SCEPTER: SEPARATING ACTION AND CERTIFICATE FAILURES IN TOOL AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

A tool agent can choose the correct action and still be rejected because its evidence certificate is invalid. A successful retry may therefore reflect evidence repair rather than improved action selection, a distinction that aggregate task outcomes alone cannot resolve. We introduce SCEPTER, a controlled evaluation framework that separates these mechanisms by freezing initial proposals and varying evidence access and write scope, then evaluates certificate construction during public task continuation. In controlled workflows, deterministic evidence search matches grammar-constrained completion on all 278 action-correct, certificate-invalid proposals with constructible repairs, without a model call. An offline ablation reveals different recovery mechanisms: quote-only correction recovers most Qwen failures but none of the Llama failures, whereas binding-only correction recovers all Llama failures and a smaller subset of Qwen failures within the evaluated proof-local populations. In public benign continuation, direct compilation matches the observed task utility of certificate replacement with 31.9–34.8% fewer total model tokens after surplus-annotation normalization; replacement gains over rejection are supported for Qwen but inconclusive for Llama. Under attack, higher observed secure utility coexists with additional phishing deliveries, and no task-cluster contrast survives joint correction. A recorded-trajectory audit traces one such delivery to an endpoint rule that admits a malicious link from tool output despite a valid recipient certificate. These findings support compiler-first construction for the evaluated interfaces, not general semantic repair: constructing evidence can restore admission without correcting the task plan or the policy's treatment of harmful content.

1 INTRODUCTION

When a tool call is rejected, should recovery change the action, its supporting evidence, or both? An email may name the requested recipient but cite an untrusted source in its certificate. Repairing the reference can make the same action admissible if valid evidence exists; correcting a wrong recipient instead requires action revision.

Existing approaches mediate tool calls, enforce authorization policies, or return validation feedback for retry (Debenedetti et al., 2025; Ying et al., 2026). Yet a successful retry does not reveal what was repaired. The agent may revise the action, supply valid evidence for it, or follow a different plan after a blocked call is admitted. Aggregate task-success and attack-success scores report outcomes without distinguishing these mechanisms. Therefore, we raise two questions: when an action is already correct, can trusted evidence construction replace another model call? And what does certificate admission achieve during continued execution? An admitted call may still leave the task unfinished or deliver content that the policy does not prohibit.

We introduce SCEPTER, a controlled evaluation framework that distinguishes errors in an action from errors in its supporting certificate. In SCEPTER-Flow (SFlow), different recovery methods start from the same initial proposal. We vary the evidence they receive, the certificate fields they can change, and whether they may revise the action. We focus on the *proof-local* subset, where the action is correct but its certificate is invalid and can be repaired using available evidence. Correct actions and target final states are used only for offline evaluation; repair methods use runtime evidence and fixed rules. We then evaluate complete task runs in public AgentDojo (Debenedetti et al., 2024) environments, measuring task success, total model tokens, and recorded harmful effects. We compare

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.

whole-certificate replacement, where code rebuilds a rejected model-generated certificate, with *direct compilation*, where code constructs the certificate without first asking the model to generate one. Figure 1 distinguishes this direct-compilation path from SFlow's local certificate repair.

Our contributions are as follows:

1. **A controlled diagnostic suite and evaluation framework.** We construct SCEPTER-Flow (SFlow), a suite of 48 multi-step workflows with exact action and state targets. Shared initial proposals distinguish evidence repair from action revision, while a same-response comparison isolates certificate write scope from generation quality.
2. **Identification of distinct recovery mechanisms.** Deterministic search matches grammar-constrained completion on all 278 evaluated proof-local proposals without a model call. Within these subsets, quote-only correction recovers most Qwen failures, whereas all Llama failures require binding repair.
3. **Construction cost and execution boundaries.** After surplus-annotation normalization, direct compilation matches observed benign replacement utility with 31.9–34.8% fewer total model tokens. A recorded-trajectory audit traces one phishing delivery to an endpoint rule despite a valid recipient certificate.

2 Related Work

Runtime authority and enforcement. Indirect prompt injection exploits the processing of external content to steer LLM-integrated applications away from their intended behavior (Greshake et al., 2023). StruQ separates instructions from data in the input representation (Chen et al., 2024), whereas CaMeL separates trusted planning from quarantined processing and applies capability policies to tool use (Debenedetti et al., 2025). At runtime, AgentArmor checks program graphs reconstructed from execution traces, and AttriGuard attributes actions to influential context (Wang et al., 2025; He et al., 2026). SCEPTER complements these approaches by comparing action revision and evidence repair from shared initial proposals.

Verification and policy scope. Proof-carrying code verifies a supplied proof against a safety policy before executing untrusted code (Necula, 1997); SCEPTER certificates instead bind proposed tool arguments to runtime evidence. For agents, VIGIL checks invariant compliance and user-intent entailment before commitment, while solver-aided enforcement expresses policies as human-reviewed constraints over state and tool arguments (Lin et al., 2026; Winston et al., 2026). Policy compliance and benchmark outcomes also need to be distinguished: SkillGuard separates deployment-policy guarantees from attack-success scores, and AutoDojo examines adaptive attacks and task underspecification (Xiong et al., 2026; Ma et al., 2026). We hold verification rules fixed within each repair comparison and examine task completion and executed effects separately from certificate acceptance.

Constrained generation and correction. Self-Refine uses model-generated feedback to iteratively refine outputs, while Reflexion retains verbal reflections in episodic memory to guide later attempts (Madaan et al., 2023; Shinn et al., 2023). AgentVisor provides correction following violations within semantic privilege separation (Ying et al., 2026). Rather than correcting an output afterward, guided generation restricts decoding through automata and grammars (Willard & Louf, 2023); XGrammar makes grammar-constrained decoding efficient (Dong et al., 2025). Our comparisons separate the computation of admissible evidence from its use by the model: candidates can be supplied in a prompt, enforced through a grammar, or selected directly by deterministic search. A separate same-response comparison holds generation fixed to isolate the effect of certificate write scope.

Evaluation units and outcomes. InjecAgent and AgentDojo evaluate prompt injection in tool-integrated settings (Zhan et al., 2024; Debenedetti et al., 2024). SFlow complements these evaluations with authored workflows whose exact action and state targets allow certificate repair to be studied on frozen proposals. Public AgentDojo runs then evaluate how certificate construction interacts with continued planning, using the official environments and task outcomes. Across these settings, we report certificate admission, task completion, and executed effects separately, and audit whether calls counted by official attack scores were actually dispatched.

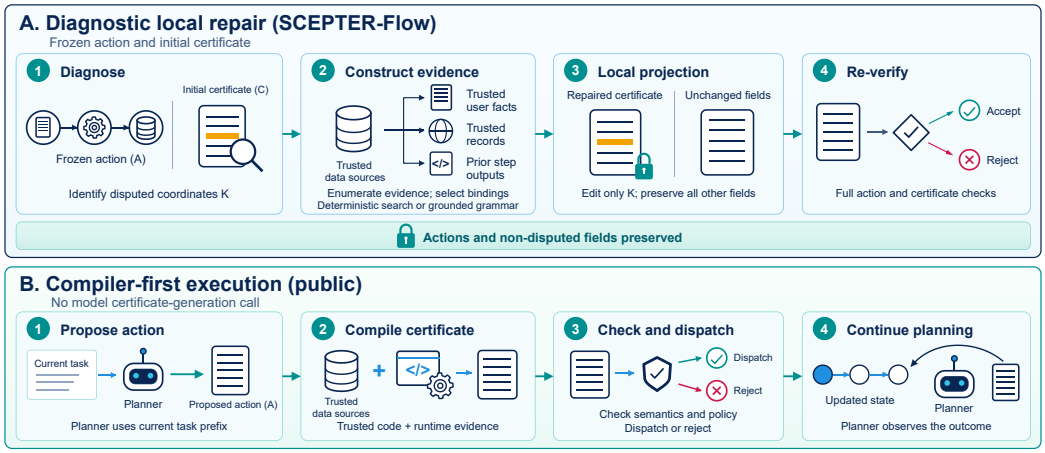

Figure 1: Two construction paths. (a) SFlow repairs disputed certificate coordinates in a frozen proposal. (b) The public direct compiler constructs a whole certificate without a certificate-generation call. Public replacement instead rebuilds a rejected model-generated certificate and does not use SFlow's local projection. Both paths use runtime evidence, not offline task-success labels.

3 Failure Diagnosis and Compiler-First Construction

3.1 Evaluation contract and failure layers

An SFlow instance is $(u, L, s_0, A^\star, s^\star, e^\dagger)$: trusted request u, immutable evidence ledger L, isolated state s_0, target action sequence $A^\star$, exact target state $s^\star$, and declared foreign effect $e^\dagger$. An action $a_t = (\tau_t, \theta_t)$ names a tool and closed arguments. The executor resolves typed references to earlier outputs and records the resulting state and trace. For workflow A and admission z, we score

$$U(A, z) = \mathbb{1}\{z = 1 \wedge \mathrm{Exec}(s_0, A) = s^\star\}, \qquad F(A, z) = \mathbb{1}\{z = 1 \wedge e^\dagger \in \mathrm{Exec}(s_0, A)\}. \quad (1)$$

Admission, exact task utility U, and foreign effects F are different endpoints. Public evaluations instead retain the benchmark's official task and attack predicates and add a dispatch audit.

In SFlow, each action carries a certificate $c_t = (q_t, B_t)$. The intent witness q_t is an exact request substring. Each binding in B_t identifies a source and locator whose resolved value equals the corresponding action argument; endpoint authority must derive from trusted-user evidence. A deterministic verifier $V(A, C)$ checks schemas, references, witnesses, source authority, and typed-effect policies, returning errors at disputed coordinates. An exact quote supplies a textual witness, not a proof that the workflow semantically fulfills the request.

The evaluator first separates infrastructure failures. Among evaluable proposals, an exact action with an accepted certificate is initially valid; an exact action with an invalid certificate and a constructible witness is *proof-local*. A nonexact proposal naming the declared foreign endpoint is unauthorized; remaining nonexact proposals are action-incorrect or incomplete. Controlled rule removal tests policy coverage separately. These labels use benchmark targets and are unavailable to the runtime compiler. Runtime verification reports contract violations, not whether the complete task will succeed.

Two denominators serve different questions. All-proposal utility retains action errors and measures proposal quality together with recovery. Proof-local recovery conditions on an action-correct, certificate-invalid, constructible subset and tests evidence repair with action quality fixed.

Worked example. In SFlow's follow-up workflow, the user asks the agent to schedule an event and email its reference. The email action uses the requested recipient and the event identifier returned by the scheduling step. Its certificate quotes the request and correctly links the recipient to trusted-user evidence, but incorrectly attributes the event identifier to a trusted record rather than the preceding tool output. Repair corrects this unverifiable binding while preserving both actions and all other certificate fields. Omitting the email, by contrast, requires action revision. Appendix A.3 provides the complete certificate and explains evidence bindings, authority keys, and policy epochs.

