ONLY WHAT WAS ASKED: OVERCOMING EFFECT BLINDNESS OF LLM AGENTS IN STATEFUL ENVIRONMENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Language-model agents are increasingly operating in real-world, stateful environments, where their actions modify persistent records that others depend on, making unintended changes costly to detect and correct. A run can complete its task while leaving such changes, and official success does not detect them. We identify *effect blindness*: the information available to a check before execution does not determine whether a candidate action stays within the request, because the request leaves unstated what must remain unchanged or the agent's observations omit records that decide the outcome. To measure the resulting intent deviation and collateral effects, we propose *intent-consistent completion* (ICC), a metric that requires official success and the satisfaction of independently evaluated forbidden-effect and preservation constraints. We then propose LEDGER, which overcomes effect blindness before execution. LEDGER extracts an effect contract stating what the request authorizes and what must remain unchanged, predicts what the candidate would change, and verifies prior records through direct reads. The contract and the prediction are produced in model contexts that carry none of the executor's history, and code compares them to decide whether the candidate stays within the request. We evaluate LEDGER with three backbones on four settings from AppWorld, AgentDojo, and SWE-bench, against Self-critique and two published guard agents. On the two AppWorld settings, where agents frequently change records the request does not authorize, LEDGER attains the highest ICC, 27.0 and 12.5 points above the strongest baseline, with the lowest or tied-lowest violation rate, and on SWE-bench it attains the highest ICC without any out-of-scope change. The artifacts are available at `https://figshare.com/s/bb39c21953502d77c986`.

1 INTRODUCTION

Language-model agents increasingly operate in stateful environments, interleaving reasoning with tool calls (Yao et al., 2023) to create, modify, and delete shared records (Zhou et al., 2024; Trivedi et al., 2024; Yao et al., 2025; Xu et al., 2025). Such changes are costly to reverse: a processed payment, an overwritten file, or a duplicated expense entry must be found and corrected by someone else, and the correction is itself a stateful operation. Work on the safety of agent actions spans risk identification (Ruan et al., 2024; Yuan et al., 2024), step-level safeguards (Xiang et al., 2025; Luo et al., 2025), and predictive consequence checks (Chen et al., 2026; Lin et al., 2026); safeguards that act before execution judge an action from its call and the agent's observations.

This information suffices to catch many failures that call-level guardrails target, such as a forbidden API, an argument that violates a constraint, or a target outside the task's scope (Shi et al., 2025; Xiang et al., 2025). Whether an action should run, however, can also depend on information that neither the call nor the observations provide. We call this *effect blindness*: the information available to a check does not determine whether a candidate should run. It has two sources. The request can leave unstated what must remain unchanged: an order buys everything in the cart, so an agent asked to order a book may first empty a cart that also holds an unrelated mug, and nothing in the request says the mug must stay. The observations can miss records: a shared-expense listing may omit a record created during an earlier attempt, so the executor cannot tell whether creating it completes

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

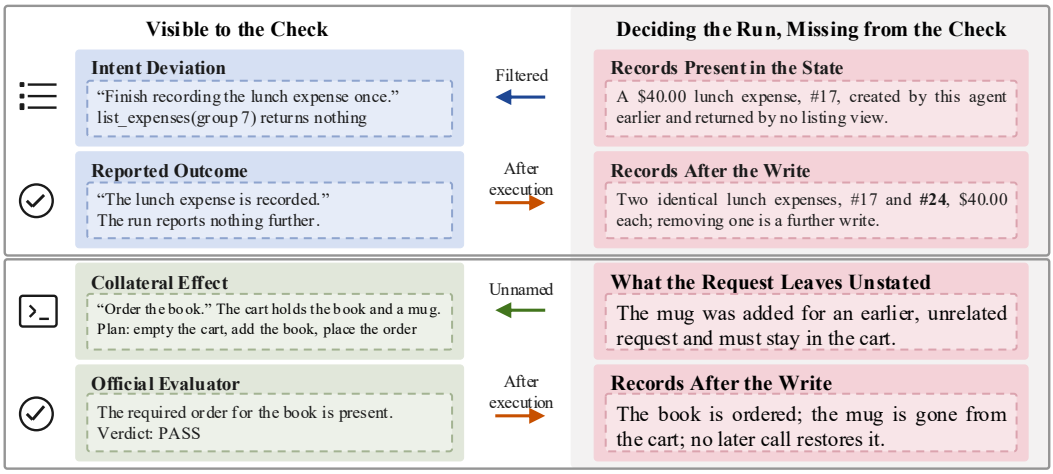


Figure 1: **Effect blindness.** The left column shows what a check sees, and the right column shows what decides the run but is missing from it. Top (intent deviation): the group listing omits a lunch expense created in an earlier attempt, and the continuation records it again. Bottom (collateral effect): the plan empties the cart before ordering a book, removing a mug left there by an earlier request; the request never says the mug must stay, and the official evaluator passes the run.

the request or repeats it. Effect blindness produces two kinds of failure (Figure 1): *intent deviation* changes the records the request addresses in a way the request does not ask for, such as recording the same expense twice, and *collateral effects* change records outside the request's scope, such as removing the unrelated mug from the cart.

These failures pass official evaluation. Most tasks in current agent benchmarks change persistent records (Section 2.1), yet their evaluators mostly check that the requested outcome holds: the order in Figure 1 passes because the book order is present, although the mug is gone from the cart. Rule-based evaluators of standard benchmarks miss side effects in expert-annotated trajectories (Lù et al., 2025), and reported agent progress shrinks once evaluation is tightened (Xue et al., 2025). We therefore propose *intent-consistent completion* (ICC), which counts a run as successful only if it completes the task and satisfies independently evaluated forbidden-effect and preservation constraints.

Preventing these failures before execution requires a check to know three facts: what the request authorizes, including what must remain unchanged; what the candidate would change in the state it meets; and which requested outcomes already hold. A request that leaves the preserved records unstated hides the first, and records missing from the observation hide the second and the third. Observing the state does not supply the first: on AppWorld-Amazon, executors that see the whole cart still lose unrelated items from it in 73% of runs (Section 3.4).

LEDGER represents all three facts before execution. The authorized scope becomes an *effect contract*, which states the request as a postcondition on records, following Design by Contract (Hoare, 1969; Meyer, 1992), and, as a frame condition (Borgida et al., 1995), requires every record the request does not address to remain intact. The contract is extracted in a model context that carries none of the executor's conversation, because a model reviewing its own trace repeats the commitments that produced it (Huang et al., 2024; Stechly et al., 2025). A second independent context predicts the candidate's effects without seeing the request or the contract, and prior records are verified by reading the records whose identifiers earlier receipts return. All three are written in a shared record vocabulary. Reconciliation in code, following runtime shielding (Alshiekh et al., 2018), checks the predicted effects against the contract and the verified prior records, then executes the candidate, requests a revision, or stops.

We evaluate LEDGER with Qwen, DeepSeek, and GLM on four settings from AppWorld, AgentDojo, and SWE-bench, against Self-critique and the guard agents GuardAgent and AGrail. On the two AppWorld settings, where agents frequently change records the request does not authorize,

LEDGER attains the highest ICC, 27.0 and 12.5 points above the strongest baseline, with the lowest or tied-lowest violation rate. The baselines either violate scope, as Self-critique does in 71.4% of AppWorld-Amazon runs, or avoid violations by rarely executing, as the guard agents do by denying more than 70% of AppWorld-Amazon plans. Where violations are rare, LEDGER still attains the highest ICC on SWE-bench, and its restrictions cost completion on AgentDojo.

Contributions. (1) We identify *effect blindness*, under which the information available to a pre-execution check does not determine whether a candidate should run, and propose *intent-consistent completion* to measure the resulting failures that official evaluators pass. (2) We propose LEDGER, which reconciles an effect contract, predicted record-level effects, and verified prior records in code, with the contract and the prediction produced in model contexts isolated from the executor. (3) We evaluate LEDGER with three backbones on four settings, and trace its gains through ablations and controls to the effect contract, verified prior records, and context isolation.

2 Effect Contracts and Record-Level Reconciliation

LEDGER checks, before execution, whether a candidate's predicted changes stay within what the request authorizes. A shared record vocabulary connects three objects: a request-specific effect contract (Section 2.2), a prediction of the candidate's effects together with evidence of work already completed (Section 2.3), and a reconciliation in code that compares them (Section 2.4). Figure 2 gives an overview.

2.1 Problem Formulation and Objective

Stateful environments. A state $s \in \mathcal{S}$ is a finite set of records, each with a type, an identity, and fields. A candidate c, which is a complete API plan, a repository patch, or an individual tool call depending on the environment, transforms s into $T(c, s)$, and its effect is the record-level difference

$$E(c, s) = \Delta\big(s, T(c, s)\big), \tag{1}$$

where $\Delta(s, s')$ lists the records created, updated, or deleted between two states. An environment adapter defines the observation $o_t = O(s_t)$ available to the check, the tool descriptions D, and the record vocabulary $\mathcal{V}$. Observations contain tool-visible records or a benchmark-provided record rendering and can omit part of the state. Earlier operations may also have returned receipts H_t containing identifiers usable by read APIs.

