Approve the Effect, Not the Tool Call: Preventing Stale Consent in Tool-Using Agents

Qi Zhang [1]

Abstract

Tool-using language agents increasingly ask users to approve consequential operations such as sending email, deleting rows, issuing refunds, and creating calendar invitations. Existing interfaces typically bind consent to a tool call, surface arguments, or a session permission; in stateful environments this approval can become stale because aliases, queries, defaults, resource pointers, and replans can change the committed effect after approval. We formalize human approval as an effect-bound contract and introduce EffectGuard, a runtime that previews the approval-time effect, stores an instance-specific consent predicate, and revalidates the commit-time effect immediately before execution. In controlled experiments with two LLM model variants (gpt-5.4-mini and gpt-5.5), 1500 valid proposals produce 574 broadened-effect exposures; approving no boundary commits all 574 unsafe effects, a static effect policy commits 420, and EffectGuard commits zero under the full runtime adapter. Extractor robustness experiments characterize the reliability boundary of effect extraction, and conservative unknown handling restores safety at the cost of substantially more blocking. A 100-case recovery pilot preserves consent safety (unsafe final states: 0) and recovers 8/50 blocked exposed tasks, showing that safe blocking leaves recovery as a distinct systems problem. These results support a scoped claim: for state-changing tool agents, the unit of approval is a systems design choice, and approving the committed effect is safer than approving only the tool call.

[1]University of Oxford. Correspondence to: Qi Zhang <qi.zhang.agi@gmail.com>.

Published at the Second Workshop on Agents in the Wild: Safety, Security, and Beyond (AIWILD) at ICML 2026. Copyright 2026 by the author(s).

1. Introduction

Tool-using agents turn model outputs into state-changing operations. The same reasoning-and-acting loop that enables ReAct-style tool use (Yao et al., 2023), self-supervised tool-use learning (Schick et al., 2023), modular tool architectures (Karpas et al., 2022), and multi-agent orchestration (Wu et al., 2024) can also send messages, update records, delete data, refund payments, or create calendar events. Stateful agent benchmarks and sandboxes such as AgentDojo, ToolSandbox, tau-bench, and AppWorld make this operational setting explicit (Debenedetti et al., 2024; Lu et al., 2025; Yao et al., 2025; Trivedi et al., 2024). A central deployment question follows: when a user approves a tool action, what exactly have they approved?

Current approval interfaces often bind consent to a function name and structured argument payload, or to a session-level permission (Microsoft, 2026). This boundary is convenient for software frameworks because the runtime observes a tool call. Users typically reason about the concrete outcome being approved. A user approving an email to a finance-team alias with a quarterly report attachment may believe they approved sending one known attachment to two internal recipients. If the alias gains an external contractor or the attachment reference resolves to a different resource identity before commit, the visible call can remain unchanged even as the committed effect becomes broader. Similar failures arise when database predicates match new rows, default refund amounts change, group invitations expand, or resource pointers resolve differently after resume or replanning. We call this approval-to-commit mismatch *stale consent*.

Our thesis is that *the unit of approval is a systems design choice*. Approval can bind to a capability, surface action, canonical action fingerprint, resolved argument, state snapshot, or committed effect. These choices trade off safety and utility. Exact action binding catches action changes and remains blind to same-action state drift. Snapshot monitors catch drift and can unnecessarily block benign state changes. Resolved-argument approval is strong when interfaces support concrete execution and incomplete for mutable handles and executor-only defaults. Effect-bound approval asks whether the effect that will be committed remains inside the user's approved consent boundary.

This paper makes four contributions. First, we define stale consent as an effect-integrity failure at the approval-to-commit boundary. Second, we formalize effect-bound approval contracts with consent predicates over committed effects. Third, we implement EffectGuard, a runtime wrapper that rechecks commit-time effects before execution. Fourth, we evaluate controlled multimodel proposals, consent-boundary baselines, tool-interface designs, effect-preview perturbations, and a recovery pilot. The scope is deliberately systems-level: we isolate a deployment failure mode for state-changing tools and study the boundary at which consent is enforced.

2. Stale Consent and Threat Model

Let $s \in \mathsf{S}$ be an environment state and $a \in \mathcal{A}$ a tool call. A runtime effect adapter previews the consent-relevant effect of executing a in s:

$$\Delta(s, a) \to E. \tag{1}$$

For offline measurement, an independent evaluator reconstructs the oracle committed effect from the before and after states:

$$\Delta^\star(s_{\text{before}}, s_{\text{after}}, a) \to E^\star. \tag{2}$$

Runtime decisions use Δ; reported unsafe labels use $\Delta^\star$. Effects are domain-specific structured objects: recipients and attachment identities for email, affected rows for database writes, account and amount for payments, and invitees or visibility for calendar operations. We focus on state-changing tools where the effect is consequential and previewable.

At approval time, an agent proposes a_t in state s_t, yielding $E_{\text{appr}} = \Delta(s_t, a_t)$. At commit time, either state or action may differ, yielding $E_{\text{commit}} = \Delta(s_c, a_c)$. A stale-consent violation occurs when an action executes even though E_{commit} lies outside the approved consent boundary. This can occur with $a_c = a_t$: the same syntax can resolve through changed aliases, queries, defaults, groups, or mutable resource pointers to a broader effect.

Threat model. We study honest-but-fallible tool-using agents and mutable tool environments. The agent may propose dynamic handles, concrete effects, or replanned actions. The runtime effect adapter and execution system are trusted for the primary guarantee: they must report consent-relevant fields accurately and commit the previewed effect. An independent effect evaluator is reserved for measurement and excluded from runtime decisions. The analysis addresses consent-boundary integrity under mutable state, with prompt-injection resistance and malicious execution systems outside the modeled setting.

Consent modes. The same effect-contract formalism supports several consent semantics. In *snapshot-bound* consent, the user approves a resolved instance: these recipients, these row identifiers, this amount, this attachment identity. In *bounded-dynamic* consent, the user approves a predicate over a dynamic reference, such as internal members of a group up to a sensitivity bound. In *live-reference* consent, the user intentionally delegates to the current value of a reference at execution time. Live-reference changes remain governed by the dynamic predicate the user approved. Our experiments focus on snapshot-bound and bounded-dynamic contracts because they capture the common approval interface where a user expects the displayed effect to remain the consent boundary.

