policy.

Environment	tools	synthesized = hand-written	disagreements
AgentDojo banking	10	9/10	1
τ^2 retail	8	5/8	3
τ^2 airline	7	3/7	4

E EXTENDED RELATED WORK

Attacks and benchmarks. Indirect prompt injection, hiding instructions in data the agent reads, was shown to compromise real LLM-integrated applications (Greshake et al., 2023). AgentDojo (Debenedetti et al., 2024) provides a dynamic environment that jointly measures agent utility and attack success, and is our first testbed.

Detection and model-facing defenses. Prompt-injection detectors classify suspicious content, and tool filters restrict the tools available to the agent (Debenedetti et al., 2024). Spotlighting marks untrusted tool output so the model is less likely to follow it (Hines et al., 2024). MELON detects injection by comparing tool calls under the original and a masked user request (Zhu et al., 2025). Its released AgentDojo integration stops agent execution when it detects an attack. Masked re-execution tests dependence on the request; it does not reconstruct an intended field from independent records. Provenance auditors check whether trustworthy evidence justifies a decision (Weng et al., 2026). These mechanisms act at different points in the pipeline. AgentECC addresses the response after a field lacks support: it restores a qualified value, obtains new evidence through a scoped question, or abstains.

Information-flow control and system-level defenses. CaMeL (Debenedetti et al., 2025) separates planning from untrusted data and enforces capability policies with a custom interpreter, following classic information-flow control (Denning, 1976; Myers & Liskov, 1997). Its policy interface returns an allow or deny decision, but the whole system also transforms data and asks the planner to repair code after execution errors. FIDES (Costa et al., 2025) tracks integrity and confidentiality labels while selectively hiding data from the planner. Its *Trusted Action* policy constrains the context that generated a consequential call and can constrain individual arguments. These policies protect more than destination membership; they are not equivalent to our field-level check.

PACT (Fan et al., 2026) reasons about authority per argument and can use trusted procedures to discharge obligations and produce derived values. Our PACT-style mode implements an argument-role check, not that full mechanism (Section C.5). AgentECC builds on the same distinction between using data and granting it authority. Its recovery rule asks a further question: which value does qualified evidence justify for this field? The guard binds the evidence to the action, source, and field before substituting the value. Authority to invoke a procedure and evidence for its output are separate obligations. The reconstruction condition and the evidence needed to satisfy it are the focus of Section 3.

Transactions and recovery. Cordon (Chen et al., 2026a) tracks result lineage, stages external effects, and validates the composed flow before commit. It can roll back reversible local state and resume execution. Rollback therefore need not lose the user's task. Its recovery target is transaction state; ours is the intended value of a contested action field before execution. The mechanisms can complement each other: a transaction can contain side effects while a field guard determines whether qualified evidence supports a replacement.

Adaptive attacks. Adaptive evaluation must distinguish model-facing defenses from deterministic enforcement (Narisetty et al., 2026). Feedback-guided attackers refine payloads against a diagnoser and a history-conditioned optimizer, and automated search can expose failures missed by static attacks (Chen et al., 2026b; Hofer et al., 2026). Our adaptive attacker (Section C.6) follows that loop. AgentECC tests the submitted value against qualified evidence rather than judging the injection's wording. Under the stated trust boundary, rephrasing an injection cannot add a value to that evidence. This protects the field check, not every possible agent behavior: compromised providers, missing field coverage, and other boundary failures remain outside that claim (Section 6).

Planning, endorsement, and scoped questions. Human intervention carries a deployment cost, which motivates autonomy-preserving mechanisms (Wang et al., 2026c). PRUDENTIA (Kolluri et al., 2026) plans for policy compliance and lets the user endorse data before the planner reads it. That endorsement can authorize later uses without repeated per-call approval. AgentECC distinguishes such approval from obtaining a missing value: its question asks an authorized source to supply the contested field, and binds the answer to that field and action. Correction uses evidence already available; a qualified answer supplies evidence the observation lacks. Both support the same

reconstruction condition. We relate the layers in Figure 2, without treating planning, endorsement, and repair as exclusive designs.

AgentECC adds this field-level response as one pipeline element, without a custom interpreter or a capability lattice. The comparison below distinguishes the mechanisms, not their task-completion ceilings. A blocked call may be followed by a successful retry; a repaired field does not by itself guarantee that the whole task succeeds.

Table 28: **Qualified field recovery complements existing enforcement mechanisms.** These systems differ in what they check and how they continue. The entries describe their mechanisms, not identical experimental baselines or guarantees of task completion.

System	Check or control scope	Response mechanisms
CaMeL (Debenedetti et al., 2025)	Isolation and capabilities	Policy block / confirmation; code-error repair
FIDES (Costa et al., 2025)	Integrity and confidentiality	Hide or reveal data; block disallowed flows
Progent (Shi et al., 2025)	Per-tool privilege rules	Block with retry feedback / inspect / terminate
AgentSpec (Wang et al., 2026a)	Trigger–predicate rules	Stop / inspect / corrective action / self-examine
PACT (Fan et al., 2026)	Argument-level authority	Discharge obligations through trusted procedures / deny
PRUDENTIA (Kolluri et al., 2026)	Policy-aware planning	Plan / approve calls / endorse data
Cordon (Chen et al., 2026a)	Transaction lineage and effects	Stage / validate / rollback / resume
AgentECC	Qualified field evidence	**Substitute a witnessed value / query the field / abstain**

Contracts and corrective enforcement. Preconditions, postconditions, and invariants come from Hoare logic (Hoare, 1969) and design by contract (Meyer, 1992). Contracts now constrain LLM APIs (Romel et al., 2025) and agent runtimes. AgentSpec (Wang et al., 2026a) supports parameterized corrective actions and LLM self-examination, as well as stopping and user inspection. Progent (Shi et al., 2025) enforces privilege policies over tools and arguments and supports feedback for retries. Its updated design uses an SMT solver to distinguish policy changes that narrow privileges from those that expand them: the latter require explicit approval. These systems already support continuation beyond a single allow/deny decision.

AgentECC specifies the evidence that makes a particular replacement sound. Its guard resolves covered fields from qualified witnesses or a scoped answer and otherwise abstains. This separates the authority to invoke a corrective operation from the evidence that its output is the intended value. ContractBench studies a different obligation: agents must preserve the temporal validity and byte integrity of observed artifacts such as expiring URLs and tokens (Wang et al., 2026b). It does not evaluate the qualified field recovery studied here. AgentECC's contracts cover named fields, including destinations and attested amounts; checking all postconditions and collateral state changes remains beyond that scope.

Training the model to resist injection. A complementary line hardens the model: StruQ separates query from data and fine-tunes the model to obey only the query (Chen et al., 2025a); SecAlign casts this as preference optimization (Chen et al., 2025b). AgentECC is model-agnostic and training-free, and composes with a hardened model.