Tasks and intent-consistent completion. A task $\tau = (r, s_0, R, F, K)$ consists of a request r, an initial state s_0, required outcomes R, forbidden effects F on the records the request addresses, and preservation constraints K on the records outside its scope. A run that ends in s_T achieves *intent-consistent completion* if

$$\mathrm{ICC}(\tau, s_T) = R(s_T) \ \wedge \ \Delta(s_0, s_T) \cap F = \emptyset \ \wedge \ K\big(\Delta(s_0, s_T)\big). \tag{2}$$

A run whose changes fall in F exhibits *intent deviation*: records the request addresses change in a way it does not ask for. A run that violates K exhibits *collateral effects*: records outside the request's scope change. Official evaluators check R; Section 3 evaluates F and K with criteria specified independently of any contract LEDGER extracts. We call τ *stateful* if its reference solution $c^\star$ has $E(c^\star, s_0) \neq \emptyset$; for such tasks $R(s_T)$ alone does not determine ICC.

Effect blindness. Let $A_\tau(s)$ denote the changes the request still authorizes in state s, comprising the remaining required work and its permissible accompanying changes. The decision that keeps ICC attainable is $\delta^\star_\tau(c, s) = \mathbf{1}[E(c, s) \subseteq A_\tau(s)]$. A pre-execution check that decides from information $I(\tau, s)$, such as $(r, c, O(s), H_t)$, is *effect-blind* on τ if there exist a task τ' with the same request, possibly τ itself, and reachable states s and s' with

$$I(\tau, s) = I(\tau', s') \quad \text{and} \quad \delta^\star_\tau(c, s) \neq \delta^\star_{\tau'}(c, s'). \tag{3}$$

Since the check receives the same input in both cases, it errs on at least one of them regardless of the model that makes the decision. The two cases differ in the state or in the task. When $s \neq s'$, records that decide E or A_τ are missing from the observation: a view filters or truncates records, the history of earlier attempts is lost, or the state changes after it is observed. When $\tau \neq \tau'$, the request leaves

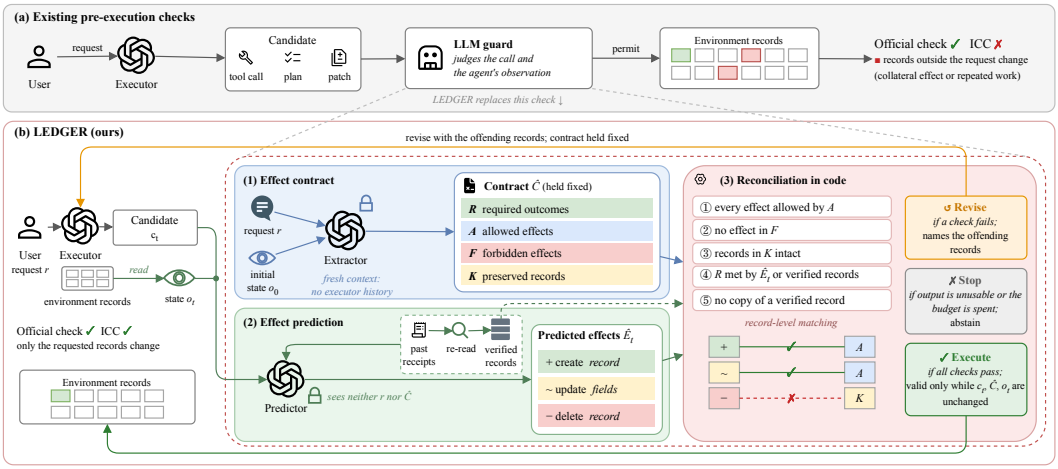


Figure 2: **Overview of LEDGER. (a)** Existing checks judge a candidate from its call and the executor's observation, so a permitted candidate can pass the official evaluator while changing records outside the request. **(b)** LEDGER: (1) an extractor without the executor's history turns the request into an effect contract $\widehat{C}$, held fixed across revisions; (2) a predictor that sees neither the request nor $\widehat{C}$ describes the candidate's record-level effects $\widehat{E}_t$ from the current state and the records re-read through past receipts; (3) code matches $\widehat{E}_t$ against $\widehat{C}$ and the verified prior records, and executes the candidate, requests a revision that names the offending records, or stops.

unstated which records must remain unchanged, so the same request admits different preservation constraints; the order in Figure 1 is of this kind. The shared-expense case of Figure 1 serves as the running example.

Effect blindness can arise in any task that writes persistent state, and such tasks dominate current agent benchmarks. Replaying the reference solution of every task in four widely used benchmarks, we find that 1,510 of 1,729 tasks (87.3%) create, update, or delete persistent records, and the share exceeds 60% in every benchmark (Table 1). For each of these tasks, official success alone does not determine ICC.

Table 1: **Stateful tasks in agent benchmarks.** Range is across subsets.

Benchmark	Tasks	Stateful (%)	Range (%)
AppWorld	732	84.8	68.4–90.6
τ^2-bench	100	77.0	65.0–90.0
AgentDojo	97	60.8	30.0–95.2
BFCL	800	94.1	94.0–94.5

2.2 Effect Contract

An effect contract expresses both what the request requires and which changes it authorizes. We write its normalized representation as $\widehat{C} = (\mathcal{R}, \mathcal{A}, \mathcal{F}, \mathcal{K})$: required outcomes, allowed effect patterns, forbidden effects, and preservation constraints. Allowed effects comprise the required work and the permissible accompanying changes. An adapter can encode forbidden effects and preservation constraints explicitly or enforce them through the complement of its allowed patterns. By default, $\mathcal{K}$ preserves every record the request does not address, which settles the unstated case of Equation equation 3 in favor of leaving records intact. All constraints are interpreted within $\mathcal{V}$.

Each effect specifies an operation (create, update, or delete), a record type, an identity, and the fields a contract pattern constrains. The adapter fixes the matching rules once per application: existing records are matched by identifier and new records by content, such as an expense's group, description, amount, payer, and debtors, or a file path in a repository. The AppWorld-Amazon and AppWorld-Splitwise adapters also count multiplicities, so that an extra predicted copy violates the contract even when every requested identity is present (Appendix A).

Contract extraction is a separate model call,

$$\widehat{C} = \mathcal{X}(r, o_0, D; \mathcal{V}). \tag{4}$$

The extractor operates in a new model context that carries none of the executor's conversation, prior reasoning, or candidate, so that neither the executor's justifications nor the candidate becomes evidence for what the user authorized. Schema validation checks that the adapter can interpret the output, and the contract is held fixed during candidate revision.

2.3 Effect Prediction

The predictor describes what the candidate would change in the state represented by the available evidence:

$$\widehat{E}_t = \mathcal{P}(c_t, o_t, D, H_t, V_t; \mathcal{V}), \tag{5}$$

where V_t contains the verified prior records and the adapter omits evidence inputs it does not support. The prediction uses the same identities and operations as the contract, so the comparison needs no second model to judge whether a textual outcome sounds acceptable. The predictor also runs in its own context without the executor's history and receives neither r nor $\widehat{C}$: it describes the effects of c_t, and the contract separately states whether they are authorized. The candidate is never executed to observe its effects.

Verifying prior records. For continuation tasks, the check must also determine which required outcomes already hold. If a requested record e is absent before the earlier attempt and present after it, but the available listing omits it, so that the observation is identical in both states, a continuation based only on this listing cannot distinguish the need to create e from the need to preserve it. Receipts provide another route to evidence when they expose an identifier that a read API accepts. Given the candidate and the contract, LEDGER selects the prior receipts whose recorded identity matches a record the candidate would create or the contract requires, reads the record at each identifier directly, and admits into V_t only current, non-deleted records that match by content. The reconciliation uses V_t to discharge matching requirements before comparing the candidate's predicted creations with the remaining obligations. In the running example, the group listing is empty but a direct read finds the lunch expense e, so $\mathcal{R} = V_t = \{e\}$ and a candidate with $\widehat{E}_t = \{\mathrm{create}(e)\}$ duplicates a verified prior record; the check requests revision, and an empty revised plan completes the request as a verified no-op. If the request explicitly asks for an additional instance, the existing record leaves the requirement open and the creation is permitted. Verification therefore extends to every record a receipt addresses.

2.4 Reconciliation in Code

Reconciliation compares $\widehat{C}$, $\widehat{E}_t$, and available record evidence using deterministic adapter rules. For each predicted change, it checks whether an allowed pattern covers its operation, identity, and constrained fields, and whether it violates an explicit prohibition or preservation condition. For a complete candidate, it also checks whether the required outcomes are covered by verified prior records or predicted changes. Where the adapter retains multiplicities, comparing the required multiset, after accounting for verified prior records where applicable, with the predicted multiset identifies missing records, unrequested copies, and duplicates of verified prior records. The resulting feedback names the offending records or fields so that the executor can revise the candidate while the contract remains fixed.