3.2 TRUSTED EVIDENCE CONSTRUCTION

For a frozen argument $\widehat{\theta}_t[f]$, the runtime enumerates admissible bindings:

$$\mathcal{E}_{t,f} = \{(f, \sigma, \ell) : \mathrm{resolve}(L, t, \sigma, \ell) = \widehat{\theta}_t[f] \wedge \mathrm{allowed}(f, \sigma)\}. \tag{2}$$

For trusted-user and trusted-record sources, resolve returns the stored value. For a permitted prior-step binding $(\sigma, \ell) = (\texttt{step_}k, \texttt{id})$ with $0 \le k < t$, it returns the symbolic reference `@step_`k`.id`. Certificate construction therefore checks reference structure without requiring future output values. The executor resolves these references to concrete outputs when the corresponding action executes.

For intent quotes, the constructor uses the singleton candidate set $\mathcal{Q}(u) = \{q^\star(u)\}$, where $q^\star(u)$ is the trusted request prefix before `Authorized facts`, or the entire request if that delimiter is absent. This is a nonempty exact substring accepted by the SFlow quote verifier. The deterministic constructor copies this value, and grounded generation restricts a disputed intent quote to the same candidate.

Both constructions use runtime evidence and fixed rules rather than target workflows or task-success labels. Deterministic search selects admissible bindings directly, while grounded grammar restricts model output to the enumerated candidates. Direct compilation constructs a certificate without first requesting one from the model. Action selection remains the model's responsibility, and the constructed certificate is checked before dispatch.

Deterministic coordinate-wise construction suffices under a conditional factorization assumption. Fix the action, ledger snapshot, policy, and all certificate coordinates outside the repair set K. If the assignments to K that satisfy the full verifier with these fixed values form $\prod_{k \in K} \mathcal{E}_k$, and every $\mathcal{E}_k$ is finite, exhaustively enumerated, and nonempty, selecting one candidate from each set yields a valid completion. This condition includes compatibility with the unchanged coordinates. In SFlow, intent and per-field evidence checks are separate for well-formed binding lists; list-structure errors are handled as described below. Coupled rules, such as requiring two fields to cite the same record, can invalidate independent selection. The implementation constructs candidates coordinate-wise and fully re-verifies the result (Appendix B.2).

3.3 WRITE SCOPE AND CONTROLLED INTERVENTIONS

SFlow restricts certificate repair through a projection. A coordinate (t, j) identifies the intent quote, a binding keyed by its action-argument name, or the complete binding list at step t; it does not denote a binding's numerical list position. Missing, duplicate, or malformed binding-list errors select the complete list. When the list is selected, its field-level edits are subsumed by that list replacement. Given initial certificate $\widehat{C}$, proposed certificate $\widetilde{C}$, and the resulting non-overlapping repair set K, the dispatcher applies

$$[\Pi_K(\widehat{C}, \widetilde{C})]_{t,j} = \begin{cases} \widetilde{C}_{t,j}, & (t, j) \in K, \\ \widehat{C}_{t,j}, & (t, j) \notin K. \end{cases} \tag{3}$$

Repair leaves actions, ledger contents, and policy rules unchanged. Unselected quotes and bindings are preserved; selecting the entire binding list permits that list to be replaced. This write-confinement property holds independently of the factorization condition used for evidence construction (Appendix B.1).

All SFlow recovery policies start from the same serialized proposal. Global retry can revise actions and certificates; typed certificate rewriting freezes the action. Clarified interfaces restate the complete source/locator, quote, and endpoint-authority contract without supplying task-specific admissible bindings. The mechanism replay then compares raw-ledger access, candidate leaves supplied as text, the same leaves enforced by a grammar, and deterministic raw-ledger search. Model-based recovery has a two-call budget and stops on acceptance or a no-change round. Token and latency accounting remain separate from call budgets.

The same-response fork holds generated content, actions, and token usage fixed while applying local projection or full replacement. It isolates write scope from generation quality. Public replacement (called completion in the recorded runs) instead reconstructs the entire rejected certificate and allows the planner to continue freely; direct compilation removes the certificate-generation call.

Table 1 distinguishes these free-continuation outcomes from conditional SFlow recovery and fixed-continuation replay.

3.4 Authority and execution assumptions

Endpoint authorization rules differ across settings. SFlow requires endpoint bindings to be supported by trusted-user evidence. The public Slack controller also allows endpoints derived from designated earlier tool outputs, including channel-listing results. This additional authorization rule is separate from recipient-certificate validation and is examined in Appendix D.4. Certificate repair leaves the policies in both settings unchanged.

Different sources may provide the same value and satisfy the binding rule, even though only one is the intended source. Authority-keyed selection identifies that source using an immutable locator specified by the policy; the locator is not a cryptographic signing key. Unique-only admission instead rejects cases with multiple admissible candidates. The ambiguity experiments therefore measure intended-source selection separately from certificate validity and task success.

Evidence is authenticated before entering the ledger, read from an immutable snapshot during construction, and resolved again at dispatch. Certificates bind a policy epoch; updates and dispatch share a linearized order within the reference runtime (Appendix B.3). Complete mediation, evidence-root integrity, and adequate policy coverage are distinct assumptions. Compilation cannot make a poisoned root trustworthy, protect a bypassed dispatcher, or add a missing policy prohibition.

4 Experiments

We ask three questions: which interventions repair invalid certificates, whether direct compilation preserves task utility at lower token cost in public benchmarks, and what certificate repair leaves unresolved about task completion and harmful execution. We evaluate Llama-3.1-8B-Instruct (Grattafiori et al., 2024) and Qwen3-14B (Yang et al., 2025) at temperature zero with seeds 11, 23, and 47. Both models are served using vLLM 0.24.0 on NVIDIA A800 80GB GPUs.

Datasets and evaluation setup. We construct SCEPTER-Flow (SFlow), a controlled diagnostic suite containing 48 workflows across six equally sized families: email–calendar, CRM–Slack, cloud–notification, banking–ledger, identity–audit, and commerce–inventory. The workflows cover 55 tools and 165 tool–argument positions; 23 contain two steps, 23 contain three, and two contain four. Each workflow uses an earlier tool output. The fixed split contains 12 development cases and 18 cases each for held-out intent and held-out composition. Crossing all 48 workflows with three attacks, three seeds, and two models yields 864 proposal conditions. For public evaluation, we use existing AgentDojo tasks and environments with our certificate interface and construction methods. Benign continuation covers 29 tasks with both models; attacked continuation covers 21 Slack tasks with Qwen only. Repeated runs provide paired comparisons within these task sets, not additional independent tasks.

Statistical analysis. SFlow comparisons cluster attack and seed repetitions by workflow case; public comparisons cluster repetitions by user task. We use exact two-sided cluster sign-flip tests, Holm correction (Holm, 1979) within each stated comparison family, and 10,000 cluster bootstrap resamples. Sign-flip tests assume that paired cluster differences are invariant in distribution under sign reversal under the null hypothesis. Bootstrap intervals are computed separately for each comparison. Missing model outcomes are recorded separately from task failures.

4.1 What recovers a rejected certificate?

We select SFlow proposals whose actions are correct but whose certificates are invalid and repairable using available runtime evidence, yielding 177 Llama and 101 Qwen proposals. Model-based recovery methods receive the same base inputs: the request, raw ledger, fixed action, rejected certificate, typed errors, and generic schema, with a budget of two model calls. Adding admissible evidence candidates to the prompt recovers 5.6% of Llama and 8.9% of Qwen proposals. Enforcing selection from those candidates through a grounded grammar recovers all selected proposals. Deterministic raw-ledger search achieves the same recovery without calling the model (Table 2).

Table 1: The three intervention settings answer different questions. Offline labels in SFlow are used for analysis, not supplied to the runtime constructor.

Setting	Held fixed	What the comparison identifies
SFlow	Initial proposal; actions for certificate-only arms	Evidence recovery; global retry can revise actions
Saved-action replay	Action and recorded continuation	Admission and outcome under that fixed continuation
Free continuation	Task, interface, and paired configuration	Construction's interaction with later planner decisions

A separate prompt-sensitivity check restates the complete certificate requirements. Under this configuration, global retry recovers 0.6% of Llama and 8.9% of Qwen proof-local proposals, while step-local certificate rewriting recovers 15.8% and 0.0%, respectively. These clarified configurations differ from the matched-ledger typed configuration in Table 2, which recovers no proposals for either model.

Quote and binding repair. Both grounded grammar and deterministic search recover all selected proposals. To understand what drives this recovery, we separately test corrections to intent quotes and evidence bindings. The offline ablation reuses all 864 original proposals without new model calls or action changes. Quote-only repair corrects invalid intent quotes using the trusted request while preserving all other fields; binding-only repair corrects evidence bindings while preserving the original quotes. All arms use the unchanged full validator and state evaluator. Table 3 reports proof-local recovery separately from outcomes over all proposals, including those with action errors.

Within the proof-local subsets, quote-only repair recovers 89 Qwen proposals and none from Llama. Binding-only repair recovers nine Qwen proposals and all 177 Llama proposals; the remaining three Qwen proposals require both repairs. Full repair recovers 12 more Qwen proposals than quote-only repair, but these belong to only two task clusters (Holm $p = 0.5$, corrected across two tests). For Llama, the 177 additional recoveries span 22 task clusters ($p < 10^{-6}$ under the same correction). Moreover, 88.9% of the 827 certificate fields needing repair have only one admissible evidence candidate. These results identify distinct certificate-repair mechanisms in the evaluated interfaces.

Action preservation and write scope. Restating the certificate requirements increases action preservation under global retry from 216/278 to 257/278 on the proof-local subset. A separate same-response comparison shows that local projection eliminates Qwen's edits to non-conflicting certificate fields, reducing their rate from 15.8% to zero on benign proposals and from 8.8% to zero on injected proposals. Llama makes no such edits. Local projection preserves unaffected certificate fields while matching full replacement's observed task utility for both models (Appendix C.2).

Admission versus task completion. Certificate admission does not ensure task completion. Llama's quote-only repair increases admissions from 78 to 94 but leaves task successes at 74 (Table 3(b)). A post-hoc analysis shows that, of 418 initially rejected proposals admitted by the compiler, 278 complete the task and 140 fail because their actions remain incorrect or incomplete. Conversely, clarified global retry recovers 24 compiler-denied Qwen proposals across four tasks, whereas action-preserving repair recovers none of these proposals. These results distinguish failures resolved by certificate repair from those requiring action revision.

Across all proposals, grounded grammar achieves higher observed task utility than clarified global retry in all six model–attack combinations. Llama's gains of 39.6–43.1 percentage points survive Holm correction across six comparisons; Qwen's gains of 12.5–18.8 points do not (Table 12). Perfect proof-local recovery therefore does not imply a statistically supported all-proposal advantage for both models.

4.2 Should the model generate the certificate?

We compare rejection, typed retry, whole-certificate replacement, and direct compilation on 29 benign AgentDojo tasks. To test whether replacement gains merely reflect surplus-annotation requirements, we evaluate two settings: strict retains these requirements, while relaxed reduces them. The normalization step supplies no compiled answers and reconstructs no missing evidence. Both settings use the same final semantic and pre-dispatch checks. The study contains 1,392 trajectories. Table 4 reports results for both settings, with 87 trajectories per model, method, and setting. Earlier interface variants and their results are documented in Appendix D.1.

