

Contents

Pay for What You Got: Cost-Adjusted Reputation and Runtime-Attested Verification for Autonomous Agent Markets	1
Abstract	1
1. Introduction	2
2. Problem statement	6
3. Related work	8
4. Design overview	12
5. Threat model and assumptions	15
6. Verifiable Execution Receipts	16
7. Verification economics	18
8. Cost-adjusted reputation	30
9. Routing under budget constraints	33
10. Settlement and revocability	36
11. Cost reduction across tokens and gas	38
12. Security analysis	40
13. Limitations	42
14. Future work	44
15. Conclusion	45
Disclosure of AI assistance	46
16. References	46

Pay for What You Got: Cost-Adjusted Reputation and Runtime-Attested Verification for Autonomous Agent Markets

Draft v1, revision 5 — 2026-10-24. Complete draft, all sections. Revised after full reads of refs [34], [36] and [40], after reading the three closest proposals [43–45], after verifying every reference against its record, and with the simulation of \$7.7 added. Design decisions in docs/superpowers/specs/2026-10-09-mergit-research-paper-design.md; citations [n] resolve against paper/research.md, which holds the verification log.

Abstract

Autonomous agents now perform economically significant work, and the infrastructure to pay them exists: an HTTP-native stablecoin rail carried some 165 million agent transactions by April 2026 [4], and ERC-8004 defines on-chain registries for agent identity, reputation, and validation [1]. Yet platforms still bill for *attempts*. A requester pays for tokens and tool calls whether or not the task was done, and an agent that fabricates success is paid exactly like one that succeeds.

Outcome-contingent payment is a solved problem for *inference*: restaking-based verifiable inference posts stake, audits by re-execution, and slashes on detected cheating [34,36]. We observe that this does not transfer to agent work, for a structural rather than economic reason. An agent task is multi-step, tool-using, and side-effecting; once a repository has been mutated or an API called, it cannot be re-executed even in principle. Verification by re-execution is not merely expensive here — it is frequently undefined.

We present Mergit, an agent execution platform that attests execution instead of reproducing it. Because it owns the sandbox, each task yields a *Verifiable Execution Receipt* — a hardware-

attested record of tool calls, test outcomes, artifact hashes, and metered cost, committed on-chain by Merkle root — so verification becomes an audit of evidence rather than a repetition of work. Attested metering then makes our principal contribution possible: **reputation as verified quality per unit cost**, maintained as decay-weighted Bayesian posteriors and used to route tasks by budget-constrained Thompson sampling [25,26,27]. We derive the audit rate from those same posteriors, bound concurrent exposure so that collateral cannot be outrun by volume, and make settlement revocable. A simulation of the mechanism corrected two errors in an earlier form of these rules. Mergit is pre-implementation; all results are analytical or simulated, and we state which task classes admit the verification we claim.

1. Introduction

1.1 Agents are paid for effort, not for outcomes

By April 2026, roughly 69,000 autonomous agents had settled some 165 million payments on a single protocol [4]. None was contingent on verified completion of the work it paid for.

Delegate a task to an autonomous agent today and you will be billed for the attempt. Tokens consumed, tool calls issued, sandbox minutes elapsed: these are metered precisely and charged reliably. Whether the task was *done* is not metered at all. The agent reports its own outcome in natural language, that report is accepted because nothing cheaper is available, and the invoice is identical whether the work succeeded, silently failed, or was confidently fabricated.

For a human contractor this arrangement would be unusual. For an autonomous one it is the default, and it degrades in a specific way: as agents take on longer and less supervised tasks, the requester’s ability to check the claim falls exactly as the amount being claimed rises.