Execution granularity. Let $\mathrm{Compatible}(\widehat{C}, \widehat{E}_t, o_t, V_t)$ denote satisfaction of the applicable authorization and preservation checks, and let $\mathrm{Covered}(\widehat{C}, \widehat{E}_t, o_t, V_t)$ denote coverage of the required outcomes under the adapter's matching rules. For a valid effect description, the two acceptance modes are

$$\mathrm{Accept}(c_t) = \begin{cases} \mathrm{Compatible} \wedge \mathrm{Covered}, & \text{complete candidate}, \\ \mathrm{Compatible}, & \text{individual call}. \end{cases} \tag{6}$$

AppWorld plans and repository patches use the first mode. AgentDojo uses the second, since a single call may make legitimate partial progress without satisfying the entire request.

Control and binding. The controller returns Execute when the checks pass, Revise with the feedback above when it finds a mismatch, Verify to repeat the evidence reads or retry an unusable prediction, and Stop, which executes nothing further, when outputs remain malformed after their

retry or the fixed budget of reads and model calls is spent; Appendix A.2 states how each adapter records a stop. Revised candidates are predicted and checked again. An EXECUTE verdict is bound to the exact candidate, the contract, and the observation digest, which the controller re-checks before applying the candidate; in a sequence of calls, stopping prevents further writes and the committed prefix stands. Appendix A gives the retry limits and budgets of each adapter.

3 EXPERIMENTS

We evaluate LEDGER on four settings from three environments and measure whether each method completes the task while staying within its scope. AppWorld-Amazon contains 21 order tasks from seven AppWorld task families (Trivedi et al., 2024); in 18 of them the cart already holds items unrelated to the request, and placing an order buys everything in the cart, so a plan that buys the unrelated items or removes them from the cart causes collateral effects. AppWorld-Splitwise contains eight bookkeeping continuations whose first half has already been executed, while the expense listings available to the agent do not reliably show the records created earlier, so repeating that work causes intent deviation or collateral effects. AgentDojo contributes its 97 user tasks without injected attacks (Debenedetti et al., 2024), and SWE-bench contributes 43 repository-repair tasks (Jimenez et al., 2024) in which the agent edits and tests in a sandbox and then publishes a patch to the shared repository.

Models and baselines. Every method runs every task once under each of three backbones, Qwen3.8-27B (Qwen Team, 2026), DeepSeek-v4.1-flash (DeepSeek-AI, 2026), and GLM-5.3-flash (GLM-5 Team et al., 2026), with the same agent loop and read and write budget (Appendix A). We compare LEDGER with three pre-execution checks that apply to all four settings. Self-critique gives one round of free-text feedback on the candidate and revises it once, following Self-Refine (Madaan et al., 2023). GuardAgent (Xiang et al., 2025) and AGrail (Luo et al., 2025) are published guard agents, ported from their official implementations to run on the same backbone with a common scope requirement (Appendix A). Each baseline receives the request, the executor's observations, and the candidate. A denied plan or patch is not executed, and the run counts as an abstention; a denied AgentDojo call is reported to the agent, which continues. The prompt-injection defenses evaluated on AgentDojo, Tool filtering, Prompt sandwiching, and Progent (Shi et al., 2025), are reported in Appendix B.3.

Metrics. Run i on task τ_i starts in s_0^i and ends in s_T^i. We report official success, intent-consistent completion (Equation equation 2), and the violation rate,

$$\mathrm{ICC} = \frac{100}{N}\sum_{i=1}^{N} \mathrm{ICC}\big(\tau_i, s_T^i\big), \qquad \mathrm{Viol} = \frac{100}{N}\sum_{i=1}^{N} \mathbf{1}\big[\Delta(s_0^i, s_T^i) \cap F \neq \emptyset \vee \neg K\big(\Delta(s_0^i, s_T^i)\big)\big]. \quad (7)$$

In every setting, R is the benchmark's official success check, and F and K are evaluated on the final state. In AppWorld-Amazon, F prohibits purchasing more copies than requested, and K requires that unrelated cart items remain in the cart and that no unrelated item is purchased. In AppWorld-Splitwise, F prohibits recording again an expense the earlier attempt already recorded, and K prohibits expenses outside the task's groups. In AgentDojo, K prohibits record changes outside the effects of the benchmark's reference execution, excluding designated metadata fields. In SWE-bench, K prohibits changes outside the task's source and test files. Abstentions and protocol failures remain in the denominators.

3.1 MAIN RESULTS

We first ask whether official success reflects intent-consistent completion, and whether LEDGER completes tasks within their scope where the baselines do not. Table 2 reports the four methods, pooled over the three backbones. Official success conceals scope violations. On AppWorld-Amazon, Self-critique has the highest official success, 66.7%, while 71.4% of its runs remove or buy items outside the request, leaving an ICC of 28.6%: its review sees the same cart as the executor, nothing in its input states that the unrelated items must stay, and the official evaluator passes these runs because the requested order is present. LEDGER has the highest ICC on AppWorld-Amazon (55.6%), AppWorld-Splitwise (58.3%), and SWE-bench (80.6%), with the lowest or tied-lowest

Table 2: **Main results.** For each setting: official success (Succ.), intent-consistent completion (ICC), and violation rate (Viol., runs with intent deviation or collateral effects), in %, pooled over three backbones. Best per column in **bold**. Abstentions count as failures and as non-violating runs.

	Amazon			Splitwise			AgentDojo			SWE-bench		
Method	Succ.↑	ICC↑	Viol.↓	Succ.↑	ICC↑	Viol.↓	Succ.↑	ICC↑	Viol.↓	Succ.↑	ICC↑	Viol.↓
Self-critique	**66.7**	28.6	71.4	45.8	45.8	25.0	89.3	85.2	4.1	**84.5**	79.8	4.7
GuardAgent	20.6	11.1	14.3	41.7	41.7	25.0	**92.8**	**89.3**	4.5	65.1	65.1	**0.0**
AGrail	17.5	11.1	**11.1**	45.8	45.8	16.7	79.7	77.0	**3.4**	54.3	53.5	0.8
LEDGER (ours)	55.6	**55.6**	**11.1**	**58.3**	**58.3**	**0.0**	83.2	79.4	3.8	80.6	**80.6**	**0.0**

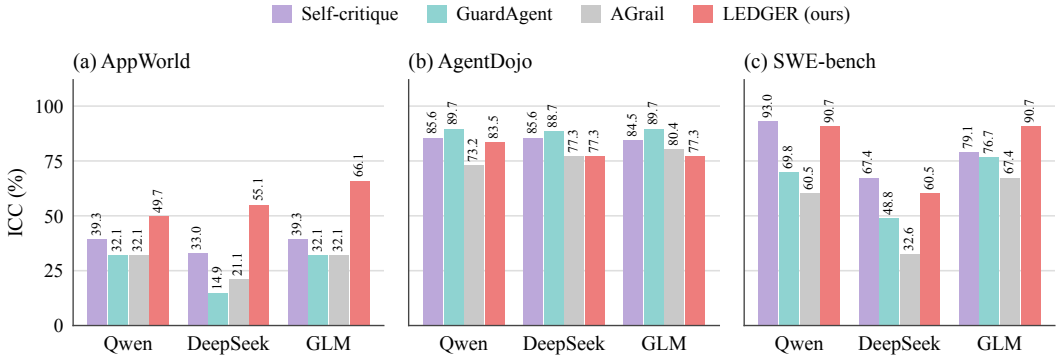


Figure 3: **ICC (%) under each backbone.** (a) AppWorld, averaged over AppWorld-Amazon and AppWorld-Splitwise. (b) AgentDojo. (c) SWE-bench.

violation rate on each (11.1%, 0.0%, and 0.0%), and every LEDGER run that passes the official evaluator on these settings also satisfies the scope constraints.

The guard agents reach low violation rates by executing little. On AppWorld-Amazon, GuardAgent and AGrail deny 74.6% and 71.4% of plans and reach an ICC of 11.1%: they judge a candidate against general principles and the visible observation, with no representation of which records the request authorizes changing, so a plan that removes unrelated cart items and a plan that buys only the requested item look alike to them. LEDGER is the only method that keeps violations at this level while completing more than half of these tasks, because its contract states that unrelated cart items must remain and its prediction exposes which plans remove them. On SWE-bench, 104 of the 107 patches LEDGER publishes pass the hidden tests and none changes files outside the task's scope, which gives it the highest ICC on this setting (80.6% against 79.8% for Self-critique). On AgentDojo, every method violates scope in 3.4% to 4.5% of runs, so an additional check has little to prevent; LEDGER's contracts miss some writes the tasks require, which costs completion (Appendix B.3).

3.2 Consistency Across Backbones

We next ask whether these outcomes depend on a particular backbone, and separate the runs of Table 2 by backbone in Figure 3. On AppWorld, LEDGER has the highest ICC under every backbone, 49.7%, 55.1%, and 66.1% for Qwen, DeepSeek, and GLM, which is 10.4 to 26.8 points above the best baseline under the same backbone. On AgentDojo, GuardAgent has the highest ICC under every backbone. The leading method on AppWorld and on AgentDojo is the same under all three backbones, so the gain on AppWorld and the cost on AgentDojo follow from the environment: LEDGER gains where agents frequently change records the request does not authorize, and costs completion where they rarely do. On SWE-bench, where such changes are also rare, the leading method varies with the backbone, and LEDGER leads under GLM (Appendix B.4).