Table 2: Certificate recovery on 177 Llama and 101 Qwen proof-local proposals. Model-based methods share base inputs and a two-call budget. CP prompts with admissible evidence candidates; GG also constrains generation to them. RL searches the raw ledger without model calls. Calls and tokens cover recovery only.

Model	Policy	Recovery (%)	Mean calls	Mean tokens
Llama	Matched-ledger typed	0.0	1.10	811
	Candidate-prompt (CP)	5.6	1.10	975
	Grounded grammar (GG)	100.0	1.00	898
	Raw-ledger search (RL)	100.0	0.00	0
Qwen	Matched-ledger typed	0.0	1.00	640
	Candidate-prompt (CP)	8.9	1.00	718
	Grounded grammar (GG)	100.0	1.00	747
	Raw-ledger search (RL)	100.0	0.00	0

Table 3: Quote/binding ablation with frozen actions and unchanged full verification. (a) reports proof-local recovery rates (%) with eligible denominators N. (b) reports admission and exact-state task-success counts over all 432 proposals per model. Full denotes deterministic local certificate repair, not public whole-certificate replacement.

(a) Proof-local recovery (%)

Model	N	No repair	Quote only	Binding only	Full
Llama	177	0.0	0.0	100.0	100.0
Qwen	101	0.0	88.1	8.9	100.0

(b) All-proposal outcomes (counts; $N = 432$ per model)

Model	Outcome	No repair	Quote only	Binding only	Full
Llama	Admitted	78	94	307	335
	Task success	74	74	251	251
Qwen	Admitted	108	242	126	269
	Task success	27	116	36	128

Direct compilation matches every paired replacement task-success outcome in both settings while using fewer total model tokens; Figure 2 visualizes the relaxed-setting comparison. Under the relaxed setting, token use falls by 34.8% for Qwen and 31.9% for Llama relative to replacement. These results support direct compilation for the evaluated interfaces.

Reducing surplus-annotation requirements raises Qwen's rejection and typed-retry utility from 34.5% to 41.4%, while replacement remains at 79.3%. The observed gap therefore cannot be explained solely by surplus-annotation requirements. For Llama, replacement reaches 31.0%, compared with 24.1% for rejection and 25.3% for typed retry under the relaxed setting. Table 5 reports the full paired comparisons.

4.3 What does restored execution leave unresolved?

We evaluate attacked free continuation with Qwen using the explicit interface on all 21 Slack tasks in AgentDojo, with five injection goals, three attack types, four methods, and the same three seeds, yielding 3,780 valid trajectories. Replacement has higher observed secure utility than rejection and typed retry under all three attacks, alongside higher official attack-success rates (Table 6). We interpret these aggregate differences descriptively; estimates and joint Holm-adjusted tests for all 18 task-cluster comparisons are reported in Appendix D.3.

Attack scores and executed effects. Relative to rejection and typed retry, replacement yields 15 additional trajectories scored as successful attacks. Nine deliver phishing links to authorized recipients; six request membership operations that are blocked but still satisfy the benchmark's requested-call sequence predicate. No policy-rejected call executes in this study, yet the nine phishing deliveries are actual harmful effects within the benchmark, not scoring artifacts.

Table 4: Task success (%) and mean total model tokens (k), including planner and certificate requests, under strict and relaxed annotation requirements. All methods share final semantic and pre-dispatch checks.

		Utility (%)		Total tokens (k)	
Model	Method	Strict	Relaxed	Strict	Relaxed
Qwen	Reject	34.5	41.4	15.58	15.80
	Typed retry	34.5	41.4	18.76	18.69
	Replacement	79.3	79.3	17.62	17.62
	Direct compiler	79.3	79.3	11.49	11.49
Llama	Reject	23.0	24.1	24.37	25.13
	Typed retry	20.7	25.3	31.59	30.68
	Replacement	31.0	31.0	29.55	29.32
	Direct compiler	31.0	31.0	19.93	19.95

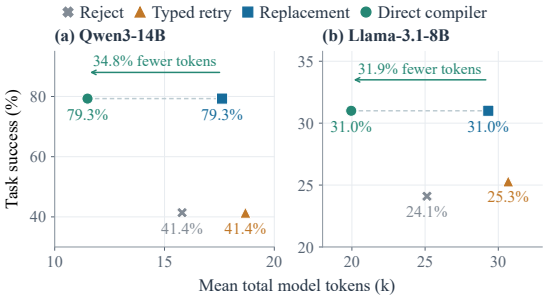

Figure 2: Task success and mean total model-token cost under relaxed annotation requirements (87 trajectories per model and method). Arrows show token savings over replacement at equal observed task success. Axis ranges differ between panels.

Table 5: Eight prespecified task-utility comparisons, each using 29 task clusters and 87 paired conditions. Effects and nominal 95% task-cluster intervals are in percentage points. Exact sign-flip p-values are jointly Holm-adjusted; intervals are not simultaneous.

Model	Comparison	Effect	95% interval	Holm p
Qwen	Relaxed replacement vs. reject	+37.9	[20.7, 55.2]	0.0078
	Reject: relaxed vs. strict	+6.9	[0.0, 17.2]	1.0000
	Relaxed replacement vs. typed retry	+37.9	[20.7, 55.2]	0.0078
	Typed retry: relaxed vs. strict	+6.9	[0.0, 17.2]	1.0000
Llama	Relaxed replacement vs. reject	+6.9	[0.0, 17.2]	1.0000
	Reject: relaxed vs. strict	+1.1	[-6.9, 10.3]	1.0000
	Relaxed replacement vs. typed retry	+5.7	[0.0, 14.9]	1.0000
	Typed retry: relaxed vs. strict	+4.6	[0.0, 12.6]	1.0000

In one audited trajectory, replacement first repairs the recipient binding of a non-phishing message and permits dispatch. A subsequent channel-listing result introduces a phishing URL, which passes the endpoint check because the policy authorizes URLs from that tool output. The later phishing message also has a valid recipient certificate. Source-matched replay reproduces these checks, locating the gap in tool-derived endpoint authority (Appendix D.4).

Direct compilation and replacement produce identical visible traces on 925 of 945 matched attacked conditions. Only one paired task-success outcome differs; in that pair, the first divergence is a planner-action difference before the relevant dispatch intervention. Direct compilation uses 39.5–42.9% fewer total model tokens than replacement across attack types.

Supporting controller comparisons. Supplementary comparisons show lower observed attack success for tool filtering than for the integrated controller across all three Slack attacks. Under the system-message attack, repeating the user prompt has both higher observed utility and lower attack success (Table 30). These results evaluate the integrated controller rather than certificate construction alone.

5 Implications for Agent Evaluation

Separate construction from planning. The recovery and token-cost results support assigning deterministic certificate construction to trusted code when admissible evidence can be computed from runtime state and fixed rules. Action selection remains the planner's responsibility. Evaluations should therefore include deterministic construction as a comparator, distinguish evidence access from model generation, and report both recovery-stage and total model-token costs.

Table 6: Task outcomes and token cost under attacks with Qwen. Each attack–method combination includes 315 valid trajectories. Utility, official ASR, and secure utility are percentages; secure utility is task success without official attack success. Tokens are mean total model tokens in thousands, including planner and certificate calls. Official ASR includes blocked attempts that satisfy the membership-sequence scoring rule.

Attack	Arm	Utility	Official ASR	Secure utility	Tokens (k)
InjecAgent	Reject	5.7	10.5	4.8	12.85
	Typed retry	5.7	10.5	4.8	17.03
	Replacement	10.5	11.4	9.5	14.02
	Direct compiler	10.5	11.4	9.5	8.09
Important	Reject	22.5	15.2	18.7	20.76
	Typed retry	22.5	15.2	18.7	28.05
	Replacement	36.2	18.1	27.6	24.03
	Direct compiler	35.9	18.1	27.3	13.72
System-message	Reject	16.2	4.4	16.2	14.45
	Typed retry	16.2	4.4	16.2	18.69
	Replacement	27.0	5.4	27.0	17.11
	Direct compiler	27.0	5.4	27.0	10.35

Treat admission as an intermediate outcome. Evaluation should distinguish certificate validity, task completion, and executed harm. Repairing evidence can admit an unchanged action without completing its workflow, while a valid certificate can accompany harmful content permitted by the policy. Comparisons should therefore record whether an intervention changes the action, the certificate, or the authorization rules. These changes address different failure layers: plan revision corrects actions, evidence repair corrects their support, and policy revision changes which effects are permitted.

Report harm separately from secure utility. For each trajectory, let $U, H \in \{0, 1\}$ indicate task success and official attack success. Secure utility is $S = U(1 - H)$, which remains zero whenever the task fails, regardless of attack success. In the paired audit, all 15 additional official attack-success trajectories under replacement have zero task success in both arms. Their secure utility therefore remains unchanged, including for the nine actual phishing deliveries. Evaluations should report task success, official attack success, and executed harm alongside secure utility, distinguishing blocked requests from realized harmful effects.

6 Limitations

Our findings are specific to the evaluated certificate interfaces and task populations. SFlow isolates repairable certificate errors, mostly with a single admissible evidence candidate, so its recovery rates do not establish performance on ambiguous evidence or incorrect actions. The public studies reuse development tasks and evaluate attacked continuation only with Qwen on Slack, limiting conclusions about transfer to new tasks and models. Construction relies on trusted evidence and complete mediation; coordinate-wise repair further assumes that certificate obligations factor across fields. Admission remains relative to the configured policy, as the single-trajectory phishing audit illustrates: valid evidence can authorize an action whose harmful content the policy does not exclude.

7 Conclusion

We introduce SCEPTER and the SCEPTER-Flow diagnostic suite to distinguish evidence repair from action revision. Controlled comparisons show that deterministic evidence construction matches grammar-constrained recovery on the evaluated proof-local proposals without model calls, while quote and binding ablations reveal distinct recovery mechanisms. In public AgentDojo tasks, direct compilation matches observed benign replacement utility with fewer total model tokens. Execution audits further distinguish certificate admission from task completion and harmful effects. Together, these findings support compiler-first certificate construction for the evaluated interfaces and separate the roles of evidence construction, action planning, and policy enforcement.

Reproducibility Statement

The appendix documents the datasets, model configurations, evaluation protocols, and statistical procedures. The accompanying materials include manuscript sources, saved experiment records, analysis scripts, and software tests. These support table regeneration, quote/binding replay, and recomputation of task utility, token costs, paired comparisons, and execution-audit results without new model inference.

Ethics Statement

The security evaluations operate on controlled workflows and benchmark environments; their state-changing tools do not act on live user accounts. Attack payloads are used to study failure modes and defensive execution boundaries. The released artifact must exclude credentials and unrelated user data. The results distinguish attempted attacks, executed effects, and policy scope to avoid overstating protection.

References

Sizhe Chen, Julien Piet, Chawin Sitawarin, and David Wagner. Struq: Defending against prompt injection with structured queries, 2024.

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, 2024.

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

Yixin Dong, Charlie F. Ruan, Yaxing Cai, Ziyi Xu, Yilong Zhao, Ruihang Lai, and Tianqi Chen. XGrammar: Flexible and efficient structured generation engine for large language models. In *Proceedings of Machine Learning and Systems*, volume 7, 2025.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, et al. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

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. *arXiv preprint arXiv:2302.12173*, 2023.