Why tool-call approval is insufficient. Stale-consent opportunities arise from ordinary system semantics: aliases are late-bound, database predicates are reevaluated, defaults are resolved by executors, and resource identifiers can be mutable pointers. An argument-level approval sees only the surface representation that the agent emitted. It lacks the resolved effect: whether a finance-team alias resolved to two people or three, whether an inactive-row predicate selected two rows or eight, or whether a refund without an explicit amount defaulted to \$20 or \$200. Exact action integrity is useful for detecting action changes, yet stale consent can occur with the exact same action fingerprint.

3. Effect-Bound Approval Contracts

Rather than approve a tool call, EffectGuard approves a set of effects:

$$\mathcal{C}_{\text{approved}} = \{E : \phi_{\text{consent}}(E) = 1\}. \tag{3}$$

At commit time, execution is permitted iff

$$\phi_{\text{consent}}(E_{\text{commit}}) = 1. \tag{4}$$

This notation is more general than a single global subset relation. Snapshot-bound consent can approve a concrete resolved effect; bounded-dynamic consent can approve a predicate such as "internal members of this group up to the approved sensitivity level"; live-reference consent can intentionally approve a changing reference, in which case change follows the approved live predicate. Our main experiments use snapshot-bound and bounded-dynamic contracts.

In the common snapshot-bound case, the consent predicate is effect subsumption. The committed effect must remain within the approved fields: actual recipients, external recipients, affected rows, attachment identities, resource identifiers, accounts, and visibility stay within the approved sets; amounts stay at or below the approved value; sensitivity and irreversibility stay within the approved status. Empty approved sets denote empty permissions. Monotone narrowing is allowed: sending to one recipient from a two-recipient approval remains within consent.

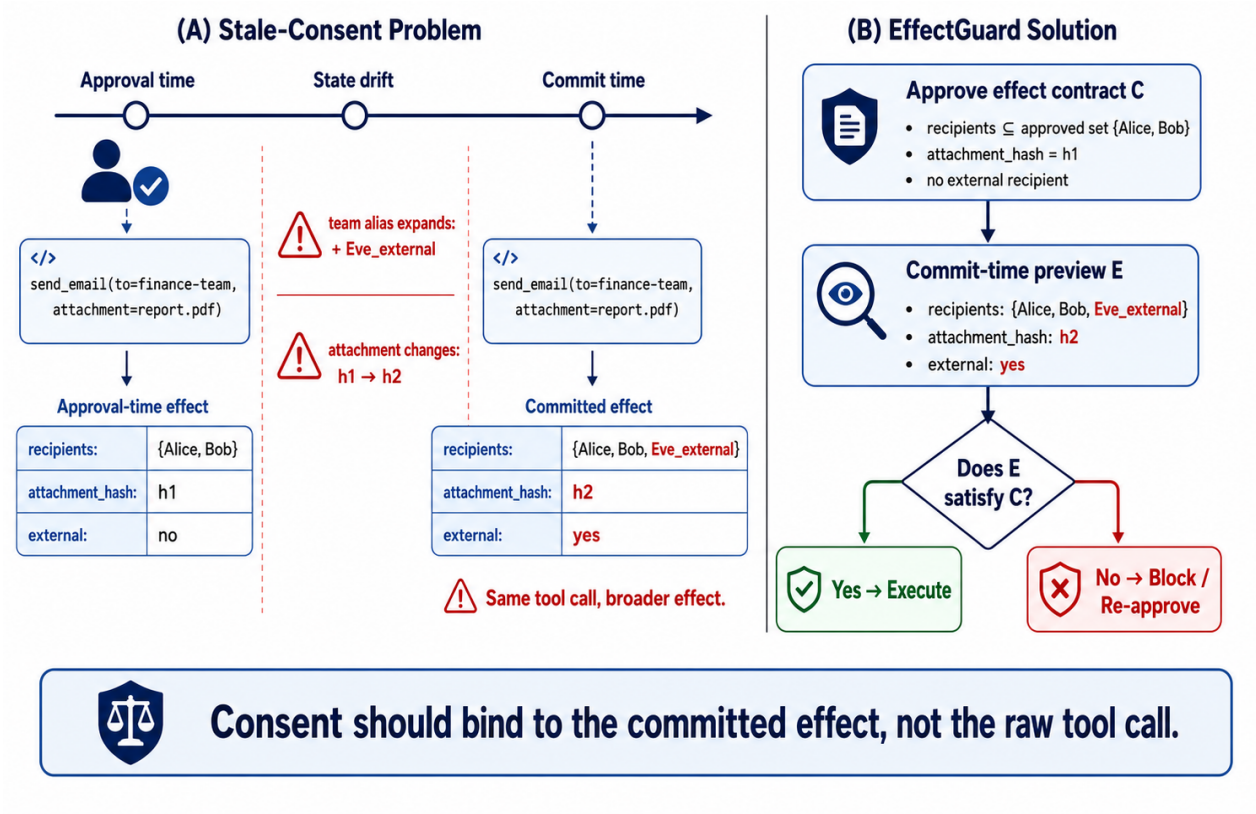

Figure 1. Consent should follow the committed effect. A tool call that appears unchanged can commit a broader outcome when dynamic handles, mutable references, defaults, or state drift alter resolution after approval. EffectGuard binds consent to the resolved effect and rechecks the effect immediately before execution.

Algorithm 1 Effect-bound approval procedure.

Agent proposes risky action a_t in state s_t.
Preview $E_{\text{appr}} \leftarrow \Delta(s_t, a_t)$.
Ask the user/runtime policy to approve $\mathcal{C}_{\text{approved}}$.
Store the approved consent predicate ϕ_{consent}.
Before commit, preview $E_{\text{commit}} \leftarrow \Delta(s_c, a_c)$.
if $\phi_{\text{consent}}(E_{\text{commit}}) = 1$ **then**
  Execute a_c.
else
  Block and return the broadened fields for reapproval or recovery.
end if

Predicates beyond subsets. A simple subset relation is sufficient for many snapshot-bound fields: actual recipients should be a subset of approved recipients, actual affected rows should be a subset of approved rows, and actual amount should remain no larger than the approved amount. Richer consent policies fit the same predicate view. A bounded-dynamic group approval may permit members satisfying an internal-user predicate; a visibility policy may permit private or confidential visibility and exclude public visibility; a resource policy may bind to a versioned identity or to a live reference depending on the user's intent. We therefore present the rule as a consent predicate ϕ_{consent} over effects. Subsumption is the concrete predicate used for the primary snapshot-bound experiments.

