The Irreversibility Budget: Fleet-Level Risk Accounting and Admission Control for Agent Operating Systems

Bardia Mohammadi
bmohammadi@mpi-sws.org
Max Planck Institute for Software Systems
Saarbrücken, Germany

Laurent Bindschaedler
bindsch@mpi-sws.org
Max Planck Institute for Software Systems
Saarbrücken, Germany

Abstract

Fleets of LLM agents now externalize effects that cannot be fully undone: they move money, deploy code, delete data, and disclose information. Current controls check one effect at a time, so a fleet of individually authorized agents can overdraw its principal's risk under a shared trigger while every local gate stays correct. We propose the *irreversibility budget*, a cumulative account of residual value-at-risk that a trusted runtime maintains for each principal across agents, workflows, and tenants. Treating irreversibility as a first-class resource, the runtime charges each effect its residual loss below the agent and denies the marginal effect once the aggregate would overdraw the budget. Getting the price right is hard, because effects are heterogeneous, adversarially declared, and correlated. We perform a controlled study in which per-effect gates admit fleet-level overdraws of up to 48× the tenant's risk limit while the budget holds every correctly charged run within that limit. Conservative, dependency-aware pricing remains the central open problem for a deployable design.

1 Introduction

Fleets of LLM agents now call tools, update durable state, and externalize effects whose consequences may not be fully reversible. They move money, deploy code, delete data, and disclose information. Managing scarce or dangerous shared quantities is the classic job of an operating system, which measures memory, CPU time, and I/O bandwidth, allocates them to principals, charges them at use, and defends them under contention. Agent operating systems are only beginning to emerge [16, 21], but any of them will inherit this job for a quantity today's runtimes leave unmanaged: the irreversible exposure a fleet creates in the outside world. Unlike memory, it is a residual loss that depends on the effect, its compensation, and how other agents are spending at the same time.

Existing controls do not manage this quantity. They answer local questions: one defers settlement, another validates a task's authority, a third prices a review, and a fourth insures an individual action [6, 7, 14, 15, 17, 18, 23]. None keeps a running balance across agents. Consider, for example, fifty individually capped procurement agents that all watch the same supplier price spike and, each acting rationally, decide to buy at once. Every purchase clears its local cap, yet together they commit a seven-figure position no reviewer approved. Each action is allowed, but the fleet overdraws. Accounting for this before commit is hard: effects differ in how much loss survives them, an agent can misdeclare or split an effect to slip under a cap, and a shared trigger correlates losses that a naive account would simply add up.

We propose the *irreversibility budget*, a cumulative account of residual value-at-risk that a trusted runtime maintains for every *principal*: an agent, the workflow it runs in, or the tenant that owns them. The key idea is to treat irreversibility as a first-class resource, the way an operating system treats memory: a trusted runtime below the agent charges each effect the loss expected to survive recovery, adds it to one running balance per principal, and then denies or escalates the next effect once that balance would exceed the authorized budget. This meets the three difficulties in turn: pricing residual loss rather than face value handles heterogeneity, charging below the agent stops a compromised model from misdeclaring or splitting an effect to slip under a cap, and a single shared balance turns a correlated burst into a visible overdraw that no per-effect gate can see. The outcome is that existing per-effect controls stop being independent gates and become spending policies over one shared resource. Realizing it takes a resource model that defines the quantity and the budgets it is charged against, plus a runtime that prices and reserves against them, both shown on the procurement example in Figure 1.

We evaluate the abstraction with a controlled simulation of this procurement fleet, a study of public agent traces, and a ledger microbenchmark. The results support the claim and mark its limit. Local gates approve every purchase, yet the fleet overdraws its risk limit by 2.4× on average, and by 48× once it has grown to a thousand agents. The budget holds every run within the limit at every size, and because it prices effects by type it admits more useful work than a flat cap at the same safety. Its weakness is pricing: when effect types are misdeclared, or a shared trigger correlates losses that the charges assumed independent, realized loss can still exceed what the ledger accounted for. The ledger itself is cheap. Making those charges conservative and dependency-aware is the open problem.

The paper makes three contributions: irreversible exposure as a typed, cumulative admission quantity for agent operating systems (§3), a trusted runtime that combines risk-typed effects, hierarchical reserve-confirm-cancel ledgers, and value-at-risk admission control (§4), and a controlled feasibility study that isolates the fleet-level composition failure and shows dependency-aware pricing to be the central open requirement (§5).