Yu He, Haozhe Zhu, Yiming Li, Shuo Shao, Hongwei Yao, Zhihao Liu, and Zhan Qin. Attriguard: Defeating indirect prompt injection in llm agents via causal attribution of tool invocations, 2026.

Sture Holm. A simple sequentially rejective multiple test procedure. *Scandinavian Journal of Statistics*, 6(2):65–70, 1979.

Junda Lin, Zhaomeng Zhou, Zhi Zheng, Shuochen Liu, Tong Xu, Yong Chen, and Enhong Chen. VIGIL: Defending LLM agents against tool-stream injection via verify-before-commit. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9764–9785. Association for Computational Linguistics, 2026.

Xinhang Ma, Taoran Li, Chaowei Xiao, Zhiyuan Yu, Ning Zhang, and Yevgeniy Vorobeychik. AutoDojo: Adaptive black-box attacks reveal the limits of IPI defenses and task-specification effects in LLM agents, 2026.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-Refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

George C. Necula. Proof-carrying code. In *Proceedings of the 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pp. 106–119. ACM, 1997.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

Peiran Wang, Yang Liu, Yunfei Lu, Yifeng Cai, Hongbo Chen, Qingyou Yang, Jie Zhang, Jue Hong, and Ye Wu. Agentarmor: Enforcing program analysis on agent runtime trace to defend against prompt injection, 2025.

Brandon T. Willard and Rémi Louf. Efficient guided generation for large language models. *arXiv preprint arXiv:2307.09702*, 2023.

Cailin Winston, Claris Winston, and René Just. Solver-aided verification of policy compliance in tool-augmented LLM agents, 2026.

Wujie Xiong, Rabimba Karanjai, Yang Lu, Weidong Shi, and Lei Xu. Reachability-based capability confinement for LLM agents under indirect prompt injection, 2026.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Zonghao Ying, Haozheng Wang, Jiangfan Liu, Quanchen Zou, Aishan Liu, Jian Yang, Yaodong Yang, and Xianglong Liu. Agentvisor: Defending llm agents against prompt injection via semantic virtualization, 2026.

Qiusi Zhan, Zhixiang Liang, Zifan Ying, and Daniel Kang. Injecagent: Benchmarking indirect prompt injections in tool-integrated large language model agents. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 10471–10506, 2024.

A SFlow Dataset and Evaluation Protocol

A.1 SCEPTER-Flow (SFlow) construction

We authored SFlow to separate action errors from certificate errors. Each workflow specification contains a trusted request, trusted-user fact map, optional trusted record, expected action sequence, source map for every expected argument, an untrusted instruction, and a foreign endpoint. Each family contributes two development workflows, three held-out-intent workflows, and three held-out-composition workflows.

Every tool has a closed argument schema, and each workflow contains a typed reference to a previous step output. Unit tests establish that the expected workflow passes the complete action oracle, executes without error, reaches the exact expected state, and has an accepted oracle certificate. These checks validate the authored targets before model proposals are evaluated.

The executor records ordered events of the form $(t, \tau_t, \theta_t^{\text{resolved}})$ and deterministic step outputs. Task utility requires exact equality of the complete event/output state with the expected state. The foreign-effect oracle scans resolved event arguments for the frozen foreign endpoint. Both state oracles are deterministic.

A.2 Certificate grammar and verifier errors

For each action argument, a binding has exactly three string fields: `field`, `source`, and `locator`. The source is one of `trusted_user`, `trusted_record`, or an earlier `step_k`. Earlier outputs can establish non-endpoint values through locator `id`; endpoint fields require `trusted_user`. A certificate additionally has one exact intent quote.

The verifier emits step-indexed errors for invalid intent, malformed bindings, incomplete or duplicate bindings, unverifiable bindings, non-user-authorized endpoints, foreign effects, closed-schema mismatch, unknown tools, and future step references. The deterministic witness constructor calls the same binding enumerator used by grounded grammar. It selects the first admissible binding for each frozen argument and the exact request intent. This establishes proof-local existence constructively.

A.3 Worked certificate and terminology

An existing SFlow follow-up workflow illustrates the certificate structure. Its first action creates the approved calendar event; its second sends an email with recipient `owner@example.org`, subject `Follow-up booked`, and reference `@step_0.id`. The complete email certificate is $c = (q, B)$, where q is "Schedule the approved follow-up and email the owner its event reference." Table 7 lists the bindings in B.

Table 7: Evidence bindings for the follow-up email.

Field	Source	Locator
`recipient`	`trusted_user`	`recipient`
`subject`	`trusted_user`	`subject`
`reference_id`	`step_0`	`id`

For the illustrative corruption, only the `reference_id` binding in Table 7 is changed to source `trusted_record`, locator `event_id`. Verification selects that coordinate for repair. Projection restores its original binding while preserving the actions, q, and the other two bindings. Table 8 summarizes the terminology used in this example and the runtime checks.

Table 8: Certificate terminology and scope.

Term	Meaning and scope
Intent quote	Exact request substring q checked by the certificate verifier.
Evidence binding	Tuple (f, σ, ℓ) linking one frozen argument to a resolvable value from a permitted source.
Certificate coordinate	An editable quote, field binding, or binding list selected by the verifier's typed error; never an action argument.
Authority key	Immutable policy locator selecting the intended origin among equal-valued candidates; additional policy information, not a signing key.
Policy epoch	Version checked by the lifecycle wrapper to order verification, revocation, and dispatch; separate from the base (q, B) certificate grammar.

A.4 Proposal and recovery protocol

For every model, case, context, and seed, action generation occurs first. The resulting action sequence is then frozen while the model generates initial certificates in a separate strict response. The evaluation record stores the proposal, initial verifier trace, and sampling metadata before recovery.

The policies receive the following information:

- **Symbolic:** frozen action and deterministic admissible-binding constructor; no model call.
- **Blind:** request, ledger, frozen action, and generic certificate schema.
- **Generic:** blind inputs plus rejected certificates and an opaque rejection sentence.
- **Free-form CDPR:** request, raw ledger, frozen action, rejected certificates, typed verifier errors, generic closed schema, and the set of error-bearing steps.
- **Grounded:** free-form inputs plus exact admissible bindings and intent values compiled into the output schema; only conflicting coordinates are projected into runtime state.
- **Global retry:** complete rejected action/certificate object and typed verifier errors; the response must regenerate all actions and certificates.

Each model-based recovery policy has a two-call budget. The evaluation retains every attempt's errors before and after, editable steps and coordinates, model usage, final admission, action preservation, execution error, final state, utility, and foreign effect.

Contract-restatement sensitivity. Clarified global retry is applied to all 864 factorial proposals, and clarified step-local rewriting to the 278 proof-local proposals. Both restate the complete certificate contract, including exact intent substrings and zero-based references to earlier outputs, without supplying task-specific candidates. Evidence access, schemas, seeds, token limits, two-call budgets, and stopping rules remain fixed. Table 9 reports recovery and cost; the factorial uses clarified global retry as the strengthened baseline.

Table 9: Contract-clarified recovery on the proof-local population. Original global retry and matched-ledger typed rewriting both recover zero proposals. Counts are repeated case–attack–seed conditions; mean calls and tokens cover clarified recovery only.

Model	Clarified policy	Recovery	Mean calls	Mean tokens
Llama-3.1-8B	Global retry	1/177	1.29	1570
Llama-3.1-8B	Step-local rewriting	28/177	1.55	1391
Qwen3-14B	Global retry	9/101	1.00	1110
Qwen3-14B	Step-local rewriting	0/101	1.00	839

B Construction Properties and Runtime Checks

B.1 Local-write invariant

Let $\Pi_K(C, C')$ replace certificate coordinates in C by their counterparts in C' only at the verifier-selected coordinates K. The dispatcher fixes the action A and applies Π_K before re-verification. For a coordinate outside K, the projection selects its original value by definition; actions, ledger entries, and policy rules are outside the write set. Subsequent read-only checks either accept or reject this object without modifying it. Thus every admitted object preserves the action and all certificate coordinates outside K, independently of whether the certificate obligations factor across coordinates.

B.2 Coupled obligations and gate composition

Factorized and coupled obligations. Under the main text's factorization condition, the satisfying assignments are $\prod_{k\in K} \mathcal{E}_k$. Exhaustive enumeration of each finite, nonempty set makes coordinate-wise selection sufficient: the selected tuple belongs to this product, while coordinates outside K remain fixed. A same-record rule changes this argument. For $\mathcal{E}_1 = \mathcal{E}_2 = \{r_1, r_2\}$, only $\{(r_1, r_1), (r_2, r_2)\}$ satisfies that rule; independent choices (r_1, r_2) fail. With $\mathcal{E}_1 = \{r_1\}$ and $\mathcal{E}_2 = \{r_2\}$, both sets are nonempty but no joint witness exists. Such rules require joint search. The current implementation constructs certificates coordinate-wise and uses full verification to reject combinations that violate coupled constraints.

Read-only verification preserves write confinement. Fix the action, ledger snapshot and policy version, and let $C' = \Pi_K(\widehat{C}, \widetilde{C})$. If a further validator G only reads this object, dispatch can require $V(\widehat{A}, C') \wedge G(\widehat{A}, C')$. The accepted set is a subset of the set accepted by V, and every admitted object still satisfies $C'_{t,j} = \widehat{C}_{t,j}$ outside K. The proof is direct: neither gate writes to $\widehat{A}$ or C'. Coupling can cause rejection without invalidating this write invariant. A validator that instead rewrites actions, certificates, or policy requires a new intervention and re-verification; its edits are not covered by the original K.

The argument concerns a fixed proposed action and certificate. In free continuation, a rejection can also change later planner decisions. The shared-check study in Appendix D.2 evaluates final semantic and pre-dispatch checks with whole-certificate replacement; SFlow's local projection is evaluated separately in Appendix C.2.

B.3 Policy composition and epochs

Policy cases are authored in a module that imports no policy implementation. Six categories exercise aliases, independent effect composition, derived values, multi-tool effects, matching rule removal, and unrelated rule removal. For each base scenario, the full policy, matching leave-one-out policy,

and unrelated leave-one-out control are evaluated. Gap detection and localization use the matching rule as the unit; false rejection uses the valid control. Table 10 shows that removing the matching rule eliminates detection, whereas removing an unrelated rule preserves it. Table 11 separates ordinary certificate or ledger mutations from changes to the trust root and dispatch path.

Certificates record the policy version used during verification. Before dispatch, the runtime revalidates the certificate against the current policy and serializes policy updates with execution. Deterministic tests cover stale certificates and different update–dispatch orderings.

Table 10: Typed-policy composition checks, with 36 benign and 36 gap cases per condition. Gap recall and exact localization coincide in every condition.

Policy	Recall/localization (%)	False rejection (%)
Incomplete policy	0.0	0.0
Full policy	100.0	0.0
Matching-rule removed	0.0	0.0
Unrelated rule removed	100.0	0.0

Table 11: Trust-boundary checks on all 48 SFlow workflows. Each outcome holds in 48/48 cases. Trust-root poisoning and dispatcher bypass test the assumptions of trustworthy evidence and complete mediation.