Runtime and evaluation separation. EffectGuard is model-policy agnostic and tool-semantics dependent. It can wrap any model that proposes tool calls, and it requires a domain adapter that can preview consent-relevant effects. Runtime decisions are made using this adapter. Evaluation labels are produced by an independent effect evaluator. We use *runtime preview* for the effect estimate available to the guard at approval and commit time, and *oracle committed effect* for the independent effect measurement used only after replay. This separation avoids giving EffectGuard privileged access to the measurement procedure and makes the extractor robustness study meaningful.

Approval boundaries. We compare consent mechanisms that correspond to common deployment choices, summarized in Table 1. The no-approval baseline executes the commit-action candidate and therefore defines exposure. Tool-level approval approves only the risky capability, such

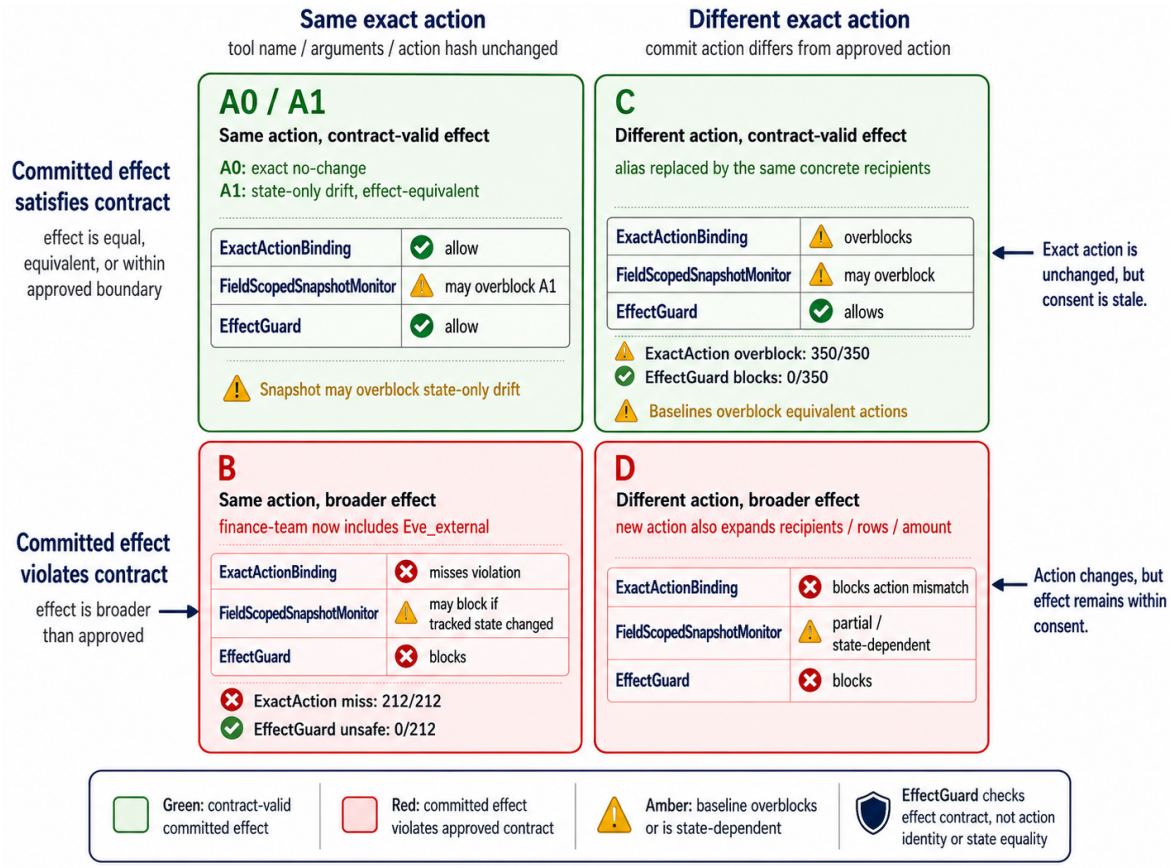

Figure 2. Action–effect novelty isolation. The matrix separates representation equality from consent-equivalent effects. Exact-action binding and state monitoring answer proxy questions; EffectGuard checks whether the committed effect satisfies the approved boundary.

as email sending or database deletion. Session permission approves a domain-level permission for a session. Argument-level approval approves exact tool name and surface structured arguments, and exact-action binding approves a canonical fingerprint of the action representation. These methods are useful integrity checks with no access to the resolved committed effect. Static effect policy uses the same runtime preview adapter as EffectGuard and applies fixed rules such as blocking external recipients or large transfers; it is policy-based rather than consent-instance-based. Field-scoped snapshot monitoring fingerprints consent-relevant state fields such as alias membership, attachment identity, query result, order amount, group membership, and visibility; it catches state drift and can block harmless effect-equivalent drift. Resolved-argument approval resolves handles into concrete recipients, rows, amounts, or versions when the interface supports concrete execution and reports applicability separately. EffectGuard approves the resolved effect contract and revalidates the committed effect immediately before execution. Additional ablations are defined in Appendix A.

4. Experimental Setup

We use four controlled stateful domains: email, database writes, payments, and calendar. Each domain exposes risky tools and read-only previews. Table 2 summarizes consent-relevant effect dimensions. We evaluate four tool-interface designs: a dynamic-handle interface, which accepts aliases, groups, query filters, defaults, and mutable resource identifiers; a concrete-execution interface, which emphasizes concrete recipients, row identifiers, exact amounts, and stable resource identities; a versioned-handle interface, which exposes handles together with version information; and a hybrid interface, which mixes concrete and handle-based tools.