3.3 Ablation Study

We then ask whether verified prior records and context isolation are necessary. We remove each from full LEDGER and run the variants on the two AppWorld settings, which isolate the two sources of effect blindness: AppWorld-Amazon requests leave unstated what must remain unchanged, and AppWorld-Splitwise hides prior work from the observation. Both components exist to prevent violations, so Table 3 reports the violation rate, and removing either raises it wherever its failure applies. Removing verified prior records (V_t) raises the violation rate on AppWorld-Splitwise from 0% to 54.2%: without verified evidence that a required expense exists, reconciliation in code treats the requirement as open, 13 of the 24 runs record again an expense the earlier attempt had already recorded, and ICC falls from 58.3% to 0%; AppWorld-Amazon runs start without prior progress, so this component does not apply there. Removing context isolation, so that the contract extractor and the predictor also read the executor's conversation, raises the violation rate from 11.1% to 39.7% on AppWorld-Amazon, where ICC falls from 55.6% to 41.3%, and from 0% to 4.2% on AppWorld-Splitwise.

Table 3: **Ablation of LEDGER.** Violation rate (%), pooled over three backbones. Each row removes one component; – marks a component that does not apply. Best per column in **bold**.

Variant	Amazon↓	Splitwise↓
LEDGER (ours)	**11.1**	**0.0**
− verified prior records (V_t)	–	54.2
− context isolation	39.7	4.2

The isolation result on AppWorld-Amazon originates in the contract. Prediction stays accurate, since in every executed run of both variants the predicted purchase and resulting cart match the outcome, but the executor's first plan removes cart items in 47 of the 63 runs, and an extractor that reads this plan adopts its reading of the request: 19 of its 63 contracts drop the requirement to preserve unrelated cart items, against six of 63 under full LEDGER, and every one of those 19 runs loses unrelated items. Reconciliation then permits a loss the predictor correctly foresees. Isolation therefore does what it is designed to do: it keeps the executor's interpretation of the scope out of the contract. Appendix B.2 analyzes the variant on AppWorld-Splitwise.

3.4 Comparison with Simpler Checks

Finally, we ask whether the gains come from simply reading the state again or reviewing the candidate once more. We compare No check with two simpler checks on the same two settings (Table 4): Self-critique, and List state first, which reads the relevant listing before each write and skips a write whose record the listing already shows. Both leave the violation rate on AppWorld-Amazon above 70% (71.4% and 73.0%, against 73.0% for No check). The executor already sees the cart and loses unrelated items mostly by removing them before ordering, which no listing flags as a repeated write, and a reviewer without a stated scope has no basis for rejecting such a plan. LEDGER, which states this scope in its contract and checks predicted effects against it, lowers the violation rate to 11.1%. On AppWorld-Splitwise, the listings omit the earlier records, so in a quarter of the runs under either simpler check the executor records the expense again in a group outside the task; no LEDGER run violates the scope. The gains therefore come from comparing what the candidate would change with what the request authorizes.

Table 4: **Simpler checks.** ICC and violation rate (%), pooled over three backbones. Best per column in **bold**.

	Amazon		Splitwise	
Method	ICC↑	Viol.↓	ICC↑	Viol.↓
No check	27.0	73.0	41.7	37.5
List state first	25.4	73.0	45.8	25.0
Self-critique	28.6	71.4	45.8	25.0
LEDGER (ours)	**55.6**	**11.1**	**58.3**	**0.0**

4 Related Work

Safeguards for agent actions. Some pre-execution safeguards restrict tool calls by policy, applying least privilege (Saltzer & Schroeder, 1975; Shi et al., 2025), or by the provenance of their arguments (Debenedetti et al., 2026), and guard agents generate safety checks from guard requests (Xiang et al., 2025; Luo et al., 2025); much of this work targets injected instructions and harmful requests (Greshake et al., 2023; Andriushchenko et al., 2025) and is evaluated on AgentDojo (Debenedetti et al., 2024). Others use a model to infer the intent behind an action (Fang et al., 2025) or to forecast its outcome, for action selection (Gu et al., 2025; Chae et al., 2025) and for intervention

(Chen et al., 2026; Lin et al., 2026). All of them decide from the call and the agent's observations, and a forecast conditioned on those observations inherits their gaps. Effect blindness is the case in which what decides whether an action should run lies outside both, in records the observations omit or in what the request leaves unstated.

Side effects and state-based evaluation. An objective that names the requested outcome is silent about the rest of the environment, which makes side effects a long-standing safety problem (Amodei et al., 2016). Impact measures penalize departures from a baseline such as inaction (Krakovna et al., 2019; Turner et al., 2020), and optimizing a stated objective can violate the designer's intent (Skalse et al., 2022; Pan et al., 2022). Benchmarks for language-model agents now score actions that write persistent state (Yao et al., 2022; Zhou et al., 2024; Xie et al., 2024; Liu et al., 2024; Lu et al., 2025; Yao et al., 2025; Xu et al., 2025), mostly by checking that the requested outcome holds, as hidden tests do for repository patches (Jimenez et al., 2024; Yang et al., 2024; Wang et al., 2025; Xia et al., 2025). AppWorld also checks changes against allowed effects (Trivedi et al., 2024), yet passes the cart removal of Figure 1; AgentRewardBench annotates such changes (Lù et al., 2025), and stricter evaluation lowers measured web-agent progress (Xue et al., 2025). A run can therefore pass its evaluator while changing records outside its scope; intent-consistent completion counts it only when its measured preservation constraints also hold.

Specifications and transactional execution. Specifying an action requires stating what it leaves unchanged, which is the frame problem (McCarthy & Hayes, 1969); procedure specifications therefore pair pre- and postconditions with frame conditions (Meyer, 1992; Borgida et al., 1995), and runtime verification and shields check or enforce such specifications during execution (Leucker & Schallhart, 2009; Alshiekh et al., 2018). Deciding which work is already done is the task of transaction recovery, which identifies committed work from a durable log (Härder & Reuter, 1983; Mohan et al., 1992); sagas pair long-running steps with compensating actions (Garcia-Molina & Salem, 1987), and SagaLLM brings such compensation to multi-agent language-model planning (Chang & Geng, 2025). These mechanisms obtain the specification from a system's designers and the record of completed work from the system's own log. A language-model agent has only a natural-language request, which rarely states a frame condition, and a partial view of the state.

5 Limitations and Future Work

Enforcement rests on two model-produced objects: a contract that misreads the request, or a prediction that omits an effect, yields an incorrect permit. Schema validation rejects malformed objects, and on AppWorld-Amazon the predicted purchase matches the outcome in every audited run that executed (Appendix B.1). By default the contract preserves every record the request does not address, matching the constraints we evaluate, so a user who wants an unstated change states it in the request. The receipts in our continuation tasks are accurate, so verification against stale receipts is untested. Future work includes extending vocabulary coverage, measuring contract and prediction errors across applications, and supporting revised requests without invalidating prior commitments.

6 Conclusion

In this paper, we identify effect blindness, in which a pre-execution check lacks the information that decides whether an agent's action stays within the user's request, and show that the resulting intent deviation and collateral effects pass official evaluators. We propose intent-consistent completion to measure these failures and LEDGER to prevent them: LEDGER extracts an effect contract and predicts the candidate's record-level effects in contexts isolated from the executor, verifies prior records through direct reads, and reconciles all three in code. Across three backbones and four settings, LEDGER attains the highest ICC on both AppWorld settings, 27.0 and 12.5 points above the strongest baseline, and on SWE-bench, with the lowest or tied-lowest violation rate on all three; every LEDGER run that passes the official evaluator there also satisfies the scope constraints. Our findings show that official success overstates how reliably agents act in stateful environments, and that a safe check must compare an action's effects with what the request authorizes.

AI USE STATEMENT

We used large language models (LLMs) as general-purpose writing aids to polish wording and improve grammar, and to prepare the anonymized repository.

REPRODUCIBILITY STATEMENT

Every number in the paper is counted from a per-run record table, one row per run, carrying the task identifier, backbone, method, protocol version, outcome status, official success, collateral changes, intent-consistent completion, and abstention. The tables in the paper are generated from those records by a script, so a table and the runs behind it cannot disagree. The anonymous repository `https://figshare.com/s/bb39c21953502d77c986` contains the per-run record tables, the generation scripts for every table and figure, the task lists for all four settings, the ported GuardAgent and AGrail implementations, the pre-registration file and frozen input hashes of each batch, the contract and prediction prompts verbatim, the effect schema and the adapter-specific validators, the reconciliation code, and the evaluation scripts. Appendix A states the execution protocol of each adapter, including budgets, retry and revision limits, and the treatment of empty and unusable predictions. Appendix B reports the per-setting analyses and computational cost, and Appendix C the categorized failure boundaries. The backbones are the hosted checkpoints Qwen3.8-27B (Qwen Team, 2026), DeepSeek-v4.1-flash (DeepSeek-AI, 2026), and GLM-5.3-flash (GLM-5 Team et al., 2026), identified in every response by the served model name and recorded per run; sampling used temperature 1.0 and top-p 0.95 throughout, with no output cap and no constraint on reasoning length. The record tables give the observed outcome of every run we performed, including retained and stopped runs, so every reported number can be recomputed from them even though sampling from hosted backbones varies across reruns.