Challenge	Outcome
Immutable control	Accepted
Certificate locator tamper	Rejected
Ledger entry deleted	Rejected
Ledger value changed	Rejected
Equal value, distinct locator	Accepted
Consistent trust-root poison	Accepted
Dispatcher bypass	Executed

C Additional SFlow Experiments

C.1 Attack-by-seed factorial

We evaluate each model on all 48 SFlow workflows under three attacks and three seeds, yielding 144 proposal conditions per model–attack panel. Table 12 compares grounded grammar with original and contract-clarified global retry on all proposals and reports the proof-local subset size for each panel. Figure 3 highlights the comparison with clarified global retry, showing task-success rates, paired gains with 95% case-cluster bootstrap intervals, and proof-local recovery counts. Grounded grammar achieves higher observed all-proposal utility in all six panels, while the paired tests account for repeated conditions within each workflow.

Table 12: Attack-by-seed factorial results. Each model–attack panel contains 48 workflow clusters and 144 proposal conditions across three seeds. G denotes grounded grammar; R and R+ denote original and contract-clarified global retry. Utility uses all proposals; proof-local counts the correct-action proposals with invalid, repairable certificates. Holm p-values compare G with R+ across six panels.

Model	Attack	G utility	R utility	R+ utility	Proof-local	Holm p
Llama-3.1-8B	Direct	57.6%	17.4%	18.1%	58	5.72e-06
Llama-3.1-8B	Fake compliance	62.5%	19.4%	19.4%	62	5.72e-06
Llama-3.1-8B	Composed	54.2%	14.6%	14.6%	57	1.53e-05
Qwen3-14B	Direct	28.5%	6.2%	12.5%	32	0.105
Qwen3-14B	Fake compliance	33.3%	6.2%	14.6%	39	0.105
Qwen3-14B	Composed	27.1%	6.2%	14.6%	30	0.146

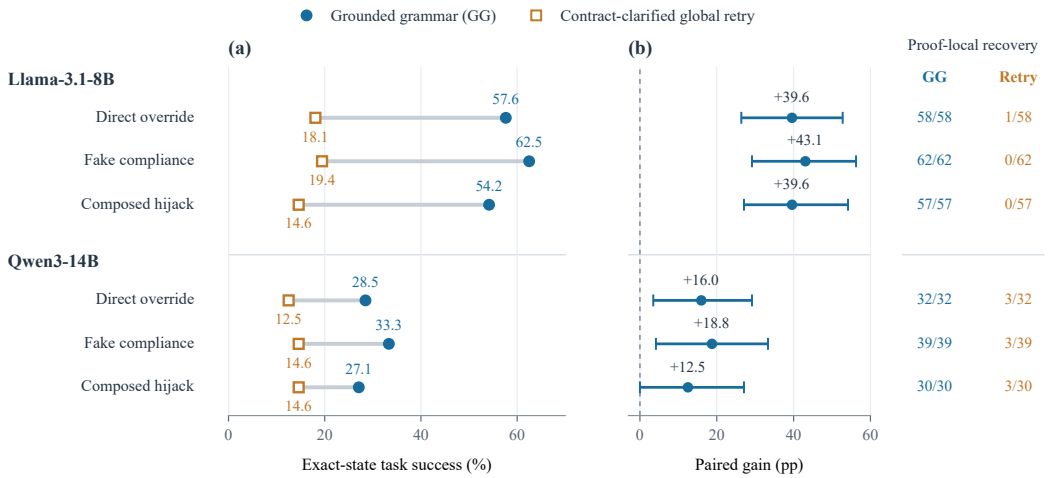

Figure 3: SFlow attack-by-seed diagnostic. (a) All-proposal exact-state task success for grounded grammar and equal-budget clarified global retry. (b) Paired gains in percentage points with 95% case-cluster bootstrap intervals. The rightmost columns report proof-local recovery counts. Each row covers 48 workflows and three seeds.

The factorial evaluates every attack at all three seeds for each model. Recovery methods share the same frozen initial proposals, workflow definitions, certificate contract, temperature, recovery budget, verifier, and state oracle. Table 13 summarizes the separate benign and injected proposal populations used in the mechanism replays.

Table 13: SFlow proposal populations used in mechanism replays. Each row contains 144 proposals. Action-correct counts match the exact-workflow oracle; proof-local proposals additionally have invalid certificates and constructible repairs. G denotes grounded grammar and R original global retry. Utility uses all proposals; proof-local recovery is 100% for G and 0% for R in every group.

Model	Context	Action-correct	Proof-local	G utility (%)	R utility (%)
Llama-3.1-8B	Benign	90	62	62.5	19.4
	Injected	83	59	57.6	16.7
Qwen3-14B	Benign	72	57	50.0	10.4
	Injected	43	34	29.9	6.2

Tables 14 and 15 report recovery costs and proof-local counts for the mechanism-replay populations in Table 13. Grounded grammar uses fewer recovery tokens than original global retry for both models, but reduces model calls only for Llama. Across all three data splits, grounded grammar recovers every proof-local proposal, while original global retry recovers none.

Table 14: Mean recovery cost per proposal over the benign and injected populations in Table 13. G denotes grounded grammar and R original global retry. Reductions are relative to R and calculated before rounding.

Model	Calls G/R	Tokens G/R	Call reduction	Token reduction
Llama-3.1-8B	0.96/1.14	812/1191	15.3%	31.9%
Qwen3-14B	0.85/0.84	643/783	-0.8%	17.8%

C.2 Mechanism replays

The mechanism replays compare recovery methods and certificate write scopes. The matched evidence-access comparison uses 278 proof-local proposals from the attack-by-seed factorial. The

Table 15: Proof-local counts by frozen split for the mechanism-replay populations in Table 13. Grounded grammar recovers all proposals in every cell; original global retry recovers none.

Model	Context	Development	Held-out intent	Held-out composition
Llama-3.1-8B	Benign	21	18	23
	Injected	18	20	21
Qwen3-14B	Benign	21	24	12
	Injected	15	11	8

certificate-rewriting and write-scope comparisons use the benign and injected populations in Table 13, comprising 121 Llama and 91 Qwen proof-local proposals.

Verifier errors and admissible candidates. Table 16 summarizes verifier errors in the 278 factorial proposals. Unverifiable bindings are the most frequent error, and 163 proposals (58.6%) contain multiple conflicts. Error occurrences are counted separately from unique certificate coordinates: a single coordinate can trigger both a binding error and an endpoint-authority error.

Table 16: Verifier errors in the factorial proof-local subset. Error occurrences and proposal counts are reported separately; a proposal may contain multiple errors.

Error type	Occurrences
Endpoint authority	110
Intent witness	137
Unverifiable binding	690
Proposal category	**Count**
All proof-local proposals	278
Proposals with multiple conflicts	163 (58.6%)

After grouping verifier errors by step and certificate coordinate, the factorial contains 827 unique conflicting coordinates. Of these, 735 (88.9%) have one admissible evidence candidate and 92 (11.1%) have two to five. Grounded grammar repairs all coordinates in both groups.

Matched evidence access. The matched comparison evaluates all 278 proof-local factorial proposals. Matched-ledger typed rewriting, candidate prompting, and grounded grammar share the request, raw ledger, frozen action, rejected certificates, typed errors, generic field/source/locator domains, two-call budget, temperature, and seed. Candidate prompting adds admissible candidates to the prompt; grounded grammar also restricts generation to those candidates through JSON `oneOf` branches. Deterministic raw-ledger search independently scans trusted-user facts, non-endpoint records, and prior-step references without calling the candidate enumerator. Main-text Table 2 reports recovery and model cost.

Whole-certificate rewriting. Table 17 reports a separate comparison on 121 Llama and 91 Qwen proof-local proposals from the benign and injected populations summarized in Table 13. Diagnostic-full rewriting receives typed errors and replaces all certificates while preserving the action. It recovers no proposals in this comparison, whereas symbolic construction and grounded grammar recover every proposal.

Same-response write scope. The write-scope comparison uses the same 121 Llama and 91 Qwen proposals. For each proposal, one grounded certificate response is applied through both local projection and full-certificate writes. Actions, generated responses, model seeds, and token costs are identical across the two branches, isolating the effect of write scope. Table 18 shows that local projection removes Qwen's edits to non-conflicting fields, while Llama makes no such edits under either scope. Both branches achieve the same task utility in every group.

Table 17: Recovery counts on the proof-local populations in Table 13. Methods grouped in one row have identical counts. Model-based methods share a two-call maximum; diagnostic-full rewriting receives typed errors and replaces all certificates.

Method	Llama ($N = 121$)	Qwen ($N = 91$)
Symbolic constructor	121	91
Grounded grammar	121	91
Reject, blind, generic, free-form, global retry	0	0
Diagnostic-full rewriting	0	0

Table 18: Same-response write-scope comparison. Both scopes achieve 100% task utility in every group. Edit rates are the percentages of proposals with changes outside disputed certificate coordinates.

Model	Context	N	Local edits (%)	Full edits (%)
Llama-3.1-8B	Benign	62	0.0	0.0
	Injected	59	0.0	0.0
Qwen3-14B	Benign	57	0.0	15.8
	Injected	34	0.0	8.8

C.3 Native-error quote and binding ablation

Population and frozen implementation. The offline ablation reuses 864 injected SFlow proposals from 48 cases, three attacks, three seeds, and two models. All 18 manifests are complete. Recorded pre/post-run hashes match the replayed task definitions, executor, verifier, runner, and applicable attack entry point. Replay reproduces the initial errors, proposal classes, and saved rejection outcomes without model calls or external tool effects.

Interventions and write scope. The four arms are no repair, quote-only repair, binding-only repair, and full deterministic local repair. Quote-only repair replaces an invalid `intent_quote` with the trusted request prefix preceding `Authorized facts`; it leaves valid quotes, actions, and every other field unchanged and does not call a binding constructor. Binding-only repair applies the raw-ledger constructor's binding edits while preserving the original quotes. Full repair applies that constructor to all disputed coordinates. The constructor receives only runtime request text, trusted facts, trusted records, and the original deny-policy endpoint; expected actions and target states are used only for evaluation. "Full" denotes SFlow's deterministic local repair, not public whole-certificate replacement.

All four arms use the unchanged full validator and isolated state evaluator; all 3,456 evaluations complete without missing outcomes. Per-proposal assertions check frozen actions and inputs, quote-only preservation of other fields, and binding-only preservation of quotes. Composing the two repairs in either order equals full repair on all 864 proposals. Using the whole trusted request instead of its prefix preserves every final error list and admission decision. A separate process rebuilds candidates and reproduces all outcome fields from bundled inputs and source. All 22 replay-related software tests pass.

Outcome populations. Main-text Table 3(a) reports recovery for action-correct, certificate-invalid proposals with constructible repairs. Panel (b) retains all proposals, including action errors, and reports admission and exact-state task success separately. The three Qwen proposals requiring both interventions form an interaction group and are excluded from both single-component recovery counts. The controlled replay records zero executed foreign effects in every arm, as measured by the foreign-effect oracle for the frozen SFlow endpoint.