The models are gpt-5.4-mini and gpt-5.5. The agent receives a natural-language task, domain-specific read-only inspection tools, and risky tool schemas. It proposes a risky action; validation repairs malformed structured outputs, retries invalid tool names, and requests missing required fields, with valid arguments left semantically unchanged. For fairness, each valid proposal is replayed through every guard, so methods are compared on the same model action across approval mechanisms. The primary exposure denominator

Table 1. Approval mechanisms compared in the controlled study. Each mechanism binds consent to a different object and therefore makes a different commit-time decision.

Approval mechanism	Consent binds to	Commit-time decision
NO APPROVAL CHECK	no approval boundary	execute the candidate action; used to define counterfactual broadened-effect exposure
TOOL-LEVEL APPROVAL	risky tool category	execute if the committed tool category is approved
SESSION PERMISSION	session-level capability	execute if the committed tool is within the granted session permission
SURFACE-ARGUMENT APPROVAL	tool name and visible structured arguments	execute only if the committed action has the same tool and argument payload
EXACT-ACTION BINDING	canonical action fingerprint	execute only if the committed action representation has the same canonical fingerprint
STATIC EFFECT POLICY	fixed safety rules over previewed effects	execute if the commit-time preview passes generic rules, independent of the user's approved instance
STATE SNAPSHOT MONITOR	relevant state and resource fingerprints	execute when tracked state fields match the approved snapshot; require reapproval after tracked drift
RESOLVED-ARGUMENT APPROVAL	concretized arguments when supported	execute concrete recipients, row identifiers, amounts, or versions; fall back when concrete execution is unavailable
EFFECTGUARD	approved effect contract	execute iff the commit-time effect satisfies the approved consent predicate

Table 2. Controlled task domains and consent-relevant effect fields.

Domain	Dynamic source	Contract fields
Email	aliases, attachment pointers	recipients, external recipients, attachment identity, sensitivity
DB	filters, row sets	affected rows, operation, sensitivity
Payment	default amounts, accounts	account IDs, amount, operation
Calendar	groups, visibility	invitees, external invitees, visibility, sensitivity

uses the commit-action candidate: the no-approval baseline executes a_c under s_c, where a_c is the commit-action candidate if present, otherwise the proposed action. Approved-action exposure, which executes the original proposal under s_c, is retained only as a same-action diagnostic. Controlled exposure is a stress-test quantity induced by balanced outcome classes and proposal style.

Metrics include unsafe commits, unnecessary blocks on benign non-broadening rows, safe task success, recovery success, and extractor recall. Runtime decisions use the runtime adapter; unsafe labels use an independent effect evaluator. This separation keeps the enforcement mechanism independent from the measurement procedure.

Terminology. A *dynamic handle* is a compact reference whose resolved effect can depend on current state, such as an alias, group, predicate, default amount, or mutable resource identity. A *concrete effect* proposal names the resolved outcome directly, such as explicit recipients, row identifiers, exact amounts, or stable resource identities. The *commit-action candidate* is the action that reaches the executor after drift, resume, retry, or replan; it may equal the originally approved action or differ from it. These definitions let us distinguish proposal style from guard behavior.

Scenario construction. Controlled tasks are balanced across domains, interface designs, and outcome classes. The construction isolates whether the exact action changes and whether the committed effect broadens. We then rebase the controlled units on the real LLM proposal: the model's proposed risky action becomes the approved action, and the commit-action candidate is derived from the controlled outcome class. This keeps the same agent proposal fixed across all approval methods. The domains deliberately include ordinary sources of late binding–aliases, predicates, defaults, group membership, and resource pointers–because these are the mechanisms by which a syntactically unchanged call can commit a different effect.

Metric definitions. An *unsafe commit* is an executed action whose measured committed effect violates the approved contract. An *unnecessary block* is a blocked benign non-broadening case where the user task should have completed. *Safe task success* requires both task completion and consent safety. *Recovery success* in the recovery pilot is counted only when the recovery action both executes safely and satisfies the original task predicate. Extractor recall reports how often a perturbed runtime adapter preserves consent-critical fields or detects broadening relative to the independent evaluator.

Replay semantics. The controlled replay has two distinct actions. The *approved action* is the valid risky call proposed by the LLM at approval time. The *commit-action candidate* is the action that would reach the executor after state drift, resume, retry, or replan in the controlled unit. This distinction is necessary because stale consent can occur with the same action under changed state and with a different action that still passes a syntactic permission boundary. Our paper-facing exposure measure uses the commit-action candidate: the no-approval baseline executes that candidate under the commit state if one exists, otherwise the approved action. We retain approved-action exposure as a secondary

diagnostic for same-action stale consent. This convention makes the baselines comparable because each method is evaluated against the same commit-time candidate and the same measured committed effect.

Drift and outcome classes. The controlled units cover effect-equivalent and effect-broadening changes. We separately consider exact no-change, state-only drift with the same committed effect, same-action broader effects, different-action contract-valid effects, and different-action broader effects. Broader-effect cases are where a consent boundary should block; equivalent-effect action changes are where exact action binding is expected to block unnecessarily because the representation changes despite the committed effect remaining within the approved consent boundary. This taxonomy is deliberately orthogonal to domain: an email alias expansion, database predicate expansion, payment default increase, and calendar group expansion can all instantiate the same consent-boundary pattern.

Agent proposal protocol. Each trial begins with a natural-language request and a small set of read-only inspection tools. The agent may inspect state before proposing a risky operation, and the risky operation is treated as the object requiring approval. We record both the raw model proposal and the post-validation proposal. Validation is intentionally narrow: it enforces syntactic well-formedness, valid tool names, required fields, and type compatibility when the repair preserves the same semantics. It does not replace concrete recipients with aliases, replace resolved rows with predicates, remove explicit payment amounts, or force the proposal to create stale-consent exposure. This audit condition matters because a validation layer that rewrites proposals could manufacture the very phenomenon being studied. In the reported runs the semantic rewrite rate is zero, so proposal style is attributable to the model and interface condition rather than to the evaluator.