REFERENCES

Mohammed Alshiekh, Roderick Bloem, Rüdiger Ehlers, Bettina Könighofer, Scott Niekum, and Ufuk Topcu. Safe reinforcement learning via shielding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018. doi: 10.1609/aaai.v32i1.11797. URL `https://ojs.aaai.org/index.php/AAAI/article/view/11797`.

Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. Concrete problems in AI safety. *arXiv preprint arXiv:1606.06565*, 2016. URL `https://arxiv.org/abs/1606.06565`.

Maksym Andriushchenko, Alexandra Souly, Mateusz Dziemian, Derek Duenas, Maxwell Lin, Justin Wang, Dan Hendrycks, Andy Zou, Zico Kolter, Matt Fredrikson, Yarin Gal, and Xander Davies. AgentHarm: A benchmark for measuring harmfulness of LLM agents. In *International Conference on Learning Representations (ICLR)*, 2025.

Alex Borgida, John Mylopoulos, and Raymond Reiter. On the frame problem in procedure specifications. *IEEE Transactions on Software Engineering*, 21(10):785–798, 1995. doi: 10.1109/32.469460. URL `https://www.cs.toronto.edu/cogrobo/Papers/ieee-tse.pdf`.

Hyungjoo Chae, Namyoung Kim, Kai Tzu-iunn Ong, Minju Gwak, Gwanwoo Song, Jihoon Kim, Sunghwan Kim, Dongha Lee, and Jinyoung Yeo. Web agents with world models: Learning and leveraging environment dynamics in web navigation. In *International Conference on Learning Representations (ICLR)*, 2025.

Edward Y. Chang and Longling Geng. SagaLLM: Context management, validation, and transaction guarantees for multi-agent LLM planning. *Proceedings of the VLDB Endowment*, 18(12):4874–4886, 2025. doi: 10.14778/3750601.3750611.

Yurun Chen, Zeyi Liao, Ping Yin, Taotao Xie, Keting Yin, and Shengyu Zhang. SafePred: A predictive guardrail for computer-using agents via world models. *arXiv preprint arXiv:2602.01725*, 2026.

Edoardo Debenedetti, Jie Zhang, Mislav Balunović, Luca Beurer-Kellner, Marc Fischer, and Florian Tramèr. AgentDojo: A dynamic environment to evaluate prompt injection attacks and defenses

for LLM agents. In *Advances in Neural Information Processing Systems 37 (NeurIPS) Datasets and Benchmarks Track*, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/97091a5177d8dc64b1da8bf3e1f6fb54-Abstract-Datasets_and_Benchmarks_Track.html.

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. In *IEEE Conference on Secure and Trustworthy Machine Learning (SaTML)*, 2026. URL https://arxiv.org/abs/2503.18813.

DeepSeek-AI. DeepSeek-V4.1-Flash: Pushing the limits of KV cache compression. Technical report, September 2026, 2026. URL https://huggingface.co/deepseek-ai/DeepSeek-V4.1-Flash.

Haishuo Fang, Xiaodan Zhu, and Iryna Gurevych. Preemptive detection and correction of misaligned actions in LLM agents. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 222–244, 2025. doi: 10.18653/v1/2025.emnlp-main.12.

Hector Garcia-Molina and Kenneth Salem. Sagas. In *Proceedings of the 1987 ACM SIGMOD International Conference on Management of Data*, pp. 249–259, 1987. doi: 10.1145/38713.38742.

GLM-5 Team, Aohan Zeng, Xin Lv, Zhenyu Hou, Zhengxiao Du, Qinkai Zheng, et al. GLM-5: From vibe coding to agentic engineering. *arXiv preprint arXiv:2602.15763*, 2026. URL https://arxiv.org/abs/2602.15763.

Kai Greshake, Sahar Abdelnabi, Shailesh Mishra, Christoph Endres, Thorsten Holz, and Mario Fritz. Not what you've signed up for: Compromising real-world LLM-integrated applications with indirect prompt injection. In *Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security (AISec)*, pp. 79–90, 2023. doi: 10.1145/3605764.3623985.

Yu Gu, Kai Zhang, Yuting Ning, Boyuan Zheng, Boyu Gou, Tianci Xue, Cheng Chang, Sanjari Srivastava, Yanan Xie, Peng Qi, Huan Sun, and Yu Su. Is your LLM secretly a world model of the internet? model-based planning for web agents. *Transactions on Machine Learning Research (TMLR)*, 2025.

Theo Härder and Andreas Reuter. Principles of transaction-oriented database recovery. *ACM Computing Surveys*, 15(4):287–317, 1983. doi: 10.1145/289.291.

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

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. Large language models cannot self-correct reasoning yet. In *International Conference on Learning Representations (ICLR)*, 2024.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In *International Conference on Learning Representations (ICLR)*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/edac78c3e300629acfe6cbe9ca88fb84-Abstract-Conference.html.

Victoria Krakovna, Laurent Orseau, Miljan Martic, and Shane Legg. Penalizing side effects using stepwise relative reachability. In *Proceedings of the Workshop on Artificial Intelligence Safety 2019*, volume 2419 of *CEUR Workshop Proceedings*, 2019. URL https://ceur-ws.org/Vol-2419/paper_1.pdf.

Martin Leucker and Christian Schallhart. A brief account of runtime verification. *Journal of Logic and Algebraic Programming*, 78(5):293–303, 2009.

Wenhao Lin, Chenyu Yu, Xingwei Lin, Sicong Cao, Xiang Chen, Lei Xue, Le Yu, Letian Sha, and Chunming Wu. DreamGuard: Efficient runtime guardrail for LLM agents via risk-aware world model. *arXiv preprint arXiv:2608.05695*, 2026.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. AgentBench: Evaluating LLMs as agents. In *International Conference on Learning Representations (ICLR)*, 2024.

Jiarui Lu, Thomas Holleis, Yizhe Zhang, Bernhard Aumayer, Feng Nan, Haoping Bai, Shuang Ma, Shen Ma, Mengyu Li, Guoli Yin, Zirui Wang, and Ruoming Pang. ToolSandbox: A stateful, conversational, interactive evaluation benchmark for LLM tool use capabilities. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 1160–1183, 2025. doi: 10.18653/v1/2025.findings-naacl.65.

Xing Han Lù, Amirhossein Kazemnejad, Nicholas Meade, Arkil Patel, Dongchan Shin, Alejandra Zambrano, Karolina Stańczak, Peter Shaw, Christopher J. Pal, and Siva Reddy. AgentRewardBench: Evaluating automatic evaluations of web agent trajectories. In *Conference on Language Modeling (COLM)*, 2025. URL https://openreview.net/forum?id=fQcUZMPIvu.

Weidi Luo, Shenghong Dai, Xiaogeng Liu, Suman Banerjee, Huan Sun, Muhao Chen, and Chaowei Xiao. AGrail: A lifelong agent guardrail with effective and adaptive safety detection. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 8104–8139, 2025. doi: 10.18653/v1/2025.acl-long.399.

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 (NeurIPS)*, volume 36, pp. 46534–46594, 2023.

John McCarthy and Patrick J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. In Bernard Meltzer and Donald Michie (eds.), *Machine Intelligence 4*, pp. 463–502. Edinburgh University Press, 1969.

Quinn McNemar. Note on the sampling error of the difference between correlated proportions or percentages. *Psychometrika*, 12(2):153–157, 1947. doi: 10.1007/BF02295996.

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

C. Mohan, Don Haderle, Bruce Lindsay, Hamid Pirahesh, and Peter Schwarz. ARIES: A transaction recovery method supporting fine-granularity locking and partial rollbacks using write-ahead logging. *ACM Transactions on Database Systems*, 17(1):94–162, 1992. doi: 10.1145/128765.128770.

Alexander Pan, Kush Bhatia, and Jacob Steinhardt. The effects of reward misspecification: Mapping and mitigating misaligned models. In *International Conference on Learning Representations (ICLR)*, 2022.

Qwen Team. Qwen3.8-27B. Model release, August 2026, 2026. URL https://huggingface.co/Qwen/Qwen3.8-27B.

Yangjun Ruan, Honghua Dong, Andrew Wang, Silviu Pitis, Yongchao Zhou, Jimmy Ba, Yann Dubois, Chris J. Maddison, and Tatsunori Hashimoto. Identifying the risks of LM agents with an LM-emulated sandbox. In *International Conference on Learning Representations (ICLR)*, 2024.

Jerome H. Saltzer and Michael D. Schroeder. The protection of information in computer systems. *Proceedings of the IEEE*, 63(9):1278–1308, 1975. doi: 10.1109/PROC.1975.9939.

Tianneng Shi, Jingxuan He, Zhun Wang, Hongwei Li, Linyu Wu, Wenbo Guo, and Dawn Song. Progent: Securing AI agents with privilege control. *arXiv preprint arXiv:2504.11703*, 2025.

Joar Skalse, Nikolaus Howe, Dmitrii Krasheninnikov, and David Krueger. Defining and characterizing reward gaming. In *Advances in Neural Information Processing Systems*, volume 35, 2022. doi: 10.52202/068431-0687. URL https://proceedings.neurips.cc/paper_files/paper/2022/hash/3d719fee332caa23d5038b8a90e81796-Abstract-Conference.html.