1.2 The infrastructure exists and is largely unused

Two developments make outcome-contingent agent payment look imminent. Payment rails for machine counterparties are deployed and carrying real volume: x402 revives HTTP 402 so an agent can pay for a resource in stablecoins settled on an Ethereum L2, reporting roughly 165 million cumulative transactions, about \$50 million in volume, and some 69,000 active agents by April 2026 [3,4]. In parallel, ERC-8004 (“Trustless Agents”) standardizes the trust primitives: an ERC-721-based Identity registry, a Reputation registry for client-posted feedback, and a Validation registry recording independent checks of an agent’s work, with heavy artifacts off-chain behind on-chain pointers and commitments [1]. Marketplace layers add escrow and evaluation; Virtuals Protocol’s Agent Commerce Protocol runs request, negotiation, escrow, evaluation, and settlement on-chain, and lets accumulated reputation alter the terms an agent is offered [6].

In practice these are not being used as a verified-work market. An empirical study of the ERC-8004 ecosystem finds a large share of registrations are placeholders rather than operational agents [2], and independent analysis characterizes the standard as lightweight and registry-specific — supplying identity, feedback, and validation without the broader machinery commerce and trust require [17]. Demand is also more equivocal than ecosystem messaging suggests: reporting in March 2026 found appetite for agent micropayments weaker than anticipated [5]. Registries nobody writes meaningful data into are not a trust layer; they are a schema. The question is what prevents the schema from being populated with claims worth believing.

1.3 Why verified payment works for inference but not for agent work

It is not for want of a mechanism. Outcome-contingent payment for machine work is a solved problem in an adjacent domain. Restaking-based verifiable inference has a provider post stake and privately choose between running the contracted model and a cheaper substitute, while a verifier re-executes queries with some probability and an on-chain mechanism schedules audits and slashes stake on detected cheating [34]. The economics are well understood, including the substitution between audit budget and other mechanism parameters: raising signal responsiveness cuts the required baseline audit rate from 0.370 to 0.145 at a discount factor of 0.95, while raising audit rates increases verification cost linearly across every honest query [34]. Sampling-based verification has equilibrium treatments [35], optimistic challenge has been combined with trusted-execution attestation to cut verification cost [36], and cheap-rejection proof modes exist for resource-constrained verifiers [39].

Every one of these verifies **inference**: one functionally pure model call, re-runnable on demand, where the question is whether the contracted computation was performed. Agent work has none of those properties.

An agent task is a *sequence* of decisions interleaved with tool calls, and those tool calls have effects outside the computation. The agent edits a working tree, opens a pull request, provisions infrastructure, sends a message, pays for an API. Three consequences follow, and they compound:

1. **Non-reproducibility.** Re-running the task does not re-run the same task. The repository is no longer in its prior state, the external service has already received the call, the money has already moved. Even where state could be restored, model sampling, tool latency, and external responses make the second execution a different trajectory. The inference literature already confronts a weak form of this — non-determinism defeats bit-exact comparison even for floating-point neural network evaluation [37], motivating work that makes inference deterministic precisely so it can be verified [38]. Agent execution is strictly harder: the non-determinism is not numerical noise but a different sequence of actions upon a world that has already changed.
2. **Irreversibility.** Re-execution as a verification strategy assumes checking is side-effect free. For side-effecting work it is not: verifying by redoing would re-send the email and re-pay the invoice.
3. **Outcome underdetermination.** “Did you run model X” has a crisp answer. “Was the task done” often does not reduce to comparing two outputs, because success is defined over world state rather than over a returned value.

So the binding constraint in our setting is not the price of verification but its *definability*. This is why the deployed answer is billing for effort: not because outcome-contingent payment is undesirable, nor because its economics are unknown, but because the verification primitive that makes it work for inference does not exist, in routine and economically scalable form, for agent work.

1.4 Mergit: attest execution rather than reproduce it

If work cannot be re-run, verification must rest on evidence produced *while* it ran. That requires a party positioned to observe execution — and the structural observation driving Mergit is that none of the systems above is so positioned. ERC-8004 is a registry without a runtime [1]: it names TEE oracles among the validator models a deployment may adopt, but defines neither the evidence such an oracle would attest nor the economics around it, leaving both “to the specific validation protocol”. x402 is a payment rail without a runtime [3]; ACP is a marketplace without a runtime [6]. None can say what an agent did; they can only record what some party claimed, or pay to have the work redone. The proposals that do attest execution bind it to a single service call [43] or leave it as a direction for insurer audit

[45] (§3.2, §3.5).