2 Background and Motivation

A running example. We make the procurement fleet concrete and reuse it throughout. Each of the fifty agents owns one product line and may place supplier orders up to $50k under a per-agent rate limit, and the tenant tolerates $250k of unhedged exposure over a trading day. On an ordinary day the agents buy independently and stay well inside that envelope. However, imagine a shared trigger where a supplier price-spike alert makes "buy now" rational for

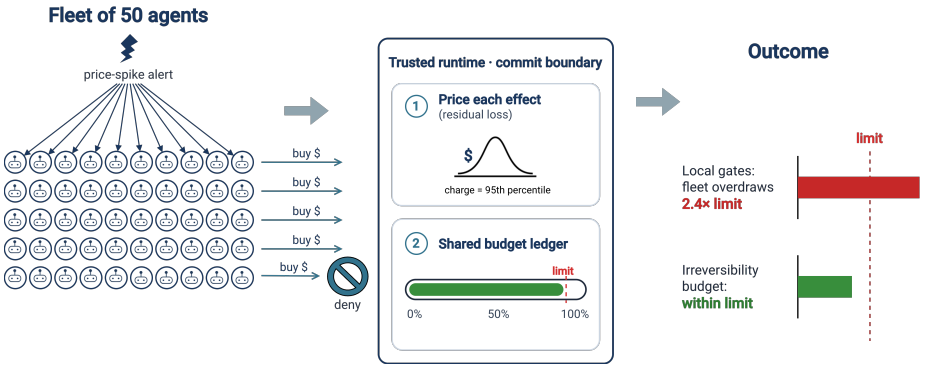

Figure 1: The irreversibility budget on the running example. Fifty procurement agents share one trigger and each proposes a locally authorized purchase (left). A trusted runtime prices each effect by its residual loss and charges it against one shared ledger, denying the marginal effect once the aggregate would exceed the tenant's limit (center). Per-effect gates overdraw by 2.4×, the budget stays within the limit (right).

every agent at once. These are the parameters simulated by the evaluation in §5.

Why it generalizes. This failure is not special to procurement. It appears wherever agents change the outside world: payroll runs, infrastructure automation, data deletion, customer communication, and incident response. Operators in these settings already cap scarce or risky quantities with quotas, approval limits, and desk-level risk controls, precisely because local permission has never implied aggregate safety. Agent fleets revive that pressure at higher speed and with weaker principal boundaries, which turns the operator's question from whether an action is allowed into how much irreversible exposure remains before the next commit.

Why counting is not enough. Answering that question means pricing effects rather than counting them. In the example, a canceled draft order, a completed supplier order, and a signed long-term contract carry very different residual loss once compensation, recovery, and revocation are exhausted. A control that treats every effect as one unit is therefore unsafe if it undercounts the irreversible ones and unusably conservative if it treats them all as maximal. Reversibility is a coordinate in the price rather than a binary gate.

The gap. Existing systems act one effect at a time. Table 1 lists the closest: Atomix defers settlement until it is safe [17], Cordon validates a task's authority before release [7], actuarial gates such as AAI quote and bind capital for a single trajectory [6], guardrails and DLP validate payloads [20], and sagas supply compensation paths [9, 10]. Each answers its local question well, and none keeps the cumulative, cross-agent balance that would have caught the fleet in the example. The missing mechanism is a pre-commit account that spans agents, workflows, and tenants.

Three requirements. The pre-commit account faces three difficulties, and each rules out a simpler design. Heterogeneous effects rule out a face-value collective cap, which charges a canceled draft and a signed contract alike and is safe only when it is uselessly conservative. Adversarial agents rule out trusting declared charges, because a compromised tool can split one large order into many sub-cap orders or misdeclare the effect type that sets the charge.

Table 1: Existing mechanisms control local effects or domain-specific counters, but none maintains a heterogeneous cross-agent exposure account.

Mechanism	Question	Control	Scope
Atomix [17]	*when*	defer settlement	per effect
Cordon [7]	*whether*	validate authority	per task
AAI [6]	*how much*	quote and bind capital	per trajectory
Guardrails / DLP [20]	*should pass*	validate payload	per call
Sagas / TCC [9, 10]	*how to undo*	compensate or refund	per effect
Quotas / rate limits	*how many*	cap a known counter	one unit
Pre-trade risk	*how exposed*	cap position risk	finance

Shared drivers rule out summing independent charges, since one price-spike alert or prompt injection makes many agents choose the same irreversible effect. A deployable control must therefore account before commit, canonicalize types, and price dependence.

3 Resource Model

§2 argued that a fleet needs a pre-commit account that prices effects, spans principals, and treats correlated exposures as more than a sum. This section defines that account as a resource. There are three parts: the charge on an effect, the budget a principal spends against, and the risk-pressure signal that drives admission.

Irreversible exposure. For an external effect e, define its charge $c(e)$ as the residual loss over a configured horizon after available compensation, recovery, and revocation have been applied. Effect outcomes are stochastic, so the runtime charges at a confidence level: $c_\gamma(e)$ is the γ-quantile of the residual-loss distribution supplied by the pricing component. The charge is a scalar in a financial domain, or a vector when money, data loss, availability, and legal exposure cannot be reduced to a common unit. Reversibility [9] enters the price as one coordinate among several rather than acting as a binary gate.