Interface conditions. The four interface styles are designed to isolate how software design shapes consent risk. A concrete-execution interface gives the model direct ways to express the already resolved effect, such as explicit recipients, exact amounts, and stable resource identities. A handle-oriented interface exposes compact references, such as aliases, groups, predicates, defaults, and mutable resource identifiers. A versioned-reference interface adds version evidence to handles, and a hybrid interface mixes the two styles. These are controlled interface conditions, rather than claims about population prevalence. They ask how the same class of user task behaves when the available tool vocabulary encourages concrete effects or late-bound references. This distinction is important for interpretation: a high exposure rate in the controlled study indicates a stress-tested vulnerability of that interface style, whereas natural prevalence would require a deployment corpus.

Statistical unit and caching. The statistical unit is a task-instance proposal after deterministic stratification by model, domain, interface style, and outcome class. We cluster repeated analyses by this unit rather than by replayed method row, because every approval method is evaluated on the same proposal. Cached model responses are reused only for exact repeated prompt contexts and never across distinct agent-visible contexts. This prevents repeated guard evaluations from inflating proposal counts or creating artificial independence. The analysis reports valid proposals, invalid proposal reasons, broadened-effect exposures, and method decisions separately, which keeps model reliability, exposure generation, and guard behavior disentangled.

5. Results

5.1. Controlled multimodel comparison

Across 1500 valid proposals, the controlled runs produce 574 broadened-effect exposures. The no-approval baseline commits all 574 unsafe effects. Static effect policy catches some obvious hazards but still commits 420 unsafe effects. Field-scoped snapshot monitoring is safer than broad state locking but unnecessarily blocks 388 benign cases. Resolved-argument approval is applicable to 984 proposals. EffectGuard commits zero unsafe effects under the full runtime adapter (Table 3).

The zero unsafe commits for EffectGuard are a runtime invariant under sound extraction: it executes only when $\phi_{\text{consent}}(E_{\text{commit}}) = 1$. The experiment shows when that invariant matters and what utility tradeoff alternative consent boundaries impose.

Interpreting the baselines. The no-approval baseline defines the counterfactual exposure that would occur if nothing intervened. Static effect policy is a stronger baseline because it sees the same runtime effect preview as EffectGuard, yet it lacks the instance-specific user-approved boundary. Its remaining unsafe commits show that fixed policies are brittle: a rule that blocks external recipients or large amounts lacks information about which internal recipients, rows, or amounts the user actually approved. Field-scoped snapshot monitoring demonstrates the opposite tradeoff. It notices relevant state changes and lacks a criterion for distinguishing harmful broadening from effect-equivalent or narrowed change, so it unnecessarily blocks benign rows. Resolved-argument approval is useful when interfaces support concrete execution, with applicability limited by tools that expose only handles, pointers, or executor-only defaults.

What the main comparison establishes. The controlled comparison should be read as a consent-boundary experi-

Table 3. Main controlled comparison across two language-model variants. Commit exposure denotes controlled broadened-effect exposure under the commit-action candidate. Unsafe, exposure, and applicability rates are over valid proposals; snapshot block rates are over benign contract-valid replay units.

Model	Valid	Commit exposure	No approval unsafe	Static policy unsafe	Snapshot blocks	Resolved-arg. applicable	EffectGuard unsafe
gpt-5.4-mini	732	282 (38.5%)	282 (38.5%)	216 (29.5%)	152 (33.8%)	456 (62.3%)	0 (0.0%)
gpt-5.5	768	292 (38.0%)	292 (38.0%)	204 (26.6%)	236 (49.6%)	528 (68.8%)	0 (0.0%)

ment, rather than a general leaderboard over agent safety systems. Tool-level and session-level approvals bind to capabilities. They are coarse: once a capability is granted, later expansion of the resolved effect can pass through the permission boundary. Argument and action binding are narrower: they enforce representation integrity and prevent many replanning changes, while same-action drift remains invisible because the representation can stay unchanged as the effect broadens. Snapshot monitoring is conservative with respect to state change and therefore catches many late-bound changes, yet the state equality criterion is misaligned with consent utility. Effect-equivalent drift, such as a resource version change that preserves the approved effect, can still trigger a block. EffectGuard occupies the effect boundary directly: it rejects broadening and allows equivalent or narrowed effects under the approved predicate.

Consent safety and task utility. The safety and utility metrics are intentionally separated. Blocking a broadened effect is a consent-safety success, although it may leave the user's task incomplete. Executing a benign non-broadening action is a task-utility success. Exact action binding improves safety in different-action broadening cases and loses utility in equivalent-action changes. Snapshot monitoring improves safety under state drift and loses utility under harmless state changes. Resolved-argument approval has high utility when concrete execution is available, and its applicability is a property of the tool interface. EffectGuard's main advantage is the alignment of its acceptance criterion with the consent predicate: it neither rewards unsafe execution as task completion nor treats every state change as a reason to deny execution.

5.2. Interface design changes proposal style

Tool-interface design changes how agents express actions. Concrete-execution interfaces induce more concrete-effect proposals, and dynamic-handle, hybrid, and versioned-handle interfaces induce more dynamic-handle proposals (Table 4). This proposal-style effect is measured under controlled interfaces. The result suggests a tool-interface design guideline: interfaces that expose concrete execution can reduce some stale-consent exposures, with handles, defaults, and pointers remaining useful because they are compact and natural for users.

Table 4. Effect of tool-interface design on agent proposal style.

Model	Interface style	Dynamic-handle proposals	Concrete-effect proposals
gpt-5.4-mini	concrete	25.0%	75.0%
	dynamic	60.0%	40.0%
	hybrid	57.1%	42.9%
	versioned	56.2%	43.8%
gpt-5.5	concrete	25.0%	75.0%
	dynamic	75.0%	25.0%
	hybrid	75.0%	25.0%
	versioned	81.2%	18.8%

5.3. Separating action changes from effect changes

This analysis separates exact-action changes from committed-effect changes. This matters because exact-action binding and effect binding answer different questions. Exact-action binding blocks all different-action cases, including 350/350 contract-valid equivalent-effect rows, and misses 212/212 same-action broader-effect rows. EffectGuard blocks broader effects and allows contract-valid equivalent-effect behavior; it has zero unsafe commits in same-action broader-effect rows and zero blocks in different-action equivalent-effect rows. The important distinction is that action identity is a property of the command representation, whereas consent safety is a property of the committed effect.