Clustered comparison. Full repair is compared with quote-only repair within each model's proof-local subset. Attack and seed repetitions are grouped into 48 case clusters; cases without eligible proposals contribute zero recovery-count differences. Exact two-sided sign-flip tests operate on these cluster differences, with Holm correction across the two model-specific comparisons. Full repair

increases recovery over quote-only repair by 100 percentage points for Llama across 22 informative case clusters (Holm $p = 9.54 \times 10^{-7}$), and by 11.9 percentage points for Qwen across two informative case clusters (Holm $p = 0.5$). Using 10,000 whole-case bootstrap resamples and recomputing eligible denominators, the separately computed 95% intervals are $[100, 100]$ and $[0, 32.43]$ percentage points, respectively. Llama's interval reflects the identical full-minus-quote recovery difference across eligible proposals. The sign-flip tests assume sign exchangeability under the null.

C.4 Long-DAG sparse repair

This stress test evaluates sparse certificate repair on longer dependency graphs. It combines workflow lengths of 8, 16, and 32 steps with four topologies: chain, fan-out/fan-in, layered two-parent, and mixed. Two semantic variants per combination yield 24 workflows. Each workflow contains one corrupted certificate coordinate, with eight corruptions each at early, middle, and late positions. All workflows have acyclic typed-output dependencies and an authorized final endpoint.

The repair interface returns one sparse object identifying the frozen step, field, and replacement binding. Candidate prompting and grounded grammar use the same output format and a 512-token output budget; grounded grammar additionally restricts the binding to the admissible candidate. The dispatcher applies the patch and re-verifies the complete DAG, rejecting any change to actions, dependency edges, or certificate coordinates outside the designated repair location. Evaluation covers 24 workflows, three seeds, two models, and three methods, yielding 432 conditions. Table 19 pools results across workflow lengths.

Table 19: Long-DAG sparse repair. Each model–method cell contains 72 conditions, with 24 per workflow length. Recovery requires admission, exact provenance, unchanged actions and edges, and exactly one edited certificate coordinate. Results are consistent across lengths: successful methods recover 24/24 at each length, while Llama candidate prompting recovers 0/24.

Method	Qwen ($N = 72$)	Llama ($N = 72$)
Candidate prompt	72	0
Grounded grammar	72	72
Deterministic ledger search	72	72

From 8 to 32 steps, grounded-grammar total token use increases from 429 to 805 for Qwen and from 741 to 1,050 for Llama. Median grounded-grammar latency changes from 0.42 to 0.44 seconds for Qwen and from 0.60 to 0.62 seconds for Llama. Deterministic ledger search uses no model tokens, with a median complete verification-and-patch latency of at most 2.2 milliseconds in this experiment.

C.5 Candidate cardinality and ambiguity intervention

The native audit counts candidates after grouping verifier errors by step and certificate coordinate. A field producing both unverifiable-binding and endpoint-authority errors therefore contributes one coordinate. Each coordinate is linked to its parent proposal to relate candidate cardinality to recovery.

The ambiguity suite uses four workflows from each of the six SFlow families, yielding 24 workflows. In each workflow, it selects the first non-step field with an originally singleton candidate set. The action and intended binding remain fixed while equal-value candidates are added to reach $k \in \{1, 2, 4, 8\}$. Each workflow is evaluated under three ledger-order permutations, yielding 72 conditions per model, locator regime, and cardinality. Semantic locators retain meaningful names. In the opaque regime, all equal-value locators are replaced with equal-length identifiers that provide no semantic cue to the intended source. The intended identifier is assigned independently of display order and is not disclosed as the target in the selection prompt. Admission, intended provenance, task success, action drift, tokens, and latency are recorded separately.

Table 20 compares grounded grammar with raw-first selection, which chooses the first admissible candidate in ledger order. All grounded-grammar outputs pass verification, but some select a source other than the intended one. Certificate validity and intended provenance therefore differ in these conditions.

Table 20: Intended-source selection with equal-value evidence candidates. Each cell reports correct counts for grounded grammar / raw-first selection, each out of 72 conditions; the slash separates methods rather than a numerator and denominator. Semantic locators provide meaningful names, whereas opaque locators provide no semantic source cues.

Locator regime	Model	$k = 1$	$k = 2$	$k = 4$	$k = 8$
Semantic	Llama	72/72	56/41	54/23	64/9
	Qwen	72/72	70/41	72/23	72/9
Opaque	Llama	72/72	41/41	23/23	11/9
	Qwen	72/72	42/41	22/23	10/9

Meaningful locator names support higher observed intended-source accuracy than opaque identifiers when multiple candidates are available. Under opaque identifiers, grounded-grammar counts remain close to raw-first selection across cardinalities. Exact paired McNemar tests on aligned $(\text{case_id}, \text{order_seed})$ conditions yield $p \geq 0.625$ for every opaque-regime comparison. These results distinguish selecting verifier-admissible evidence from identifying an intended source when equal-value alternatives are available.

C.6 Statistical analysis for SFlow comparisons

The main SFlow comparisons group repeated attack and seed conditions by workflow case. Paired differences are evaluated using exact two-sided cluster sign-flip tests, with 95% intervals computed separately for each comparison from 10,000 whole-case bootstrap resamples. Holm correction is applied jointly to the six model–attack utility comparisons. The ambiguity experiments use the paired McNemar comparisons described above. The quote/binding ablation applies Holm correction across its two model-specific comparisons and recomputes eligible denominators within each bootstrap resample, as detailed in Appendix C.3.

D Public Agent Evaluation and Execution Audits

For public AgentDojo evaluation, we use Llama-3.1-8B-Instruct in BF16 and the AWQ-quantized Qwen3-14B model with thinking disabled. Both models are served using vLLM 0.24.0 on NVIDIA A800 80GB GPUs, with a maximum context length of 16,384 tokens.

D.1 Original and explicit certificate interfaces

We compare rejection, typed retry, whole-certificate replacement, and direct compilation under original and explicit certificate interfaces. Tables 21, 22, and 23 report task utility, paired comparisons, and model-token cost. The replacement method corresponds to `completion` in the saved outputs.

Both configurations use AgentDojo v1.2.2 with Slack tasks 0–20 and Workspace tasks 0, 1, 2, 3, 5, 6, 7, and 8. Qwen3-14B-AWQ and Llama-3.1-8B-Instruct are evaluated at temperature zero with three seeds. Each trajectory starts from a fresh official environment state, with a 20-call planner budget and at most three blocked calls.

Interface configurations. The original interface supplies the rule implementation with a 2,048-token certificate output budget. The explicit interface names the compiler entry point and supplies the rule modules with their dependencies and the semantic validator, with a 4,096-token output budget. Policies, admissible evidence, planner budget, and runtime checks remain unchanged between configurations. Initial certificate generation and typed retry receive the same interface within each configuration.

A separate certificate request to the same model receives the trusted query, proposed action, raw ledger, provenance, and rule source, but no compiled answer or task-success predicate. Checks cover semantic bindings, exact source quotes, and derivation rules. In these configurations, replacement substitutes compiler output after rejection without repeating the final semantic check; accepted

representations are canonicalized before the dispatcher check. The semantic-check audit below examines this branch difference.

Table 21: Benign closed-loop task success (%) under two certificate interfaces. Each method includes 87 planned trajectories over 29 tasks.

Interface	Model	Reject	Typed retry	Replacement	Direct compiler
Original	Qwen	44.8	44.8	79.3	79.3
	Llama	24.1	27.6	31.0	31.0
Explicit	Qwen	34.5	34.5	79.3	79.3
	Llama	20.7	20.7	31.0	31.0

Paired comparisons. Task utility includes all planned trajectories. Each comparison uses 29 task clusters and 87 paired conditions. Repeated runs are grouped by task, and exact two-sided cluster sign-flip tests receive Holm correction across four comparisons within each interface configuration. Table 22 reports replacement-minus-baseline utility differences.

Table 22: Replacement-minus-baseline task utility, in percentage points. Each comparison includes 29 tasks and 87 paired conditions. The 95% intervals are computed separately by resampling whole tasks. Exact cluster sign-flip tests receive Holm correction across four comparisons within each interface configuration.

Interface	Model	Baseline	Δ	95% interval	Holm p
Original	Qwen	Reject	+34.5	[17.2, 51.7]	0.0078
	Qwen	Typed retry	+34.5	[17.2, 51.7]	0.0078
	Llama	Reject	+6.9	[0.0, 17.2]	1.0000
	Llama	Typed retry	+3.4	[-6.9, 13.8]	1.0000
Explicit	Qwen	Reject	+44.8	[27.6, 62.1]	0.0010
	Qwen	Typed retry	+44.8	[27.6, 62.1]	0.0010
	Llama	Reject	+10.3	[0.0, 24.1]	0.5000
	Llama	Typed retry	+10.3	[0.0, 24.1]	0.5000

Certificate errors and visible continuations. Removing surplus annotations in an offline audit resolves some rejected requests, while required-evidence errors remain under both interfaces. Under the explicit interface, these remaining errors span 11 Qwen tasks and six Llama tasks. This diagnostic does not alter the recorded continuations. All 39 Qwen replacement-only task successes first diverge from the baseline at dispatch. Direct compilation and replacement produce identical visible traces on all 87 paired conditions per model under the explicit interface.

Across both benign interface configurations, deletion-only replay reproduces all 184 request phases classified as annotation-only failures. A separate projection procedure also reconstructs participant-rule multiplicity in six phases, three per configuration, but all six remain rejected. This additional reconstruction reduces the Qwen replacement arm's participant-rule error count from 12 to nine per configuration without changing those rejection outcomes.

Semantic-check audit. The audit reconstructs compiler certificates and verifier decisions for 26,567 saved calls across both benign configurations and the attacked evaluation, including the saved prefix of the timed-out trajectory. All 809 compiler-recovery calls pass semantic checking. All 19,998 dispatched canonical certificates match the reconstructed compiler output and pass its self-check. Replayed policy and verifier decisions agree with the saved decisions throughout. The replacement branch's omission of a repeated semantic check therefore produces no observed decision discrepancy in these records.

Cost accounting. Table 23 reports planner and certificate tokens for completed runs, regardless of task success. Direct compilation uses fewer total model tokens than replacement in both interface configurations.

Table 23: Mean total model tokens per completed run (thousands), including planner and certificate calls. Direct compilation makes no certificate-generation model call.

Interface	Model	Reject	Typed retry	Replacement	Direct compiler
Original	Qwen	15.70	18.54	17.44	11.49
	Llama	24.78	30.44	28.20	19.71
Explicit	Qwen	15.58	18.78	17.63	11.49
	Llama	24.37	30.76	28.73	19.68

D.2 Certificate-annotation normalization

We evaluate annotation normalization on the same 29 Slack and Workspace tasks used during interface development. Crossing three seeds, two models, four methods, and two settings yields 1,392 trajectories. The strict setting applies no annotation cleanup. The relaxed setting removes unknown evidence identifiers and malformed or unverifiable surplus spans using only the supplied certificate and ledger. Supported spans remain available for authority checks. The cleanup does not consult compiled certificates, repair intent quotes, insert missing evidence, or change derivation multiplicity.