Irreversibility budget. Each principal p is assigned a windowed budget B_p, and the runtime maintains a balance of reserved plus committed exposure against it. The invariant is simple: a principal may not externalize an effect when the marginal charge would exceed its budget at the required confidence level. In the running

example, each of the fifty agents spends against its own allocation, its workflow's, and the tenant's $250k envelope, so no agent's local freedom can overdraw the fleet's ceiling. Budgets are replenished on a horizon set by the budget authority: short windows bound bursts, longer windows bound campaigns.

Risk pressure. We call the depletion signal *risk pressure*, the analog of memory pressure: as a kernel reclaims pages and throttles when free memory runs low, the runtime tightens admission as the free budget falls and, in the extreme, freezes externalization (§4). Risk pressure is a scheduling signal rather than a proof of safety, turning a fixed per-call verdict into a control loop over a scarce quantity.

4 The Runtime

The resource model says what to charge, and the runtime is the machinery that charges it. Its jobs are to make an effect's exposure observable before the effect commits, to stop a fleet from double-spending one budget, and to decide admission when the budget runs low. The mechanisms below meet these jobs, each an existing primitive moved below the agent's trust boundary, so a compromised model can neither price nor quietly spend its own risk.

Risk-typed effects. To make exposure observable before commit, tool specifications declare effect classes, exposure bounds, reversibility paths, compensation mechanisms, authority requirements, and dependency hints. Existing taxonomies of reversible, compensatable, and irreversible effects provide the initial type structure [9]. Pricing sits below the agent, so charges $c_\gamma(e)$ come from specifications and a trusted pricing service, never from the model; the charge itself is only as trusted as the specification author and its auditor. Many domains already encode partial versions of this information in approval rules, cloud quotas, and retention policies, and the runtime makes those implicit controls executable at the commit boundary.

Canonicalization. A charge is only as sound as the type it is read from, so the runtime assigns that type itself rather than accepting one from the caller. A trusted registry binds each effect class to the evidence that identifies it: the calling tool's identity and signed specification, the API endpoint and request schema the call resolves to, and the settlement receipt that later confirms what actually happened. A compromised workflow can therefore propose a final transfer, but it cannot declare that transfer refundable, because it never supplies the label. What the registry cannot resolve is charged at the most expensive matching class, so an unrecognized effect is throttled rather than silently admitted. The trusted base is wider than the ledger: the registry and its auditors, the tool wrappers, the pricing service, and principal attribution.

Charge lifecycle. Charges are initialized from the conservative bound in the effect specification, calibrated offline by replaying traces and stress cases against realized residual loss, and audited against settlement receipts, which are the only ground truth about how much recovery actually returned. They are adjusted when the pricer detects a dependency, either a correlation class declared on a workflow or a shared trigger inferred from the proposal stream, so the marginal charge for the next effect in that class rises instead of repeating the independent price. §5 breaks both the binding and the calibration, and reports what each is worth.

Algorithm 1 Value-at-risk admission (reserve).

```
input: effect e, principal path P (agent → workflow → tenant)
c ← PRICER.CHARGE(e, context, γ)            ▷ charge, never from the agent
atomically
for all ledger ℓ ∈ P do
    if ℓ.reserved + ℓ.committed + c > ℓ.B then return DENY
    end if
end for
for all ledger ℓ ∈ P do ℓ.reserved += c
end for
return reservation id                       ▷ CONFIRM commits it; CANCEL refunds it
```

Hierarchical ledgers and reservations. Per-principal ledgers expose native `reserve`, `confirm`, and `cancel` calls that keep a fleet from double-spending one risk envelope: speculative branches reserve before acting, confirm on commit, and refund on cancellation or successful compensation [10, 19]. Admission checks every ledger on the effect's principal path and reserves atomically across them. Algorithm 1 gives the reserve procedure. The ledger must guarantee four invariants: no overdraft, idempotent confirmation, crash recovery, and hierarchical roll-up.

Admission and scheduling under risk pressure. *Value-at-risk admission control* is the rule: an effect commits only if reserved and committed exposure stay within allocation at every principal on the agent-to-workflow-to-tenant path. As risk pressure rises, the runtime tightens confidence, demotes low-value spenders, expires stale reservations, batches approvals, or escalates to a rate-limited authority with explicit latency and capacity budgets. Escalation does not raise B. Instead, the authority mints a separate audited exception allocation and charges the effect against that, so the principal's ordinary budget is untouched and the extra exposure is recorded as approved rather than as headroom, because approved exposure is risk a principal chose to take, whereas the failure mode is risk the ledger failed to stop. Delegation treats *capabilities as risk currency*: attenuable tokens denominated in risk units that an agent must spend rather than merely present. Schedulers already trade fairness, priority, and utilization under scarcity; here the scarce quantity is irreversible exposure, and denial happens before the effect commits.

The contract. The ledger guarantee is narrow. It prevents overdraft in *declared charge units*, under four assumptions: authenticated principals, strongly consistent budget authorities, idempotent reservation lifecycles, and charges supplied by trusted effect specifications. It does not guarantee bounded realized loss under stale prices, correlations absent from the charge model, malicious type declarations, or misattributed workflows. The design therefore exposes two layers of guarantee: a *declared-charge bound* that the ledger enforces unconditionally, and a *realized-loss bound* that depends on calibrated, dependency-aware pricing.