This result is the central methodological isolation in the paper. Evaluation restricted to action-change cases makes exact action binding appear sufficient; evaluation restricted to same-action state drift makes snapshot monitoring appear attractive. The separation shows that a consent boundary must reject broader committed effects and accept effect-equivalent changes. EffectGuard implements this criterion directly; exact-action and snapshot methods approximate it through representation equality or state equality.

Same-action and action-change exposure. The same-action diagnostic remains important because it captures the classic stale-consent pattern: the approved action and committed action share the same representation, while late-bound state makes the effect broader. The action-change diagnostic captures resume and replanning behavior, where a syntactic guard may notice the changed command even if the committed effect remains valid. The controlled isolation shows why neither diagnostic alone is sufficient. A method that only protects action identity can miss same-action broadening. A method that only monitors state can block contract-valid replans and state-only equivalent drift.

Table 5. Effect of preview quality on EffectGuard. The full preview is the primary setting; perturbed previews test robustness.

Preview setting	Unsafe	Unnec. block	Crit. recall	Detect recall
Full preview	0.0%	0.0%	100.0%	100.0%
Stale alias	21.3%	1.5%	88.7%	44.5%
Row truncation	3.3%	0.0%	97.7%	91.3%
Amount rounding	3.3%	0.0%	92.6%	91.3%
Conservative unknowns	0.0%	91.8%	47.7%	100.0%

Table 6. Recovery after EffectGuard blocks a broadened committed effect.

Group	Cases	Unsafe final	Safe task	Recovery	Extra turns
Benign controls	50	0	50/50	0/0	0.00
Blocked effects	50	0	8/50	8/50	1.12
Overall	100	0	58/100	8/50	0.56

The effect-contract view covers both cases with a single semantic condition.

5.4. Robustness to effect-preview quality

The primary guarantee assumes a sound runtime effect adapter. We test this boundary by perturbing runtime previews while keeping the measurement evaluator independent. Under the full adapter, EffectGuard has 0.0% unsafe commits and 0.0% unnecessary blocks. Perturbations such as stale alias resolution, row preview truncation, and amount rounding reintroduce unsafe commits (Table 5). Conservative unknown handling restores unsafe commits to 0.0% and unnecessarily blocks 91.8% of non-broadened rows. Thus, effect-preview quality is part of the systems assumption; unknown handling is a safety–utility policy choice.

Critical-field recall measures whether the runtime preview preserves consent-relevant fields such as recipients, affected rows, amounts, attachment identities, external-party labels, and visibility. Broadening-detection recall measures whether the preview detects that the committed effect is outside the approved contract. Conservative unknown handling treats missing consent-critical fields as requiring escalation or reapproval. The adapter study therefore reports the trusted computing boundary of EffectGuard rather than a separate model capability.

5.5. Recovery after blocking

We ran a small recovery study to test whether agents can recover after EffectGuard blocks an unsafe committed effect. The study samples 100 cases: 50 blocked broadened-effect cases and 50 benign controls. EffectGuard maintained consent safety throughout: unsafe final states were 0. Benign controls completed safely in 50/50 cases. Recovery on blocked cases was modest, succeeding in 8/50 cases (Table 6). Many failed recoveries required reapproval or produced a consent-safe action that fell short of the original task. Safe blocking therefore leaves task recovery as a separate design problem.

Failure taxonomy and practical implications. A follow-up taxonomy of the 50 blocked exposed cases clarifies why recovery is modest. Thirty-four cases ended with the agent declining to issue a recovery call and indicating that renewed approval was needed; five repeatedly hit the contract boundary; and three executed a consent-safe action that fell short of the original task success predicate. These outcomes show that an effect difference can preserve safety, and a product still needs an explicit reapproval path and a task-aware recovery policy to recover utility. In practice, approval interfaces should display resolved effect fields rather than only tool calls. Tool interfaces should expose concrete execution or versioned handles when possible, because this reduces ambiguity at commit. When sound extraction of a critical field is unavailable, the runtime should surface uncertainty and avoid silently treating unknown fields as safe. Thus, EffectGuard is best understood as a consent-contract layer that complements least privilege, exact-action integrity, and human reapproval workflows.

6. Discussion

Consent-boundary design. The experiments suggest a design principle for state-changing agents: approval should be placed at the boundary where the user can understand and where the system can enforce. Tool names are convenient for frameworks, and argument payloads are convenient for logging, yet users usually approve outcomes. This gap becomes operationally important when late-bound references let the executor resolve the same command against changed aliases, predicates, defaults, groups, or resource identities. Capability permissions, exact action binding, state monitoring, and resolved-argument approval remain useful layers. The contribution of effect-bound approval is to define the semantic contract that resolves their conflict: a changed representation can be acceptable when the effect remains within consent, and an unchanged representation can be unacceptable when the effect broadens.

Implications for tool-interface design. Interface design changes both model behavior and runtime enforceability. Concrete execution gives the agent a way to express the final outcome directly, while handle-oriented tools remain attractive because aliases, groups, predicates, and defaults are concise and natural. Versioned references offer an intermediate point: they preserve compact handles while exposing evidence that can be checked at commit. A deployed system should therefore treat approval as part of tool-interface design rather than as an external confirmation dialog attached after the fact.

Scoped guarantee and recovery. EffectGuard provides a runtime guarantee under a clear trusted computing base: the preview adapter must expose consent-relevant fields accurately, and the executor must commit the previewed effect. Within that boundary, the guard blocks committed effects outside the approved predicate and allows effects inside it, including monotone narrowing. The recovery pilot emphasizes that enforcement and recovery are different problems. A guard can preserve consent safety by blocking a broadened effect, and the user may still need a practical reapproval path to finish the task.