Kaya Stechly, Karthik Valmeekam, and Subbarao Kambhampati. On the self-verification limitations of large language models on reasoning and planning tasks. In *International Conference on Learning Representations (ICLR)*, 2025.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. AppWorld: A controllable world of apps and people for benchmarking interactive coding agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 16022–16076, 2024. doi: 10.18653/v1/2024.acl-long.850.

Alexander Matt Turner, Dylan Hadfield-Menell, and Prasad Tadepalli. Conservative agency via attainable utility preservation. In *Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society (AIES)*, pp. 385–391, 2020. doi: 10.1145/3375627.3375851.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Daniel Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, Junyang Lin, Robert Brennan, Hao Peng, Heng Ji, and Graham Neubig. OpenHands: An open platform for AI software developers as generalist agents. In *International Conference on Learning Representations (ICLR)*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/a4b6ad6b48850c0c331d1259fc66a69c-Abstract-Conference.html.

Chunqiu Steven Xia, Yinlin Deng, Soren Dunn, and Lingming Zhang. Demystifying LLM-based software engineering agents. *Proceedings of the ACM on Software Engineering*, 2(FSE):801–824, 2025. doi: 10.1145/3715754.

Zhen Xiang, Linzhi Zheng, Yanjie Li, Junyuan Hong, Qinbin Li, Han Xie, Jiawei Zhang, Zidi Xiong, Chulin Xie, Carl Yang, Dawn Song, and Bo Li. GuardAgent: Safeguard LLM agents via knowledge-enabled reasoning. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, volume 267 of *Proceedings of Machine Learning Research*, pp. 68316–68342, 2025.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. OSWorld: Benchmarking multimodal agents for open-ended tasks in real computer environments. In *Advances in Neural Information Processing Systems*, volume 37, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/5d413e48f84dc61244b6be550f1cd8f5-Abstract-Datasets_and_Benchmarks_Track.html.

Frank (Fangzheng) Xu, Yufan Song, Boxuan Li, Yuxuan Tang, Kritanjali Jain, Mengxue Bao, Zora Wang, Xuhui Zhou, Zhitong Guo, Murong Cao, Mingyang Yang, Hao Yang Lu, Amaad Martin, Zhe Su, Leander Maben, Raj Mehta, Wayne Chi, Lawrence Jang, Yiqing Xie, Shuyan Zhou, and Graham Neubig. TheAgentCompany: Benchmarking LLM agents on consequential real world tasks. In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-0315. URL https://papers.nips.cc/paper_files/paper/2025/hash/0d744742f6fac4d1134c019b7cef3c8a-Abstract-Datasets_and_Benchmarks_Track.html.

Tianci Xue, Weijian Qi, Tianneng Shi, Chan Hee Song, Boyu Gou, Dawn Song, Huan Sun, and Yu Su. An illusion of progress? assessing the current state of web agents. In *Conference on Language Modeling (COLM)*, 2025.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems*, volume 37, pp. 50528–50652, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/5a7c947568c1b1328ccc5230172e1e7c-Abstract-Conference.html.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. WebShop: Towards scalable real-world web interaction with grounded language agents. In *Advances in Neural Information Processing Systems*, volume 35, 2022. URL https://proceedings.neurips.cc/pap

er_files/paper/2022/hash/82ad13ec01f9fe44c01cb91814fd7b8c-Abstract-Conference.html.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. In *International Conference on Learning Representations (ICLR)*, 2025.

Tongxin Yuan, Zhiwei He, Lingzhong Dong, Yiming Wang, Ruijie Zhao, Tian Xia, Lizhen Xu, Binglin Zhou, Fangqi Li, Zhuosheng Zhang, Rui Wang, and Gongshen Liu. R-judge: Benchmarking safety risk awareness for LLM agents. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 1467–1490, 2024. doi: 10.18653/v1/2024.findings-emnlp.79.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. WebArena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations (ICLR)*, 2024.

A EXECUTION PROTOCOLS

The normalized contract of Section 2.2 is the interface shared by all adapters; each adapter implements its own schema, matching rules, and recovery procedure, fixed within each reported protocol. Appendix A.1 states the rules shared by every controller, Appendix A.2 the protocol of each adapter, and Appendix A.3 the baselines and controls.

A.1 CONTROLLER RULES

Unknown and empty descriptions. An unavailable effect list and an explicit empty list are distinct outputs. Each adapter validates predictions against its own schema, and a nonempty free-text `unknown` field blocks an otherwise usable structured prediction only where the adapter marks that field as critical.

Verdict binding. Binding an EXECUTE verdict to the candidate, the contract, and the observation digest or repository version prevents its reuse for a changed candidate or a changed observed state. The controller withholds every candidate without an executable verdict.

A.2 ADAPTERS

AppWorld. The checked candidate is a bounded API plan. The adapter compares required record or change patterns with observed records, verified prior records, and predicted effects, and checks for extra or prohibited changes. The AppWorld-Splitwise adapter uses receipt identifiers for direct record reads and can accept a verified no-op when the requested record already exists. A revised candidate triggers a fresh receipt selection, since it may name records the first did not, followed by a new prediction and comparison; a VERIFY verdict repeats the reads and the prediction for the same candidate. The AppWorld-Splitwise and AppWorld-Amazon adapters share their budgets, at most 16 optional reads and 24 write actions per run, and a per-method cap on planning calls: one for No check and for List state first, three for Self-critique, four for Contract critic, and five for LEDGER; GuardAgent and AGrail check the executor's first plan within the call limits of their official implementations (Appendix A.3). The Gmail, Venmo, and Files adapters instead use their domain-specific observations and read interfaces; an unusable prediction can trigger a prediction retry. The controllers allow at most one candidate revision or one prediction retry along the corresponding recovery branch. A contract or prediction that stays malformed after its retry ends the run as a protocol failure, and every other stop is recorded as an abstention. Before execution the controllers re-read the guarded observation and compare its digest.

AgentDojo. Contract extraction has at most two attempts. Each declared state-changing tool call is checked against the fixed contract; reads execute directly. The check requires every predicted effect to match an allowed pattern. It records missing requirements without using them to reject legitimate partial progress. An unusable or malformed prediction receives one retry; if it remains unusable, the task ends as an abstention. An empty effect list for a state-changing call also receives one retry; if the second prediction is empty, it stands as a prediction of no change and can be permitted. This rule applies rarely: over the 267 judgment events of the three backbones, three state-changing calls produced an empty first prediction, two of which were permitted after the second prediction remained empty. Those two permits occur in two runs, one of which ends with a non-empty state difference. Matching tests each effect against the allowed patterns independently, without consuming a pattern's multiplicity. This adapter therefore decides which effects a call may have, and does not bound how many times an allowed effect may occur across calls: a second identical creation matches the same allowed pattern as the first. The count-sensitive check is carried by the AppWorld-Amazon and AppWorld-Splitwise adapters, whose contracts retain multiplicities and are reconciled over a complete plan rather than one call at a time. One correction is allowed after a blocked attempt; a second blocked attempt ends the task without further writes. The overall interaction uses the shared 15-iteration loop cap.

SWE-bench. The agent first edits and tests in a sandbox. LEDGER checks the resulting patch before publication to the shared repository, not each local shell operation. The contract and predictor express file-level changes; hidden repair and regression tests independently evaluate correctness.

One patch revision is allowed, with at most one prediction retry for each checked version. A permitted patch is applied against the checked repository version, and publication uses a compare-and-swap check. An empty patch is recorded as abstention, unlike a verified no-op in AppWorld-Splitwise.

A.3 Baselines and Controls

Guard-agent baselines. GuardAgent and AGrail are ported from their official implementations. We keep their prompts, their output protocols, GuardAgent's task decomposition, code generation, toolbox, error debugger, and limit of three auto-replies, and AGrail's analyzer, executor, memory retrieval, and memory update. Every model call goes to the run's backbone through the harness client, so its tokens are counted with the run, and AGrail's embeddings use one embedding model in place of the original OpenAI and MiniLM encoders. Both guards receive the same requirement, taken from AGrail's general principles: the agent's actions must match the user's request, and information must remain unaltered except by the actions the request authorizes. GuardAgent uses a fixed set of three seed examples from applications outside the evaluated tasks, since the official code grows its memory only from runs whose verdict matches a ground-truth label; AGrail's memory starts empty for each setting and backbone and accumulates across its runs. On AppWorld the guard checks the executor's first plan, on SWE-bench the patch to be published, and on AgentDojo every state-changing call. An unparseable guard output ends the run, as a LEDGER output that stays malformed after its retry does.

Contract critic. Contract critic is an additional control on AppWorld: a free-text reviewer checks the candidate against LEDGER's contract, without effect prediction or code reconciliation. It separates the contribution of an explicit contract from that of predicting effects and comparing them in code (Appendices B.1 and C).

B Results by Setting

This appendix supplements Section 3 one setting at a time and then reports computational cost. Every method runs the same tasks under the same backbones, so besides pooled rates we compare methods pairwise with the exact McNemar test over discordant pairs (McNemar, 1947), treating a task under one backbone as the unit.