5 Evaluation

We evaluate the runtime design as a controlled feasibility study rather than a deployed agent OS. Five research questions organize it: whether per-effect safety fails under composition and a budgeted ledger can bound exposure before commit (RQ1), whether the need

Table 2: Simulation parameters.

Parameter	Value
Fleet / window	50 agents / 1,000 ticks
Proposal prob. (calm, burst)	0.0005; 0.05 for 50 ticks
Purchase value	log-normal, median ∼$15k
Recovery fraction r	Beta$(6, 2)$, mean 0.75
Tolerance R / budget B	\$250k / R
Charge $c_{0.95}(e)$	0.52 v (95th-pct. residual)
Local baseline	\$50k call cap + rate limit

for accounting persists under fleet growth, fragmentation, and reactive alternatives (RQ2), what liveness and scheduling costs the budget introduces (RQ3), when risk typing and calibration help and when they fail (RQ4), and whether public agent traces support the shared-trigger correlation the simulation assumes (RQ5).

Workload. The main workload is a discrete-event procurement simulation of the running example, and Table 2 lists its parameters. Routine demand typically fills roughly 85% of the budget, a correlated market trigger raises every agent's proposal probability for 50 ticks, and an executed purchase of value v realizes residual exposure $v(1 - r)$. Charges come from declared effect types, and only the ledger microbenchmark models admission latency.

Baselines. The *local-gates* baseline enforces a per-call cap and per-agent rate limit with no shared state, and the *budget* runtime reserves the 95th-percentile charge $c_{0.95}(e) = 0.52\,v$ and denies any effect whose charge would push the reserved balance past B. We also run the strongest aggregate alternatives an operator would use: face-value and pooled-quantile caps, static per-agent partitions, a reactive circuit breaker, and hierarchical sub-budgets.

Metrics and protocol. A run *overdraws* when realized exposure that no authority approved exceeds R in any window, so overdraw measures what the ledger let through silently. We also report realized exposure relative to R, admitted routine and burst traffic, attacker value moved, sibling-workflow throughput, and reservation latency. Results are means over 300 seeded runs with 95% confidence intervals and Wilson intervals for overdraw proportions. We release the simulator, seeds, parameters, benchmark, and trace analysis at https://github.com/mpi-dsg/irreversibility-budget.

Ledger overhead. The ledger microbenchmark measures the reservation lifecycle in isolation rather than an integrated runtime, on a single host, in memory, with no persistence, replication, or crash recovery. A reserve-then-confirm across a three-level agent-workflow-tenant path costs 2.6 μs at the median and 240 bytes per live reservation, and a shared tenant root under 32 concurrent threads sustains a few $\times 10^5$ reservation cycles per second. This number bounds the accounting work rather than the cost of a durable budget authority, which must add a log write on every spend and, across hosts, a round of coordination on any violating spend.

5.1 Composition (RQ1)

We ask whether per-effect safety composes to the fleet. Every proposal stays within its per-call cap and rate limit, so the only difference between arms is whether admission also checks an aggregate budget.

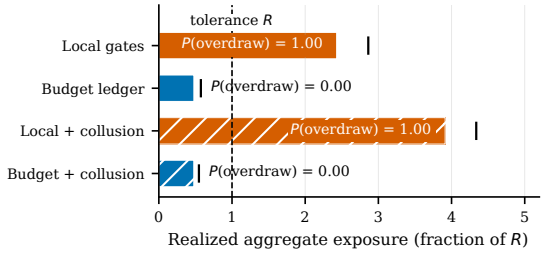

Figure 2: Realized aggregate exposure as a fraction of tolerance R, with and without a colluding fragmentation attack. Bars are means over 300 runs; ticks mark the 95th percentile. Local gates approve every action and overdraw in every run. The budget bounds exposure below R in both settings.

Results. Figure 2 contrasts the two arms. Local gates approve every purchase and still overdraw in all 300 runs, at 2.4× tolerance on average, while the budget overdraws in none of them, with a 95% upper bound of 1.3% on overdraw probability, and settles near 0.48× tolerance. The gap is structural rather than a tuning artifact: no per-call gate can see the running sum, so none can deny the marginal purchase that tips the fleet over. The budget's 0.48× occupancy is simply the arithmetic of a full ledger whose realized exposure concentrates near the mean-to-quantile ratio 0.25/0.52 of B.

5.2 Scale and alternatives (RQ2)

We ask whether the gap grows, and whether cheaper alternatives close it. We hold R fixed and grow the fleet from ten to a thousand agents, then add two variants: a fragmentation attack that splits one economic action into many sub-cap ones, and a circuit breaker that halts the fleet once observed loss reaches R.