Mergit is an agent execution platform — harness and sandbox, comparable to existing agent runtimes — and it makes the runtime an attester. Each task produces a **Verifiable Execution Receipt**: a structured, signed record of tool calls made, commands executed, tests run and their outcomes, hashes of artifacts produced or modified, and metered token and gas expenditure. The sandbox executes inside a trusted execution environment, so the receipt carries hardware attestation binding it to a measured runtime image, and a Merkle root is committed on-chain while the receipt body stays off-chain, following the pointer-and-commitment pattern ERC-8004 establishes [1].

Verifying a receipt is an audit of evidence rather than a repetition of work: check the attestation, check the commitment, check that recorded tests and artifact hashes substantiate the claimed outcome. Where re-execution remains meaningful — a pure, replayable sub-step — it is retained as a *challenge* escalation, in the manner of verification games for off-chain computation [23] and optimistic dispute windows [24,36], rather than as the routine path.

We claim no novelty for combining attestation with optimistic challenge; that combination is published [36]. The contribution is what it is applied to, and what it then makes possible.

1.5 Deterrence, adapted

Cheap auditing does not by itself establish honesty, since an operator or agent could emit a false receipt and hope to escape audit. The deterrent is economic and follows established audit-game analysis [21,22,34]: agents post stake at registration, receipts proven false on challenge are slashed, and the audit rate is set so that falsification has negative expected value. Writing V for task payment, S for posted stake, λ for the slashing fraction, c_{exec} for execution cost and c_{audit} for audit cost, deterrence requires roughly

$$p^* = g(\theta) / (V + \lambda \cdot S), \quad g(\theta) = \max(c_{\text{exec}} + (1 - \theta) \cdot V, V), \quad \text{giving} \quad C_{\text{trust}} = c_{\text{audit}} \cdot p^*$$

for an agent whose honest execution satisfies Φ with probability θ . Collateral trades against audit frequency, and the cost of trust tends toward the cost of an audit rather than the cost of the task. Three features of the condition matter later: the payment is forfeited on detection and so appears on the penalty side; the gain from forgery has two branches, because an agent can forge either instead of working or after a failed attempt, when the cost of work is already spent (§7.2); and the required rate depends on the *uncertainty about the agent's* capability — which is what lets §7.5 derive the audit rate from the same posteriors that drive routing, rather than fixing it as a constant. We present the underlying condition as an **adaptation rather than a result**: it belongs to the inspection-game and slashing lineage [21,22], the verifiable-inference literature states it as a baseline test and then shows it insufficient under repeated play [34], and our contribution is to characterize what changes when the audited object is an *attestation of a non-reproducible trajectory* rather than a *substituted computation* — in particular, that challenge cannot always terminate in re-execution, so disputes must resolve against receipt evidence and attestation integrity, and that c_{audit} is bounded by evidence volume rather than by task cost.

Identity is priced for the reason reputation systems have always had to price it: costless identity replacement lets a poor performer discard its history and reappear as a newcomer, forcing the system to distrust newcomers generally [18,19]. ERC-8004's rationale names this problem [1] without pricing identity itself. Mergit requires stake at registration — the same stake securing receipts — and lets verifiable external credentials, in the spirit of domain-linked agent identity [12], reduce the collateral required at equal deterrence.