7. Related Work

Tool-using agents build on reasoning-and-acting methods such as ReAct (Yao et al., 2023), tool-use learning such as Toolformer (Schick et al., 2023), modular tool systems such as MRKL (Karpas et al., 2022), and multi-agent frameworks such as AutoGen (Wu et al., 2024). Stateful environments including AgentDojo, ToolSandbox, tau-bench, and AppWorld evaluate agents interacting with tools, users, and mutable state (Debenedetti et al., 2024; Lu et al., 2025; Yao et al., 2025; Trivedi et al., 2024). Our work focuses on a deployment boundary inside such settings: whether approved state-changing actions still commit the approved effect.

Runtime safety systems such as ToolEmu, AgentSpec, ProbGuard, and MiniScope study risk identification, runtime enforcement, probabilistic monitoring, and least-privilege authorization for tool-calling agents (Ruan et al., 2024; Wang et al., 2025; 2026; Zhu et al., 2025). These systems ask which tools or behaviors should be allowed by policy. We ask a narrower consent-integrity question: once a user has approved a specific action instance, what must remain true at commit time? Exact-action integrity binds approval to the action representation. EffectGuard binds approval to the committed effect.

Action, state, and effect integrity. The consent boundary studied here sits between action integrity and state integrity. Action integrity ensures that the command presented for approval is the command submitted later. State integrity ensures that selected state predicates or resource versions remain unchanged. Effect integrity asks whether the outcome that will be committed still satisfies the approved consent predicate. These notions overlap and differ. Action equality can hold while effects broaden through state drift. State equality can fail while the committed effect remains equivalent or narrower. Effect-bound approval therefore targets the semantic object closest to user consent.

8. Limitations

Scope of evidence. Our evaluation is a controlled study of representative state-changing tools. The domains are stylized to isolate consent-boundary behavior, and controlled exposure should be read as a stress-test quantity rather than natural prevalence.

Trusted infrastructure. EffectGuard relies on a runtime preview adapter that exposes consent-relevant fields accurately and an executor that commits the previewed effect. The robustness study makes this dependency explicit and shows the safety–utility tradeoff of conservative unknown handling.

Human and recovery factors. The recovery pilot is intentionally small. Future work should study richer production workflows and user-facing presentations of effect contracts. The central systems point remains: when consent-relevant effects can be previewed, the approval boundary can be placed on the committed effect itself.

9. Conclusion

Stale consent arises when approval binds to a tool call and execution commits a broader effect. We formalize approval as an effect contract and implement EffectGuard, which checks the commit-time effect against the approved consent predicate. Across two models and 1500 valid controlled proposals, EffectGuard commits zero unsafe effects under a sound runtime adapter; capability, policy, and exact-action boundaries either miss broadened effects or unnecessarily block benign changes. The result is a bounded systems claim: for state-changing tool agents, approving the effect is a safer consent boundary than approving only the tool call.

References

Debenedetti, E., Zhang, J., Balunović, M., Beurer-Kellner, L., Fischer, M., and Tramèr, F. AgentDojo: A dynamic environment to evaluate prompt injection attacks and defenses for LLM agents. In *Advances in Neural Information Processing Systems*, 2024. URL https://arxiv.org/abs/2406.13352.

Karpas, E., Abend, O., Belinkov, Y., Lenz, B., Lieber, O., Ratner, N., Shoham, Y., Bata, H., Levine, Y., Leyton-Brown, K., Muhlgay, D., Rozen, N., Schwartz, E., Shachaf, G., Shalev-Shwartz, S., Shashua, A., and Tenenholtz, M. MRKL systems: A modular, neuro-symbolic architecture that combines large language models, external knowledge sources and discrete reasoning. *arXiv preprint arXiv:2205.00445*, 2022. URL https://arxiv.org/abs/2205.00445.

Lu, J., Holleis, T., Zhang, Y., Aumayer, B., Nan, F., Bai, H., Ma, S., Ma, S., Li, M., Yin, G., Wang, Z., and Pang, R. 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, Albuquerque, New Mexico, 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.findings-naacl.65. URL https://aclanthology.org/2025.findings-naacl.65/.

Microsoft. Using function tools with human-in-the-loop approvals. Microsoft Agent Framework Documentation, 2026. URL https://learn.microsoft.com/en-us/agent-framework/agents/tools/tool-approval. Accessed May 2026.

Ruan, Y., Dong, H., Wang, A., Pitis, S., Zhou, Y., Ba, J., Dubois, Y., Maddison, C. J., and Hashimoto, T. ToolEmu: Identifying the risks of LM agents with an LM-emulated sandbox. In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2309.15817.

Schick, T., Dwivedi-Yu, J., Dessì, R., Raileanu, R., Lomeli, M., Hambro, E., Zettlemoyer, L., Cancedda, N., and Scialom, T. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems*, 2023. URL https://arxiv.org/abs/2302.04761.

Trivedi, H., Khot, T., Hartmann, M., Manku, R., Dong, V., Li, E., Gupta, S., Sabharwal, A., and Balasubramanian, N. 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*, pp. 16022–16076, Bangkok, Thailand, 2024. Association for Computational Linguistics. URL https://aclanthology.org/2024.acl-long.850/.

Wang, H., Poskitt, C. M., and Sun, J. AgentSpec: Customizable runtime enforcement for safe and reliable LLM agents. *arXiv preprint arXiv:2503.18666*, 2025. URL https://arxiv.org/abs/2503.18666.

Wang, H., Poskitt, C. M., Wei, J., and Sun, J. ProbGuard: Probabilistic runtime monitoring for LLM agent safety. *arXiv preprint arXiv:2508.00500*, 2026. URL https://arxiv.org/abs/2508.00500.

Wu, Q., Bansal, G., Zhang, J., Wu, Y., Li, B., Zhu, E., Jiang, L., Zhang, X., Zhang, S., Liu, J., Awadallah, A. H., White, R. W., Burger, D., and Wang, C. AutoGen: Enabling next-gen LLM applications via multi-agent conversation. In *Conference on Language Modeling*, 2024. URL https://arxiv.org/abs/2308.08155.

Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., and Cao, Y. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023. URL https://arxiv.org/abs/2210.03629.

Yao, S., Shinn, N., Razavi, P., and Narasimhan, K. R. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. In *International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2406.12045.