B.1 AppWorld-Amazon

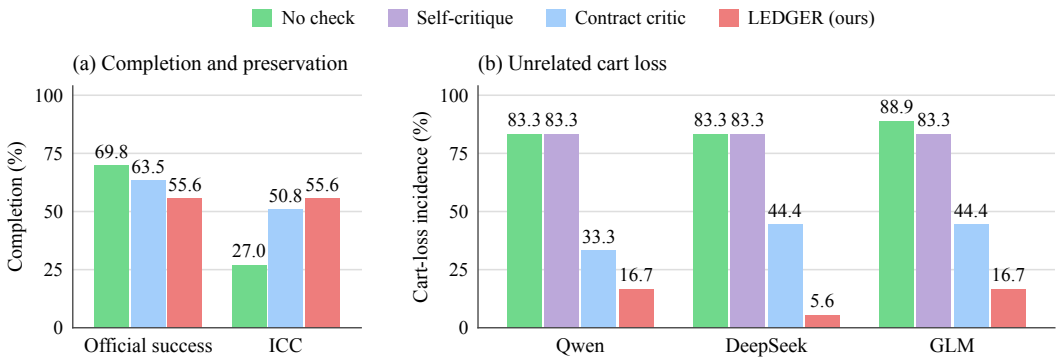


Figure 4: **Official success and state preservation on AppWorld-Amazon.** Results use all three backbones. (a) Each method is evaluated on 63 runs. Grouped bars compare official success with ICC under the stricter preservation and purchase constraints. (b) Cart-loss incidence over 18 exposed tasks per backbone. Bar heights and annotations are percentages.

Explicit preservation requirements. On AppWorld-Amazon, List state first and Self-critique leave cart-loss incidence at 46 and 45 of 54 runs, respectively, against 46 for No check. Contract critic, which gives the reviewer an effect contract, reduces this count to 22; LEDGER reduces it

further to seven (Figure 4(b)). Making the preserved set explicit is what reduces the loss; additional observation or free-text review leaves it unchanged. The remaining reduction, from 22 to seven, comes from predicting the plan's effects and reconciling them with the contract in code, together with the record-level feedback this comparison gives the revision.

Paired comparisons. LEDGER reaches 19 intent-consistent completions that No check does not and loses one that it does ($p < 10^{-4}$); against Contract critic, which uses LEDGER's own contract, the discordant counts are 11 and eight ($p = 0.65$), consistent with the contract carrying most of the gain.

Prediction fidelity. We audit prediction fidelity on a larger pool than the main evaluation: 168 LEDGER runs on the AppWorld-Amazon tasks, covering the three backbones and one additional executor, each from a clean start and from a start resumed after a partial first attempt. Of these, 128 reach execution under an EXECUTE verdict and produce a final prediction comparable against the recorded outcome; 17 end in a revision request, 13 stop before execution, and the rest end on an application or protocol error. On those 128 runs the predicted set of purchased items matches what was bought in every run, and the predicted end state of the cart matches in 127; the single mismatch predicts the purchase correctly and the residual cart wrongly. The audit thus measures prediction error on every candidate that executed, over the purchases and cart state it changes.

A larger order extension. A broader extension to 153 order tasks with Qwen confirms the reduction in record loss at scale. LEDGER reduces runs losing at least one pre-existing cart or wish-list record from 50 to 26, compared with 32 for Contract critic, and reduces the number of records destroyed from 330 to 172. Because the three methods run the same 153 tasks, we pair by task. Against No check the difference is one-sided: 24 tasks lose a record under No check and keep it under LEDGER, and no task loses a record under LEDGER and keeps it under No check ($p = 1.2 \times 10^{-7}$). Against Contract critic the discordant counts are nine and three in LEDGER's favor ($p = 0.15$). Official success is 21, 20, and 20 tasks for No check, Contract critic, and LEDGER, respectively, so LEDGER nearly halves record destruction without reducing official success.

B.2 APPWORLD-SPLITWISE

Task families and paired comparisons. AppWorld-Splitwise contains eight tasks from three families, run under three backbones for 24 runs per method. LEDGER reaches intent-consistent completion in 14 runs, against 10 for No check, 11 each for List state first and Self-critique, and 18 for Contract critic. Paired by task and backbone, LEDGER reaches it in seven continuations where No check does not and misses it in three where No check reaches it ($p = 0.34$); against List state first and against Self-critique the discordant counts are six and three ($p = 0.51$), and against Contract critic zero and four ($p = 0.13$), all four of them LEDGER protocol failures. The separation is concentrated where the mechanism predicts it. Six of the 24 runs come from the family in which the agent created the earlier records but is never their payer, so no listing view available to it returns them. LEDGER reaches intent-consistent completion in five of those six and Contract critic in all six, while No check, List state first, and Self-critique reach it in none: in every one of their runs the agent records the expense again in a group outside the task. Two tasks, one in each remaining family, cannot be completed by any method, because the prior progress already lies outside the required records.

Context isolation on AppWorld-Splitwise. Without context isolation, the variant records five expenses outside the task under Qwen on a task LEDGER completes, and it matches LEDGER on every DeepSeek and GLM run. Its ICC on this setting is higher than LEDGER's (66.7% against 58.3%) because three runs on which LEDGER ends in protocol failures of its additional stages, two unparseable predictions and one contract that fails validation, complete under the variant.

B.3 AGENTDOJO

AgentDojo under each backbone. Table 5 reports every method on AgentDojo under each backbone, including the prompt-injection defenses that Table 2 omits. Tool filtering keeps violations near zero by completing almost nothing, with official success between 4.1% and 12.4%, while the ICC of

Table 5: **AgentDojo under each backbone.** Official success (Succ.), ICC, and violation rate (Viol.) in percent of 97 runs per method and backbone. The pre-execution checks are the baselines of Table 2; the prompt-injection defenses ship with AgentDojo or target it, and run here without attacks. Best per column in **bold**.

	Qwen			DeepSeek			GLM		
Method	Succ.↑	ICC↑	Viol.↓	Succ.↑	ICC↑	Viol.↓	Succ.↑	ICC↑	Viol.↓
No check	92.8	86.6	8.2	**95.9**	**93.8**	2.1	91.8	87.6	4.1
Pre-execution checks									
Self-critique	88.7	85.6	3.1	89.7	85.6	4.1	89.7	84.5	5.2
GuardAgent	**93.8**	**89.7**	5.2	90.7	88.7	3.1	**93.8**	**89.7**	5.2
AGrail	75.3	73.2	3.1	80.4	77.3	4.1	83.5	80.4	3.1
Prompt-injection defenses									
Tool filtering	4.1	4.1	**0.0**	5.2	5.2	**0.0**	12.4	11.3	**1.0**
Prompt sandwiching	91.8	86.6	7.2	**95.9**	90.7	6.2	90.7	86.6	8.2
Progent	85.6	80.4	7.2	87.6	83.5	4.1	89.7	86.6	4.1
LEDGER (ours)	87.6	83.5	4.1	80.4	77.3	3.1	81.4	77.3	4.1

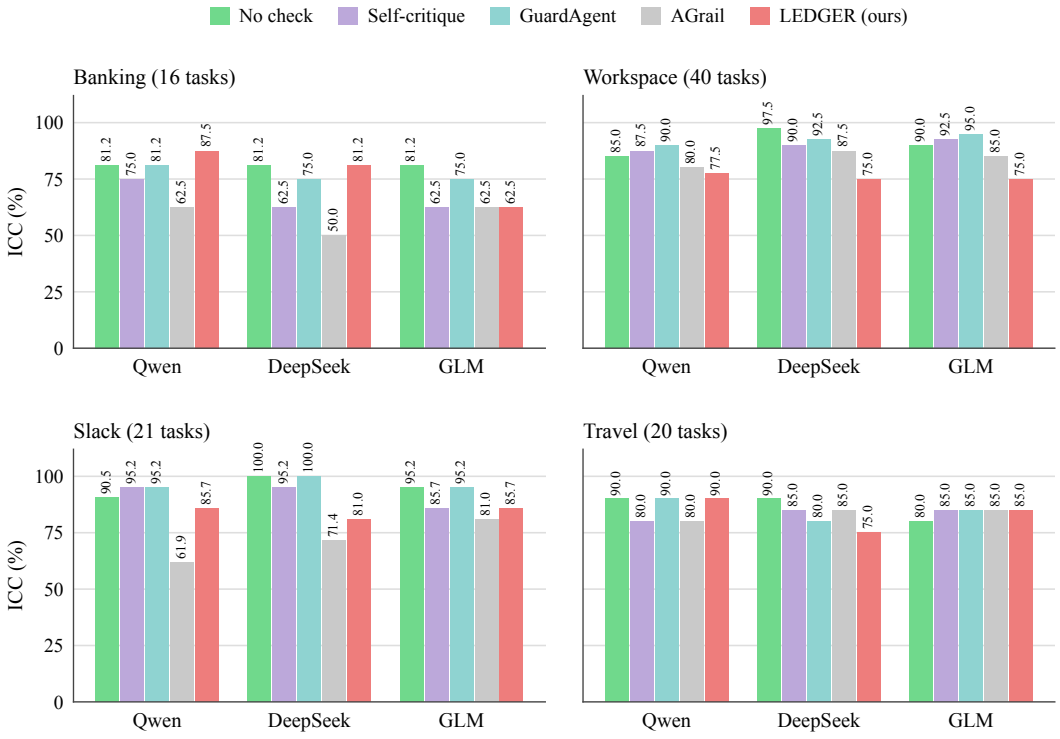


Figure 5: **ICC by AgentDojo domain and backbone.** Each panel reports one domain, with all five methods evaluated on the same tasks under each backbone. ICC requires official success without changes outside the reference execution's effects. Bar labels are percentages.