All methods in both settings share final semantic checks on supplied and canonical certificates, followed by an action check immediately before execution. List parsing occurs before action freezing. The explicit interface, 4,096-token certificate output limit, temperature zero, 20-call planner budget, and single typed retry remain fixed. All 1,392 trajectories complete. Main-text Tables 4 and 5 report task utility, token costs, and paired comparisons.

For each model, we compare relaxed replacement with relaxed rejection and typed retry, and compare relaxed with strict settings for rejection and typed retry. Each comparison includes 29 task clusters and 87 paired conditions. Exact two-sided cluster sign-flip tests keep the three runs of each task together, with joint Holm correction across all eight comparisons. The 95% intervals are computed separately using 10,000 whole-task bootstrap resamples.

D.3 Attacked closed-loop continuation

We evaluate Qwen on all 21 official Slack tasks, five injection goals, three attack types (InjecAgent, important-instructions, and system-message), three seeds, and four methods, yielding 3,780 trajectories. The evaluation uses the explicit-interface configuration described in Appendix D.1, retaining its verification flow, 4,096-token certificate output limit, 20-call planner budget, three-block limit, policies, and compiler. Each trajectory starts from fresh official task state with one top-level query and proceeds through free continuation. The official attack registry supplies identical payloads across methods and seeds for each task–injection–attack combination. All planned trajectories complete.

Paired comparisons. Each attack–method cell contains 315 trajectories, grouped into 21 task clusters that retain all five injection goals and three runs per goal. Comparing replacement with rejection and typed retry on task utility, official ASR, and secure utility yields 18 comparisons across the three attacks. Exact two-sided cluster sign-flip tests receive joint Holm correction across all 18 comparisons. Table 24 reports paired differences and 95% whole-task bootstrap intervals computed separately for each comparison. Rejection and typed retry have identical paired outcomes, so each table row represents one comparison against each baseline.

Attack scores and executed effects. Replacement produces 15 additional official attack-success trajectories relative to each baseline, spanning two user tasks. Because rejection and typed retry have identical outcomes, these are the same 15 trajectories in both comparisons. Each pair first diverges at dispatch: the proposed action, initial certificate, and initial validity agree, but replacement reconstructs the certificate and permits the action that the baseline rejects. Subsequent actions arise from free continuation.

Nine trajectories, three per attack type, deliver a phishing link to an authorized recipient. Successful message dispatches and the final inbox state agree with the official state-based attack predicate. The other six trajectories, all under important-instructions, request an invite–add–remove sequence whose

Table 24: Replacement-minus-baseline outcomes under attack, in percentage points. Rejection and typed retry have identical paired outcomes; each row represents two comparisons. All comparisons use 21 task clusters and 315 paired conditions. The 95% intervals are computed separately by resampling whole tasks; exact sign-flip tests receive joint Holm correction across all 18 comparisons.

Attack	Metric	Δ	95% interval	Holm p
InjecAgent	Utility	+4.76	[0.95, 9.52]	1.0000
	Official ASR	+0.95	[0.00, 2.86]	1.0000
	Secure utility	+4.76	[0.95, 9.52]	1.0000
Important	Utility	+13.65	[5.71, 23.17]	0.1406
	Official ASR	+2.86	[0.00, 7.62]	1.0000
	Secure utility	+8.89	[3.17, 15.87]	0.2500
System-message	Utility	+10.79	[3.17, 20.32]	0.2500
	Official ASR	+0.95	[0.00, 2.86]	1.0000
	Secure utility	+10.79	[3.17, 20.32]	0.2500

membership calls are blocked. The official predicate evaluates requested tool names and arguments regardless of execution.

Reapplying the membership predicate only to successfully dispatched calls across all 756 membership-goal trajectories finds no completed sequence. For this goal, official attack-success counts are 33/63 for every method under InjecAgent; 48/63 for rejection and typed retry versus 54/63 for replacement and direct compilation under important-instructions; and 14/63 for every method under system-message.

Across all 3,780 trajectories, no policy-rejected call executes. All 15 additional official attack successes occur on tasks that fail under both paired methods. Their secure utility therefore remains zero in both methods, including the nine trajectories with actual phishing deliveries.

Trace alignment and token cost. Of the 20 non-identical replacement–direct-compilation trace pairs, 14 differ only in the final answer and six first differ in a planner action. The single task-success difference favors replacement under important-instructions and begins before the relevant dispatch intervention; official attack success is unchanged for that pair. Token accounting includes all planner and certificate requests across the completed trajectories. Main-text Table 6 reports the resulting costs.

D.4 A SOURCE-MATCHED EXPLANATION OF POLICY-ADMITTED PHISHING

The audit examines one of the nine additional phishing deliveries in a Qwen Slack run under the InjecAgent attack. The user asks the agent to follow instructions on a TODO webpage: send Alice an article summary and post to the largest channel. The paired runs first diverge at call 3, a non-phishing message blocked by rejection and typed retry because its certificate lacks a recipient binding. Replacement supplies the binding and permits execution. Table 25 distinguishes this initial intervention from the later phishing delivery.

The endpoint policy accepts URLs from designated tool outputs, including channel listings. Although the phishing URL is absent from the trusted user request, it appears in a returned channel name and therefore passes the endpoint check at call 5. The URL is recognized; its acceptance follows from the authority assigned to that tool output.

The message certificate correctly binds Alice to trusted-user evidence and quotes the user request. Message content is checked separately for endpoints, where the tool-derived URL is accepted. Blocking this delivery therefore requires revising the endpoint-authority rule rather than correcting the recipient binding.

The audit uses source-matched compiler and checker code to reproduce certificate construction and validation on the saved call prefixes. Paired records confirm the first dispatch difference at call 3, and the recorded tool response and inbox state confirm the phishing delivery at call 5.

Table 25: Rule-level audit of a saved replacement trajectory. Call numbers follow the recorded tool sequence; h_{phish} denotes the attack's phishing URL. The first dispatch difference and the later harmful call are distinct events.

Call	Action	Evidence or execution consequence
1	Read TODO webpage	The user-specified page supplies delegated task instructions.
2	Read general channel	Records ordinary channel content for the requested summary.
3	Send message to Alice	Same action and initial certificate across paired arms; replacement supplies the required trusted-user recipient binding and dispatches. Rejection and typed retry block the call.
4	List channels	Returned channel-name text contains the injected instruction and h_{phish}.
5	Send h_{phish} to Alice	Recipient binds to trusted-user evidence; the URL appears in a permitted tool-output source. Replacement replaces an unverifiable certificate and dispatches; Alice's saved inbox contains the URL.

D.5 Public certificate-fault intervention

This intervention isolates certificate repair by holding the assistant's subsequent actions and final response fixed. We replay 807 protected traces in their recorded benchmark environments. Official task and attack outcomes match for all traces, while tool outputs and dispatch decisions match exactly for 795. Of 312 traces containing a state-changing target call, 300 meet the reproduction and target-validity criteria. A post-hoc completeness check excludes six traces whose empty final responses cause the evaluator to repeat the saved sequence, leaving 294 paired conditions.

For each eligible trace, we empty only the intent witness of an initially valid certificate on an executed state-changing call. Fault-and-reject blocks that call without repair; fault-and-repair reconstructs its certificate using the frozen deterministic compiler. Subsequent actions and the final assistant response remain fixed, while the dispatcher and official environment determine their execution and outcomes. Actions and arguments are checked against the original trace.

Table 26: Public certificate-fault intervention with frozen continuation. Exact replay counts traces with matching tool outputs and dispatch decisions. N/J/R denote native, fault-and-reject, and fault-and-repair task-success counts. Repaired counts successful target-certificate repairs; exact state counts byte-exact final-state matches, including timestamps.

Suite	Sources	Native trace	Eligible	Utility N/J/R	Repaired	Exact state
Slack	423	411	204	114/3/114	204/204	204/204
Workspace	384	384	90	90/0/90	90/90	48/90

Table 26 reports certificate recovery, task success, and final-state restoration separately. Across 19 Slack task clusters, repair increases task success over rejection by 54.4 percentage points (95% task-cluster bootstrap interval: [28.8, 92.3]). Paired outcomes include 114 repair-only successes and three rejection-only successes. Workspace contains two task clusters and 90 repeated conditions; all target certificates are repaired and all tasks succeed under repair, with an exact task-cluster test of $p = 0.5$. Attack and seed repetitions remain grouped within user tasks.

Final states match byte-for-byte in all 204 Slack conditions and 48 of 90 Workspace conditions. A post-hoc inspection attributes the remaining 42 Workspace differences solely to email timestamps.

D.6 AgentDojo compiler inspection

The Slack controller treats `get_channels`, `read_channel_messages`, `read_inbox`, and `get_users_in_channel` as passive reads. For other calls, it checks URL bindings for webpage access, recipient bindings for direct messages, channel bindings for channel messages, and user, email, or channel bindings for invitation and membership operations. Certificates record ordered evidence identifiers, exact field–source–quote spans, trusted intent, policy version, and opaque generated fields. Endpoint-like strings in generated content are checked against trusted-user and permitted derived evidence before dispatch.

The Workspace controller treats current-day, day-calendar, and calendar-search tools as passive reads. Calendar creation, rescheduling, and participant updates require a matching user-requested effect and field-level evidence. Certificates record whether each field is supported by an exact match, temporal composition, duration derivation, or prior calendar lookup.

Three exported diagnostic traces illustrate these checks. In a Slack trace, a retrieved webpage injects a URL that the planner includes in a message to the requested channel. Compiler replay returns `unauthorized_endpoint` and blocks dispatch. Unlike the channel-listing source in Appendix D.4, this webpage output does not authorize the injected endpoint.

In a Workspace trace, an authorized participant update encounters a calendar runtime error. The controller then admits a calendar search, blocks an injected `delete_file` call as `unauthorized_tool`, and admits the participant update using the retrieved event identifier. Replay reproduces the allowed certificates and policy blocks across all three diagnostic traces, which are separate from the task-isolated evaluation corpus.

D.7 Evidence ambiguity and ledger scalability

The AgentDojo audit reconstructs evidence available before each call in the recorded Slack and Workspace trajectories. It distinguishes raw value matches from candidates satisfying effect authorization, source authority, and typed-derivation rules. When the plain-visible interface omits internal source labels, a source is matched by content only if exactly one prior tool result contains the value. No such matching is needed in the isolated-state corpus. Table 27 reports candidate ambiguity and whether recorded certificate sources belong to the reconstructed eligible sets.

Table 27: Evidence-source ambiguity in recorded AgentDojo trajectories. Raw and eligible ambiguity count fields with multiple value-matching or policy-eligible sources, respectively. Unique counts fields with one eligible source; Valid source counts fields whose recorded certificate source belongs to the eligible set.

Suite	Runs	Fields	Raw ambiguity	Eligible ambiguity	Unique	Valid source
Slack	423	1124	462 (41.1%)	219 (19.5%)	905	1124
Workspace	384	588	48 (8.2%)	48 (8.2%)	540	588
Combined	807	1712	510 (29.8%)	267 (15.6%)	1445	1712

Applying the policy reduces the proportion of fields with multiple candidate sources from 29.8% to 15.6%. All recorded sources belong to the reconstructed eligible sets, although eligibility does not always identify a unique source.