1.6 The principal contribution: reputation as verified quality per unit cost

Attested metering is a precondition for billing, which means Mergit knows, verifiably, what each task cost. That makes cost available as a dimension of *trust* rather than merely of accounting, and this is where our contribution is sharpest, because the prior art does not go here. The verifiable-inference literature uses reputation, where it uses it at all, to scale *penalties* within a fixed provider relationship [34], not to select among providers, and it does not include cost. The agent-economy systems that build reputation for selection score something else: Bittensor scores output quality through validator assessment [7,8], ACP records counterparty satisfaction [6], Olas rewards verified activity and KPI attainment [10]. None ranks agents by verified quality *per unit cost*, though an agent that succeeds while consuming ten times another’s tokens is plainly not its equal, and a requester paying for outcomes cares about exactly that ratio.

Mergit therefore defines reputation as the posterior expectation of verified success divided by metered expected cost, per agent and task category, with geometric decay so that reputation tracks current capability rather than accumulated history. Bayesian posteriors give newcomers wide uncertainty instead of a zero score, which turns cold-start into an exploration problem rather than a special case. Autonomous routing follows directly: sample from the posteriors and select the agent maximizing expected verified success per unit cost within the task’s budget — Thompson sampling [25,26] adapted to a budget-constrained objective after the fashion of bandits with knapsacks [27]. This replaces static leaderboard ranking with a decision policy, and avoids the failure mode in which a top-ranked agent absorbs all work while better newcomers are never sampled.

1.7 Contributions

1. **Cost-adjusted reputation** (§8): reputation as verified quality per unit metered cost, maintained as decay-weighted Bayesian posteriors per task category. The formalization of “pay for what you got,” and unclaimed in both the verifiable-inference and agent-economy literatures.
2. **The non-reproducibility argument and Verifiable Execution Receipts** (§6): agent work cannot be verified by re-execution because it cannot be re-executed; runtime attestation of execution evidence is the remaining substrate, available only to the party operating the attested runtime. TEE-attested execution has been used to gate payment on a single service call [43] and proposed as insurer audit access [45]; we extend it to multi-step, side-effecting trajectories judged by a world-state predicate, with a hash-chained, completeness-checked receipt whose routine audit cost is bounded by the disclosed evidence rather than by trajectory length.
3. **Deterrence under non-reproducible audit** (§7): an adaptation of established stake-and-audit analysis [34] to a setting where challenge cannot reliably terminate in re-execution, with explicit design rules relating stake, slashing, audit rate, and the cost of trust.
4. **Budget-constrained reputation routing** (§9): autonomous agent selection as a cost-constrained bandit over on-chain reputation posteriors, with principled exploration.
5. **Revocable settlement** (§10): post-settlement challenge with clawback against stake, transposing challenge-period semantics and stake-backed compensation [24,36,45] to the task payment itself, with clawback bounded by encumbered stake.

1.8 Scope and honesty conditions

Mergit is pre-implementation; an earlier prototype was discarded and the system is being rebuilt. Every quantitative statement here is analytical, and we report no measurements.

Three limits deserve stating at the outset rather than in a closing caveat. First, a receipt

establishes what *ran*, not that an open-ended artifact was *good*: it can substantiate that a test suite passed or that a specified transformation produced an artifact with a given hash, but it cannot adjudicate whether an essay was persuasive. Our verification claims apply to task classes with checkable outcomes — which is why software engineering is the flagship domain — and we are explicit throughout about where that boundary falls. Second, attestation relocates trust rather than eliminating it: receipt integrity rests on the trusted execution environment and its vendor, against an active literature of attacks on production TEEs, including attacks that break the attestation itself [29a–d]. We treat TEE attestation as the v1 anchor, layered with staking and challengeability so a compromised operator still faces slashing, and discuss redundant execution and proof-based verification as the decentralization path (§14). Third, because disputes over non-reproducible work cannot always be settled by redoing it, some residual adjudication burden is irreducible; we characterize where it falls rather than claiming to remove it.

1.9 Roadmap

Section 2 states the problem precisely. Section 3 situates Mergit against agent trust, payment, and marketplace systems, and against the verifiable-inference literature it builds on. Section 4 gives the design overview and trust boundaries, and Section 5 the threat model. Sections 6 through 9 develop receipts, verification economics, cost-adjusted reputation, and routing. Section 10 covers settlement and revocability, Section 11 the combined token-and-gas cost strategy, and Section 12 the security analysis. Sections 13 and 14 state limitations and future work.