Zhu, J., Tseng, K., Vernik, G., Huang, X., Patil, S. G., Fang, V., and Popa, R. A. MiniScope: A least privilege framework for authorizing tool calling agents. *arXiv preprint arXiv:2512.11147*, 2025. URL https://arxiv.org/abs/2512.11147.

A. Supplementary Approval Mechanisms

The main text focuses on the consent boundaries most relevant to the central comparison. We also implemented several additional approval mechanisms and ablations during the study. Tool-level approval grants a risky capability, such as email sending or database writing. Session permission grants a class of capabilities for the duration of an interaction. Surface-argument approval binds consent to the exact structured arguments proposed by the agent. The no-recheck ablation stores an approval-time effect contract and skips the commit-time preview; it isolates the importance of revalidating the effect immediately before execution. These variants are useful for auditing the design space because they correspond to natural engineering choices: approving capabilities, approving commands, approving state, or approving effects.

B. Commit-Action Exposure Definition and Controlled Outcome Classes

The primary exposure definition executes the commit-action candidate under the commit state if such a candidate exists; otherwise it executes the approved action under the commit state. This definition aligns the exposure measure with the action each guard sees at commit time. Same-action exposure is retained as a diagnostic for stale consent caused purely by state drift.

Table 7. Controlled outcome classes used to separate representation changes from effect changes.

Outcome class	Action relation	Effect relation
Exact no-change	same action	same effect, no relevant state change
State-only equivalent drift	same action	same effect after state change
Same-action broadening	same action	committed effect becomes broader
Equivalent action change	different action	committed effect remains within consent
Broader action change	different action	committed effect becomes broader

Table 8. Valid controlled rows by action/effect outcome class. These counts support the isolation study in the main text.

Model	Exact no-change	State-only equivalent	Same-action broadening	Equivalent action change	Broader action change
gpt-5.4-mini	122	170	80	158	202
gpt-5.5	128	156	132	192	160

C. Supplementary Interface-Design Results

Table 9 provides the full count version of the proposal-style analysis. The concrete-execution interface shifts proposals toward resolved concrete effects, whereas the dynamic-handle, hybrid, and versioned-handle interfaces induce more handle-based proposals. This supports the interpretation that stale-consent exposure is shaped jointly by tool-interface design and agent proposal style.

Table 9. Proposal style by tool-interface design and model. Counts are over valid controlled proposals.

Model	Interface style	Valid	Dynamic proposals	Dynamic rate	Concrete proposals	Concrete rate
gpt-5.4-mini	Concrete-execution	192	48	25.0%	144	75.0%
gpt-5.4-mini	Dynamic-handle	180	108	60.0%	72	40.0%
gpt-5.4-mini	Hybrid	168	96	57.1%	72	42.9%
gpt-5.4-mini	Versioned-handle	192	108	56.2%	84	43.8%
gpt-5.5	Concrete-execution	192	48	25.0%	144	75.0%
gpt-5.5	Dynamic-handle	192	144	75.0%	48	25.0%
gpt-5.5	Hybrid	192	144	75.0%	48	25.0%
gpt-5.5	Versioned-handle	192	156	81.2%	36	18.8%

D. Supplementary Effect-Preview Robustness Results

The main text reports representative preview-quality perturbations. Table 10 gives the full set used in the robustness study. The full preview row is the primary deployment assumption. Perturbed rows study how incomplete effect previews change the safety-utility tradeoff. Conservative unknown handling restores zero unsafe commits in this controlled setting and substantially increases unnecessary blocking.

Table 10. Effect of preview quality on EffectGuard across both models.

Preview setting	Valid	Commit exposure	Unsafe	Unsafe rate	Unnec. blocks	Unnec. block rate	Field recall	Broadening recall
Full preview	1500	574	0	0.0%	0	0.0%	100.0%	100.0%
Stale alias resolution	1500	574	320	21.3%	14	1.5%	88.7%	44.5%
Row-preview truncation	1500	574	50	3.3%	0	0.0%	97.7%	91.3%
Amount rounding	1500	574	50	3.3%	0	0.0%	92.6%	91.3%
Missing attachment identity	1500	574	0	0.0%	180	19.4%	89.6%	100.0%
Missing external-recipient label	1500	574	0	0.0%	0	0.0%	98.5%	100.0%
Secondary-effect omission	1500	574	0	0.0%	0	0.0%	63.8%	100.0%
Wrong external-recipient label	1500	574	0	0.0%	0	0.0%	98.5%	100.0%
Conservative unknowns	1500	574	0	0.0%	850	91.8%	47.7%	100.0%
Hybrid unknown handling	1500	574	0	0.0%	850	91.8%	47.7%	100.0%

E. Supplementary Recovery Results

The recovery study separates consent safety from task recovery. Table 11 gives the detailed recovery metrics. Table 12 summarizes the primary outcome category for the 50 blocked broadened-effect cases.

Table 11. Detailed recovery metrics for the controlled recovery study.

Group	Cases	Initially blocked	Safe task success	Recovery success	Unsafe final states	Repeated block	Approvals
Benign controls	50	0	50/50	0/0	0	0/50	1.00
Blocked broadened effects	50	50	8/50	8/50	0	6/50	1.72
Overall	100	50	58/100	8/50	0	6/100	1.36

Table 12. Primary recovery outcomes among the 50 blocked broadened-effect cases.

Primary outcome	Count	Share	Extra turns	Approvals
Recovered safely	8	16.0%	1.12	1.00
Renewed approval needed	34	68.0%	1.00	2.00
Task goal incompatible with approved contract	3	6.0%	1.00	1.00
Repeated contract boundary	5	10.0%	2.00	1.40
Invalid or missing recovery action	0	0.0%	0.00	0.00

F. Effect Schema Notes

Effects include recipients, external recipients, affected rows, attachment identities, resource identifiers, account identifiers, amount, visibility, sensitivity, irreversibility, and provenance. Empty approved sets are empty permissions. Unknown fields should be surfaced explicitly; optimistic omission can be unsafe, and conservative unknown handling may preserve safety at the cost of reapproval or unnecessary blocking.

G. Implementation and Secrecy Notes

The submission bundle excludes secrets, local environment material, and low-level execution artifacts. Results are derived from the paper-facing controlled analyses using commit-action exposure measurement and independent effect evaluation.