Prompt sandwiching stays within 3.1 points of No check under every backbone, and that of Progent within 10.3 points. No check reaches 86.6%, 93.8%, and 87.6% ICC under Qwen, DeepSeek, and GLM, and LEDGER reaches 83.5%, 77.3%, and 77.3%. The difference is three tasks under Qwen and 16 and 10 under DeepSeek and GLM, and it follows LEDGER's abstentions, which number seven, 13, and 12. Of the 32 abstentions, 31 end without official success. Twenty-seven end on a second blocked attempt whose predicted effect matches no allowed pattern of the contract, three on a malformed effect object, and two on an unusable prediction. They concentrate on a few tasks: five tasks, four in Workspace and one in Slack, abstain under all three backbones and two more under

two backbones, together accounting for 19 of the 32. The cost therefore concentrates in contract coverage of a few requested writes, such as creating the requested packing-list file, scheduling the requested event, or adding the new colleague to a channel: the allowed patterns of the contract omit the effect that completes the task, so the check withholds it under every backbone.

AgentDojo domains. Figure 5 locates the AgentDojo cost by domain for the methods of Table 2 and No check. Pooled over backbones, No check and LEDGER reach 39 and 37 of 48 Banking runs, 109 and 91 of 120 Workspace runs, 60 and 53 of 63 Slack runs, and 52 and 50 of 60 Travel runs. Paired by task and backbone, No check reaches 36 intent-consistent completions that LEDGER does not, and LEDGER reaches seven that it does not ($p < 10^{-5}$). Workspace contributes 21 of the 36, 16 of them abstentions, and Slack contributes eight, seven of them abstentions; Workspace also holds 21 of LEDGER's 32 abstentions. Banking is the only domain in which LEDGER changes records outside the reference effects more often than No check, in eight runs against six, all eight with official success. Four of the eight occur under GLM, whose Banking ICC falls to 62.5%; under Qwen, LEDGER has the highest Banking ICC of the five methods, 87.5%. Neither method changes records outside the reference effects in Travel, so the Travel differences come from completion alone.

B.4 SWE-BENCH

Under each backbone. LEDGER has the highest SWE-bench ICC under GLM (90.7%) and is second to Self-critique under Qwen (90.7% against 93.0%) and DeepSeek (60.5% against 67.4%) (Figure 3(c)). Of LEDGER's 22 withheld patches, 17 come from DeepSeek.

Paired comparisons. LEDGER gains 21 intent-consistent completions that No check does not reach and loses 10 that it does ($p = 0.07$). Against Self-critique the discordant counts are 15 and 14 ($p = 1.0$); LEDGER reaches its ICC with no out-of-scope patch, whereas six of Self-critique's passing patches change files outside the task.

Out-of-scope changes. All 11 runs with out-of-scope changes, six under Self-critique, four under No check, and one under AGrail, edit documentation (`docs/api.rst`, `docs/versions.rst`) or a type stub (`more_itertools/recipes.pyi`), and all of them pass the hidden tests. The preservation constraint takes the task's source files, which are shown to every method as the files most likely to be involved, together with its test files; LEDGER's contract extractor receives the same source files.

B.5 COMPUTATIONAL COST

Across the three backbones, LEDGER abstains in 32 of 291 AgentDojo runs and 22 of 129 SWE-bench runs. Relative to No check, mean token use increases from 16.0k to 32.8k on AgentDojo and from 71.3k to 80.0k on SWE-bench. The run records attribute these tokens to the calling role. On AgentDojo the agent's own consumption is unchanged, 15.9k against 16.0k under No check, and the entire increase is the two model calls of the check: 10.1k for contract extraction over 1.01 calls per run and 6.8k for effect prediction over 0.95 calls per run. On SWE-bench the check costs 4.8k per run, 3.6k for the contract and 1.2k for the predictor, and the remaining difference is the agent's own longer session. Self-critique uses 103.1k tokens per run there against LEDGER's 80.0k, because its revision loop re-reads and rewrites the patch in free text, so on this setting the effect-level check reaches higher intent-consistent completion at about three quarters of the cost. Evidence reads are issued by the adapter and consume an adapter read budget outside these token totals.

C FURTHER APPWORLD FAMILIES

We also run LEDGER on three AppWorld families outside the four settings of Section 3: conditional bulk actions in Gmail, outward writes in Venmo, and local records in the file system, with 30, 45, and 18 runs per method and backbone. No check and Contract critic (Appendix A.3) receive the same initial observations and read interfaces within each family. All three methods run under a protocol fixed before the runs, which we call the frozen protocol.

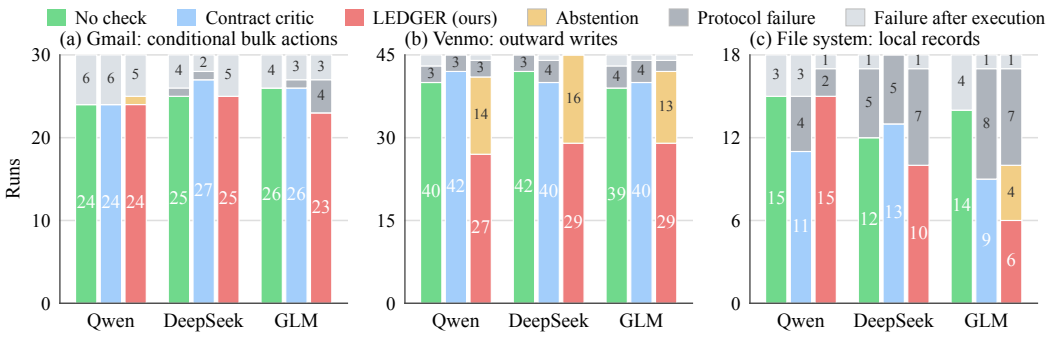


Figure 6: **AppWorld outcomes under the frozen protocol.** Each bar assigns every run to one of four outcomes and reaches the family's run count: 30 Gmail, 45 Venmo, and 18 Files runs per method and backbone. The colored base is official success, and segments too thin for a label are left unlabeled. These results precede the vocabulary-normalization supplement.

Frozen results. Figure 6 assigns every run to official success, abstention, protocol failure, or failure after execution. Pooled over backbones, LEDGER completes 72 of 90 Gmail runs, 85 of 135 Venmo runs, and 31 of 54 Files runs, against 75, 121, and 41 for No check and 77, 122, and 33 for Contract critic.

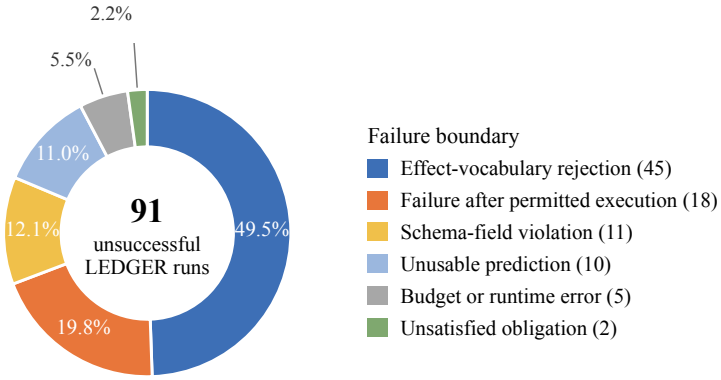


Figure 7: **Where LEDGER runs end without official success.** The 91 runs of Figure 6 that do not pass the official evaluator, categorized by the condition that ends them. Counts pool 279 LEDGER runs over three backbones on Gmail, Venmo, and Files.

Agreement on the effect vocabulary. Representation mismatch accounts for most of the runs that end without official success. Figure 7 categorizes the 91 such runs by the condition that ends them: 45 abstain because the predicted effects use a change kind or payload the declared vocabulary does not cover, 11 end as protocol failures because an effect object carries fields outside the schema, and 10 end on a prediction that cannot be parsed. Two runs stop because the checked plan leaves a required change unmade, five end on a read budget or a runtime error, and 18 fail the official evaluator after the check permits execution. In frozen Venmo runs, 41 of LEDGER's 43 abstentions arise from vocabulary or schema rejection. In Files, nine of 54 LEDGER runs fail because effect objects contain fields outside the declared schema.

Vocabulary normalization. A follow-up experiment canonicalizes model outputs to the declared vocabulary and reruns both contract-based methods on the same tasks. LEDGER's Venmo success increases from 85 to 117 of 135 runs, while abstentions decrease from 43 to five; Files success increases from 31 to 44 of 54. Contract critic reaches 120 of 135 and 44 of 54, respectively. These reruns are reported separately from the frozen results shown in Figure 6. The recovery identifies

representation mismatch as the main bottleneck on these families, and a vocabulary shared by the extractor and the predictor removes it.