Results. The overdraw grows with the fleet. Local gates are safe at ten agents but reach 2.4× tolerance at fifty, 9.7× at two hundred, and 48× at a thousand, because aggregate demand scales while each agent individually stays compliant. The budget holds near 0.48× at every size. The alternatives fail for instructive reasons. Fragmentation slips the full \$1.5M transfer past local gates to 3.9× tolerance, yet the ledger admits only \$459k of that value, however finely the attacker splits it, because the charge attaches to exposure rather than to call count. The circuit breaker ends at $1.03 \times R$ with instant observation and $2.2 \times R$ at a fifty-tick lag, because it reads loss only after the irreversible effect has already committed.

Takeaway. The gap widens with scale, and only a control that reserves before commit, rather than counting calls or reacting to loss, stays safe as the fleet grows.

5.3 Liveness (RQ3)

Safety by denial is easy to state and easy to abuse, so we ask what the budget costs in useful work. We sweep the budget size B/R. Separately, four scheduling arms stress hoarding, sibling isolation, campaign pacing, and re-authorization.

Results. Figure 3 shows the trade-off. A budget of $B = R$ passes routine pre-burst traffic in full and admits 14% of the correlated

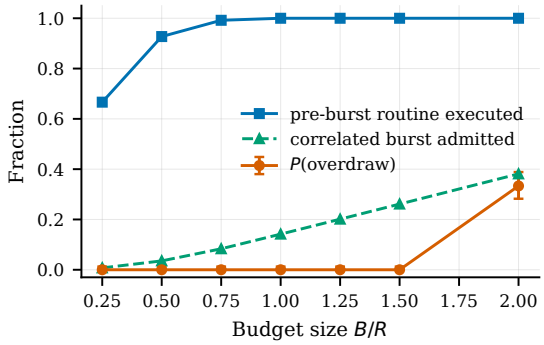

Figure 3: Safety and liveness against budget size B/R (300 runs; error bars are 95% Wilson intervals on the overdraw proportion). Larger budgets admit more burst traffic and, past $1.5R$, increase overdraw risk.

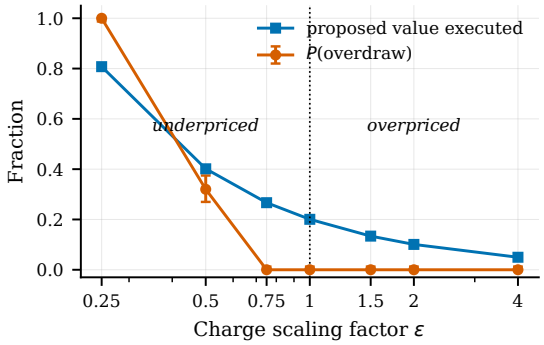

Figure 4: Mispricing sensitivity: declared charges scaled by ε relative to the 95th-percentile charge (300 runs; error bars are 95% Wilson intervals). Underpricing fails toward silent overdraw; overpricing toward throttling.

Table 3: Four secondary arms: the budget's scheduling policies and liveness stress tests, 300 runs each.

Arm	Setup	Result
Fair-share	per-principal cap $0.05B$, fragmentation	attacker value moved \$459k → \$200k; bound holds against 40 colluders
Replenishment	three windows, paced campaign	per-window resets admit 72% of a campaign; a flat $1.5R$ horizon ledger is an outage or a no-op
Hierarchy	rogue workflow, flat vs. nested ledger	sibling throughput 48% → 89% with a $0.6B$ sub-budget; tenant bound holds
Re-authorization	rate-limited authority, H effects/window	H=50 restores post-burst liveness to 93%, adding \$75k of audited exposure

burst, and even $B = R/2$ preserves 93% of routine traffic, so the price of safety falls mostly on the burst rather than on ordinary work. The sharper cost is temporal: once the window's budget is spent, post-burst traffic starves until replenishment. Enlarging the budget buys burst headroom but reintroduces overdraw, up to 33% of runs at $B = 2R$, so the knob is real but not free. Table 3 shows the scheduling arms recovering most of the lost liveness: fair-share admission curbs hoarding, per-workflow ledgers lift an innocent sibling's throughput from 48% to 89%, and rate-limited re-authorization restores post-burst liveness to 93% while making the extra \$75k of exposure audited rather than silent.

Takeaway. The budget trades burst liveness for safety on a tunable knob, and per-workflow ledgers plus paced re-authorization recover most of it without hiding exposure.

5.4 Typing and pricing (RQ4)

The ledger enforces only declared charges, so we ask whether typing effects by residual loss beats flat counting, and how charge errors actually fail. We scale declared charges by a factor ε around the true quantile, compare a typed ledger against face-value and pooled caps on a mixed refundable-and-final workload, and then break the charges' independence assumption with a burst that depresses recovery.