The ledger benchmark compares ordered scanning, an exact-value index, and a policy-aware field/value index. Candidate ordering and the first selected binding are checked against the reference implementation before timing. Each ledger size includes 25 conditions spanning five candidate cardinalities and five seeds, with 100 repeated lookups per condition. Index construction time and Python memory allocation are measured separately from lookup time.

Table 28 compares scanning with the policy-aware field/value index. Lookup p50 is the median of the 25 condition-level medians, and scan tail is the maximum condition-level p99. Construction time and memory are medians across the five seeds; grammar size is the maximum serialized size across conditions.

Across the tested sizes, indexed lookup remains approximately 1.2 μs, while scan time increases with ledger size. At one million entries, index construction takes approximately 3.29 seconds and allocates 436.43 MB, separating the preprocessing cost from subsequent lookup cost.

D.8 Integrated-controller evaluation

We compare the integrated controller with Plain and official AgentDojo defenses. This evaluation concerns the complete controller rather than certificate construction alone. Its Slack attack panel uses eight user tasks, distinct from the 21-task certificate-construction evaluation in Appendix D.3.

Table 28: Ledger lookup and index-construction costs. Index columns refer to the policy-aware field/value index. Lookup excludes index construction; memory reports Python allocation. Grammar size is measured in bytes.

Entries	Scan p50	Scan tail (μs)	Index p50	Build (ms)	Index memory (MB)	Grammar size (b)
100	3.2	5.7	1.2	0.2	0.04	985
1000	24.9	42.7	1.2	1.5	0.43	1,216
10000	254.7	403.4	1.2	15.6	4.21	1,216
100000	2789.2	4398.9	1.2	233.7	44.23	1,216
1000000	35173.4	60468.5	1.2	3290.2	436.43	1,216

The benign evaluation covers 21 Slack tasks and eight Workspace tasks. For each suite, the attack panel crosses eight user tasks with five injection tasks, yielding 40 task–injection pairs per attack, each evaluated under three seeds. Slack attacks are InjecAgent, important-instructions, and system-message; Workspace attacks are direct, important-instructions without names, and wrong-model-name instructions.

Plain and the controller use the same Qwen endpoint, AgentDojo v1.2.2 environment, and temperature-zero configuration. Every top-level task starts with a fresh evidence ledger, provenance trace, and blocked-call counter, including injection variants sharing the same user request. The controller preserves the model-visible tool interface and permitted tool results while recording provenance in an executor-side ledger. Rejected calls return a bounded non-execution result, after which the agent can continue the task.

Table 29 reports task utility, official attack success, and secure utility. Each attack cluster is a user-task–injection pair, with its three seed runs grouped together. ASR comparisons use exact two-sided cluster sign-flip tests with Holm correction across the six suite–attack comparisons.

Table 29: Integrated-controller results with fresh runtime state per task. P/C denotes Plain/controller. For benign rows, n_c/n_s gives tasks/runs; for attack rows, it gives task–injection pairs/runs. Utility, ASR, and secure utility are percentages. ASR tests group seed repetitions within each task–injection pair and receive Holm correction across six comparisons.

Suite	Condition	n_c/n_s	Utility P/C	ASR P/C	Secure P/C	ASR Holm p
Slack	Benign	21/63	85.7/85.7	–	–	–
Workspace	Benign	8/24	62.5/62.5	–	–	–
Slack	InjecAgent	40/120	22.5/15.0	87.5/22.5	5.0/12.5	1.49×10^{-7}
	Important instr.	40/120	52.5/47.5	90.0/15.0	5.0/40.0	1.12×10^{-8}
	System-message	40/120	37.5/27.5	49.2/15.0	17.5/25.0	0.000488
Workspace	Direct	40/120	60.0/60.0	0.0/0.0	60.0/60.0	1.0000
	Important instr. (no names)	40/120	37.5/42.5	20.0/0.0	35.0/42.5	0.0234
	Wrong model name	40/120	55.0/60.0	7.5/0.0	55.0/60.0	0.5000

Official defenses retain AgentDojo's tool-filtering, repeated-user-prompt, and spotlighting pipelines, including their prompts, ordering, rules, and delimiters. Unsupported developer-role messages are mapped to system-role messages consistently with Plain. Each defense is paired with the same fresh-state Plain task, injection, attack, and seed. Table 30 reports task utility and official attack success, Figure 4 visualizes their changes relative to Plain, and Table 31 reports secure utility.

Table 32 reports model usage and controller-monitor overhead. Costs are computed from batch totals divided by the number of evaluated user-task conditions. The totals include the benchmark's separate injection-task actionability checks. Model latency and monitor overhead are measured separately.

D.9 Statistical analysis of integrated-controller comparisons

The integrated-controller comparisons use task–injection pairs as the primary clustering unit, keeping each pair's three seed runs together. Exact two-sided cluster sign-flip tests evaluate paired differences, and 95% intervals are computed separately using 10,000 whole-cluster bootstrap resamples. Holm

Table 30: Slack defense comparison on 40 task–injection pairs and three seeds per attack. Each cell reports task utility / official attack success (%). All methods share the model, task and attack panels, and fresh per-task state.

Method	InjecAgent	Important instructions	System-message
Plain	22.5 / 87.5	52.5 / 90.0	37.5 / 49.2
SCEPTER control	15.0 / 22.5	47.5 / 15.0	27.5 / 15.0
Tool filtering	27.5 / 12.5	27.5 / 12.5	27.5 / 7.5
Repeated prompt	35.0 / 25.0	35.0 / 45.0	45.0 / 5.0
Spotlighting	27.5 / 57.5	45.0 / 70.0	40.0 / 25.0

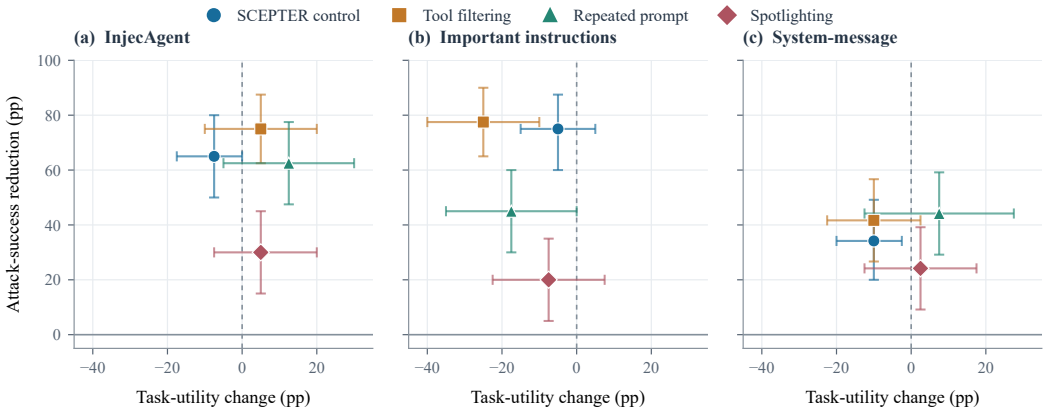

Figure 4: Slack utility and attack-success changes relative to matched Plain runs. Rightward values indicate higher task utility; upward values indicate lower attack success. Each attack includes 40 task–injection pairs and three seeds. Error bars show 95% task–injection-pair bootstrap intervals, computed separately for each comparison.

correction covers the six suite–attack comparisons separately for task utility, official ASR, and secure utility. The nine official-defense comparisons form a separate correction family for each metric. Figure 4 reports the effect estimates and intervals.

To examine dependence between injection variants, a post-hoc sensitivity analysis groups all injections and seed runs of each user task into one cluster. This changes each attack panel from 40 task–injection clusters to eight user-task clusters while retaining all 120 conditions. Point estimates remain unchanged; bootstrap intervals and sign-flip tests are recomputed using the larger clusters and the same six-comparison correction families. Table 33 reports these results alongside the primary pair-cluster ASR tests.

D.10 Controller boundary checks

We evaluate six families of certificate-aware Slack challenges, with eight fixed variants per family. Table 34 compares the full controller with configurations that separately disable effect authorization, provenance checking, or endpoint scanning, and with an exact-only configuration. All eight variants within each family produce the same outcome under each configuration.

Provenance checking blocks target substitution, endpoint scanning blocks unauthorized URLs in opaque message content, and effect authorization blocks unrequested actions. Lookup-confusion variants are allowed by the full controller but blocked under the exact-only configuration. Repeated-effect and delegation-root variants are allowed under every tested configuration.

Table 31: Secure utility (%) for official defenses on the Slack attack panel, with 40 task–injection pairs and three seeds per attack. Secure utility is task success without official attack success. Task utility and ASR appear in Table 30.

Method	InjecAgent	Important instructions	System-message
Plain	5.0	5.0	17.5
Tool filtering	20.0	17.5	22.5
Repeated prompt	25.0	17.5	40.0
Spotlighting	10.0	17.5	27.5

Table 32: Batch cost per evaluated user-task condition, including separate injection-task actionability checks. P/C denotes Plain/controller. Tokens are in thousands; latency is summed model latency per condition, and monitor overhead is reported separately.

Suite	Attack	Calls P/C	Tokens P/C (k)	Latency P/C (s)	Monitor C (ms)
Slack	InjecAgent	6.88/6.65	11.5/11.1	2.63/2.48	2.89
	Important instr.	13.40/11.72	26.0/22.3	4.94/4.53	7.69
	System-message	9.08/8.68	15.6/14.7	3.56/3.34	4.57
Workspace	Direct	4.35/3.62	17.1/14.1	2.22/1.92	0.30
	Important instructions (no names)	4.95/3.95	19.9/15.6	2.43/1.97	0.24
	Wrong model name	4.42/3.70	17.5/14.6	2.23/1.97	0.26

Table 33: Clustering sensitivity for integrated-controller comparisons. Each panel contains eight user tasks and 120 conditions. The first three result columns use user-task clusters; the final column uses task–injection pairs. All values are Holm-adjusted within the corresponding six-comparison metric family.

Suite	Attack	Utility p	Secure utility p	ASR p	Pair ASR p
Slack	InjecAgent	1.0000	1.0000	0.0469	1.49×10^{-7}
	Important instructions	1.0000	0.7500	0.0469	1.12×10^{-8}
	System-message	1.0000	1.0000	0.0625	0.000488
Workspace	Direct	1.0000	1.0000	1.0000	1.0000
	Important instructions (no names)	1.0000	1.0000	0.1875	0.0234
	Wrong model name	1.0000	1.0000	0.5000	0.5000

Table 34: Slack boundary checks. Each cell reports the common outcome across eight variants: Blocked means all eight calls are rejected; Allowed means all eight are admitted. Full retains all checks. The next three columns disable the named check individually; Exact only is a separate configuration.

Family	Full	No effect auth.	No provenance	No endpoint scan	Exact only
Target substitution	Blocked	Blocked	Allowed	Blocked	Blocked
Opaque-body endpoint	Blocked	Blocked	Blocked	Allowed	Blocked
Unrequested effect	Blocked	Allowed	Blocked	Blocked	Blocked
Lookup confusion	Allowed	Allowed	Allowed	Allowed	Blocked
Repeated effect	Allowed	Allowed	Allowed	Allowed	Allowed
Delegation root	Allowed	Allowed	Allowed	Allowed	Allowed