Results. Figure 4 shows that mispricing is asymmetric. Underpricing fails silently, overdrawing in 32% of runs already at half the true charge, whereas overpricing fails only as throttling ($\varepsilon = 4$ executes 5% of proposed value), a failure the scheduler can see and correct; charges should therefore fail conservatively. When charges are right, typing pays: the typed ledger executes 1.8× the value of a face-value cap and 1.7× the strongest pooled cap, a gain that tracks heterogeneity and shrinks to 1.2× as effect classes converge. Two failures bound that result. Misdeclaring colluding transfers as refundable re-opens the cap to \$1.0M and overdraws in 77% of runs, and a shared trigger that depresses recovery from 0.75 to 0.25 makes additive per-effect charging overdraw in 59% of runs while the ledger believes itself safe. Repricing at the burst quantile fixes the latter only with near-instant detection: a five-tick lag in a fifty-tick burst still overdraws 39% of runs, and a ten-tick lag is no better than no detector at all.

Takeaway. Typing buys real utility when charges are honest, but misdeclaration and correlated recovery let realized loss outrun the ledger's belief, which makes conservative, dependency-aware pricing the central open requirement.

5.5 Trace evidence (RQ5)

The whole model rests on shared triggers correlating losses, which the simulator cannot justify on its own, so we test it against real behavior. We analyzed 38,452 trajectories from the public τ-bench [24] and AgentDojo [8] benchmarks and asked whether irreversible effects are agent-idiosyncratic or task-determined. In τ-bench we read the 165 tasks with at least three trials as fleets of independent agents meeting the same context, and compare their agreement against a permutation null that breaks the task-to-effect link while preserving each effect's marginal rate. In AgentDojo the runs sharing one planted prompt injection, a single instruction placed in a shared resource and met by agents running different user tasks, are the fleet; we report the 35 injections with at least 20 runs. A fixed vocabulary map tags pay, refund, delete, update, send, and create as external effects, and marks a prefix read-only only when every call in it unambiguously is.

Results. External effects account for 22% of tool calls, and a single agent's effects are typically sparse and end-loaded, so no agent bursts on its own. The correlation lives across agents: which external effect fires is task-determined rather than agent-specific (z up to 198 against an independent-agent null), and one planted instruction propagates the same external effect across 84% of a heterogeneous fleet on average (minimum 48%). This is precisely the structure the additive charges in RQ4 fail to price.

Takeaway. Real traces confirm that shared context drives correlated external effects, so the regime the budget targets is the common case.

6 Discussion and Limitations

Limitations. The study establishes the accounting mechanism rather than a deployed system. It does not show that value-at-risk is the right risk measure, that calibration is solved, or that distributed ledgers stay available under partition, and it does not exercise the full reserve-confirm-cancel lifecycle, capability delegation, vector charges, or refunds. Two biases follow: liveness cost is measured pessimistically because refunds are absent, and safety is measured optimistically because prices are constructed rather than learned. These failures define the contract a deployable budget must meet: trusted effect typing, dependency-aware charging, workflow-level attribution, and denial before commit.

Pricing and budget setting. The budget is only as sound as its charges and as useful as the authority's choice of B. A business tolerance does not directly define a charge budget, and larger budgets trade liveness for weaker realized-loss bounds. Vector budgets may be needed when losses are not commensurable, since data exposure, availability damage, and reputational harm resist a common scalar in a way money does not. We use value-at-risk because operators know it and it is cheap to simulate, but the abstraction does not require it, and value-at-risk is not subadditive while expected shortfall is coherent [2]. Charges should fail conservatively, so mispricing escalates rather than silently admits.

Correlation. The tractable middle ground between independent charges and worst-case coupling remains open: correlation classes declared per workflow, shared-trigger detection from proposal streams, and marginal charges under an explicit dependency model. Detection must run early enough to affect admission before most of the burst's exposure commits.

Distributed accounting. A budget that can be double-spent provides telemetry rather than control. Across hosts, hard budgets require coordination on any violating spend, while advisory budgets reconcile later and cannot enforce safety. The runtime should separate the two, shard by principal hierarchy, and report the availability cost of strong budget authorities.

Adversarial spenders. Prompt-injected or colluding agents can fragment actions below thresholds, hoard reservations, misdeclare types, or manipulate refunds [8, 20]. Canonicalization must therefore span agents and workflows, because the shared budget itself becomes the target, and re-authorization must be rate-limited and audited rather than an unlimited human escape hatch. Our bonded-recourse design settles the compensations themselves [5].

Liveness. A budget converts some unsafe commits into denial, contention, and auditable re-authorization. It can hold the bound and still starve routine work after a burst, or let an attacker convert theft into reservation denial-of-service. Reservation expiry, workflow sub-budgets, fair-share admission, replenishment horizons, and authority overrides are scheduling policy.

7 Related Work

This work combines existing mechanisms from database transactions, OS resource accounting, financial risk control, and agent safety. The novelty is their use together as a cross-principal admission resource for typed, heterogeneous, adversarial agent effects.

Database and OS primitives. Database escrow [19] and saga compensation [9, 10] supply the reserve-confirm-cancel lifecycle, resource containers [4] supply the move of charging a principal rather than a process, and coordination avoidance [3, 11] separates invariant-preserving local updates from updates that require agreement. We apply these primitives to risk units at the tool-commit boundary, where the runtime must be able to deny before an external effect settles.

Agent effect controls. The mechanisms summarized in Table 1 protect individual effects or branches by controlling settlement time, semantic authority, or payload validity [7, 17, 20]. Our group's vision papers argue for OS-level confinement of exploratory effects [13] and for checking effects instead of trusting code [12]; the budget adds what these lack: cumulative accounting across agents, workflows, and tenants.

Risk pricing and safe learning. Settlement and underwriting systems price or compensate single actions and trajectories [14, 23]. Constrained and safe reinforcement learning caps cumulative safety cost inside one agent's policy optimization [1]. Single-agent actuarial admission gates price actions and defend against fragmentation within a trajectory [6]. Our setting moves pricing below untrusted agents and aggregates across principals, so collusion and shared drivers naturally become first-order concerns.

Governance and operational caps. A governance proposal already names an *irreversibility budget*, but treats it as an additive cap with human re-authorization [22]. Additivity suits sparse, human-supervised settings rather than dense fleets with fragmentation, misdeclaration, and correlated recoverability. Credit-card limits, cloud quotas, and pre-trade checks cap one known unit, whereas agent fleets need typed residual-loss accounting. The budget also differs from access control, which decides whether an action is allowed: the budget denies an allowed action because the aggregate would overdraw, a claim no per-call mechanism expresses.

8 Conclusion

No runtime tracks the irreversible exposure that fleets of tool-using agents build up together. We proposed the irreversibility budget, a cumulative per-principal account that a trusted runtime charges before commit, denying an effect once the fleet would overdraw. Local gates approved every action and still overdrew by up to 48×, while the budget held every correctly charged run within the risk limit. Conservative, dependency-aware pricing remains open, but irreversibility as a first-class resource gives agent operating systems a place to account for it.

References

[1] Joshua Achiam, David Held, Aviv Tamar, and Pieter Abbeel. 2017. Constrained Policy Optimization. In *Proceedings of the 34th International Conference on Machine Learning (ICML 2017) (Proceedings of Machine Learning Research, Vol. 70)*, Doina Precup and Yee Whye Teh (Eds.). PMLR, Sydney, Australia, 22–31. https://proceedings.mlr.press/v70/achiam17a.html

[2] Philippe Artzner, Freddy Delbaen, Jean-Marc Eber, and David Heath. 1999. Coherent Measures of Risk. *Mathematical Finance* 9, 3 (July 1999), 203–228. doi:10.1111/1467-9965.00068

[3] Peter Bailis, Alan D. Fekete, Michael J. Franklin, Ali Ghodsi, Joseph M. Hellerstein, and Ion Stoica. 2014. Coordination Avoidance in Database Systems. *Proceedings of the VLDB Endowment* 8, 3 (Nov. 2014), 185–196. doi:10.14778/2735508.2735509

[4] Gaurav Banga, Peter Druschel, and Jeffrey C. Mogul. 1999. Resource Containers: A New Facility for Resource Management in Server Systems. In *Proceedings of the Third USENIX Symposium on Operating Systems Design and Implementation (OSDI '99)*. USENIX Association, New Orleans, LA, USA, 45–58. https://www.usenix.org/conference/osdi-99/resource-containers-new-facility-resource-management-server-systems

[5] Laurent Bindschaedler, Quentin Botha, and Christoph Siebenbrunner. 2026. Bonded Recourse for Smart-Contract Settlement of Compensable Agent Side Effects. In *Proceedings of the 8th International Conference on Blockchain Computing and Applications (BCCA 2026)*. IEEE, Barcelona, Spain. To appear.

[6] Hao-Hsuan Chen. 2026. Insuring Every Action: An Authority Frontier Framework for Runtime Actuarial Control of Autonomous AI Agents. arXiv:2605.25632 [cs.AI] doi:10.48550/arXiv.2605.25632

[7] Zheng Chen, Hanqing Liu, Duling Xu, Dong Dong, Jialin Li, Bangzheng Pu, and Jidong Zhai. 2026. Cordon: Semantic Transactions for Tool-Using LLM Agents. arXiv:2606.17573 [cs.OS] doi:10.48550/arXiv.2606.17573

[8] Edoardo Debenedetti, Jie Zhang, Mislav Balunović, Luca Beurer-Kellner, Marc Fischer, and Florian Tramèr. 2024. AgentDojo: A Dynamic Environment to Evaluate Prompt Injection Attacks and Defenses for LLM Agents. In *Advances in Neural Information Processing Systems 37 (NeurIPS 2024)*, Vol. 37. Curran Associates, Inc., Vancouver, Canada, 82895–82920. doi:10.52202/079017-2636 Datasets and Benchmarks Track.

[9] Hector Garcia-Molina and Kenneth Salem. 1987. Sagas. In *Proceedings of the 1987 ACM SIGMOD International Conference on Management of Data (SIGMOD '87)* (San Francisco, CA, USA). ACM Press, New York, NY, USA, 249–259. doi:10.1145/38713.38742

[10] Pat Helland. 2007. Life beyond Distributed Transactions: an Apostate's Opinion. In *Proceedings of the Third Biennial Conference on Innovative Data Systems Research (CIDR 2007)*. CIDR, Asilomar, CA, USA, 132–141. https://www.cidrdb.org/cidr2007/papers/cidr07p15.pdf

[11] Joseph M. Hellerstein and Peter Alvaro. 2020. Keeping CALM: When Distributed Consistency Is Easy. *Commun. ACM* 63, 9 (Sept. 2020), 72–81. doi:10.1145/3369736

[12] Jinhao Hu, Ashvin Goel, and Laurent Bindschaedler. 2026. Don't Trust the Code, Check Its Effects. In *Proceedings of the 5th Workshop on Practical Adoption Challenges of ML for Systems (PACMI 2026)*. ACM. To appear.

[13] Jinhao Hu, Bardia Mohammadi, Ashvin Goel, and Laurent Bindschaedler. 2026. Externalization Barriers: An OS Abstraction for Untrusted Agent Exploration. In *The 2nd Workshop on OS Design for AI Agents (AgenticOS 2026)*. ACM, Prague, Czech Republic. To appear.

[14] Wenyue Hua, Tianyi Peng, Chi Wang, Jiaxin Pei, Ian Kaufman, Bryan Lim, and Chandler Fang. 2026. Quantifying Trust: Financial Risk Management for Trustworthy AI Agents. arXiv:2604.03976 [cs.AI] doi:10.48550/arXiv.2604.03976

[15] David Madras, Toniann Pitassi, and Richard S. Zemel. 2018. Predict Responsibly: Improving Fairness and Accuracy by Learning to Defer. In *Advances in Neural Information Processing Systems 31 (NeurIPS 2018)*, Vol. 31. Curran Associates, Inc., Montreal, Canada, 6150–6160.

[16] Kai Mei, Xi Zhu, Wujiang Xu, Mingyu Jin, Wenyue Hua, Zelong Li, Shuyuan Xu, Ruosong Ye, Yingqiang Ge, and Yongfeng Zhang. 2025. AIOS: LLM Agent Operating System. In *Second Conference on Language Modeling (COLM 2025)*. OpenReview.net, Montreal, Canada. https://openreview.net/forum?id=L4HHkCDz2x Also available as arXiv:2403.16971.

[17] Bardia Mohammadi, Nearchos Potamitis, Lars Henning Klein, Akhil Arora, and Laurent Bindschaedler. 2026. Atomix: Timely, Transactional Tool Use for Reliable Agentic Workflows. In *ICLR 2026 Workshop on Agents in the Wild: Safety, Security, and Beyond (AIWILD)*. OpenReview.net, Rio de Janeiro, Brazil. https://openreview.net/forum?id=UeRbEpSVUz Also available as arXiv:2602.14849 [cs.LG].

[18] Hussein Mozannar and David Sontag. 2020. Consistent Estimators for Learning to Defer to an Expert. In *Proceedings of the 37th International Conference on Machine Learning (ICML 2020) (Proceedings of Machine Learning Research, Vol. 119)*, Hal Daumé III and Aarti Singh (Eds.). PMLR, Virtual Event, 7076–7087. https://proceedings.mlr.press/v119/mozannar20b.html

[19] Patrick E. O'Neil. 1986. The Escrow Transactional Method. *ACM Transactions on Database Systems* 11, 4 (Dec. 1986), 405–430. doi:10.1145/7239.7265

[20] OWASP Gen AI Security Project, Agentic Security Initiative. 2025. Agentic AI - Threats and Mitigations. White paper, version 1.1. https://genai.owasp.org/resource/agentic-ai-threats-and-mitigations/ Version 1.1, February 2025.

[21] Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. 2023. MemGPT: Towards LLMs as Operating Systems. arXiv:2310.08560 [cs.AI] doi:10.48550/arXiv.2310.08560

[22] Subramanyam Sahoo. 2026. The Controllability Trap: A Governance Framework for Military AI Agents. In *ICLR 2026 Workshop on Agents in the Wild: Safety, Security, and Beyond (AIWILD)*. OpenReview.net, Rio de Janeiro, Brazil. https://openreview.net/forum?id=yLkGf4Uw14 Also available as arXiv:2603.03515 [cs.CY].

[23] Binyan Xu, Xilin Dai, Fan Yang, and Kehuan Zhang. 2026. When Agent Automation Becomes Profitable: Quantifying and Insuring Autonomous AI Risk through Trace-Economic Underwriting. arXiv:2606.16465 [cs.AI] doi:10.48550/arXiv.2606.16465

[24] Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. 2025. τ-bench: A Benchmark for Tool-Agent-User Interaction in Real-World Domains. In *The Thirteenth International Conference on Learning Representations (ICLR 2025)*. OpenReview.net, Virtual Event. https://openreview.net/forum?id=roNSXZpUDN Also available as arXiv:2406.12045.