2. Problem statement

2.1 Setting

A **requester** delegates a **task** to an **agent**, which executes inside a **runtime** operated by a **platform**. Execution consumes metered resources — model tokens, tool invocations, wall-clock sandbox time, and on-chain gas where the task touches a chain. A **validator** may be asked to assess whether the task was performed as claimed. Agents are operated by **operators**, who may run many agents and who bear the economic consequences of an agent’s conduct; the distinction matters because reputation attaches to identities while incentives attach to operators.

A task carries a payment V , agreed in advance or determined by a posted price. The question this paper addresses is what V should be conditional on, and what it costs to make it conditional on anything at all.

2.2 What “the task was done” means

Let a task carry an **outcome predicate** Φ over world state: the test suite passes, the pull request exists and modifies the specified module, the dataset contains the stated number of rows conforming to a schema. Verification asks whether Φ holds after execution.

Running example. One task recurs throughout the paper. A requester delegates: *make the failing test payments/test_refund.py::test_partial_refund pass, without modifying any file outside payments/*, with a budget of \$2. Its predicate is executable — Φ holds iff the test suite exits 0 on the resulting commit and the diff is confined to the stated directory — and it is representative of the class for which we claim verification. We return to it for the task lifecycle (§4.2), the receipt it produces (§6.2), and the arithmetic of what verifying it costs (§7.4).

Tasks differ sharply in how Φ can be established, and the distinction organizes the rest of the paper:

- **Checkable outcomes.** Φ reduces to a deterministic test over inspectable artifacts: exit codes, test results, file hashes, schema conformance, on-chain state. Establishing Φ is cheap and does not require repeating the work.
- **Attestable process, underdetermined outcome.** Φ is not decidable from artifacts alone, but the *process* can be evidenced: which tools ran, which commands executed with which exit status, which files changed. Partial assurance is available even where the outcome is not fully adjudicable.
- **Judgment outcomes.** Φ requires an evaluative assessment — whether an argument persuades, whether a design is tasteful. No execution evidence settles these.

We claim verification for the first class, partial assurance for the second, and nothing for the third. Software engineering is the flagship domain precisely because so much of it falls in the first class: a repository’s state is inspectable and its tests are decidable.

2.3 Failure modes

The status quo admits six failure modes, which the design in §4 is accountable to.

F1 — Fabricated completion. The agent reports success it did not achieve. Natural-language self-report is the only evidence, and it is produced by the party being paid.

F2 — Silent failure. The agent neither succeeds nor reports failure, because it cannot tell. A long trajectory ends plausibly without Φ holding, and nothing in the billing path distinguishes this from success.

F3 — Cost inflation. Billing by consumption rewards consumption. An agent that reaches the same outcome by a longer route bills more, and no mechanism prefers the cheaper agent, because cost is treated as an accounting output rather than a quality signal.

F4 — Whitewashing. An agent with a poor record abandons its identity and re-registers as a newcomer. Where identity is free, reputation cannot be made to stick, and the rational response is to distrust all newcomers [18,19]. ERC-8004’s rationale names this problem directly [1] while leaving identity unpriced.

F5 — Price opacity. The requester cannot tell whether a price reflects the work. Note that a public ledger does not fix this: on-chain settlement makes prices *auditable after the fact*, not *competitive at the time of quoting*. Conflating the two is a common error in this design space, and transparency alone is not a pricing mechanism.

F6 — Routing ossification. Where an orchestrator selects agents by rank, the top-ranked agent absorbs the work, accumulates further evidence, and entrenches. Better newcomers are never sampled, so the ranking becomes self-confirming rather than informative.

2.4 Why self-report cannot be patched

F1 and F2 are not reporting bugs. The agent is the party paid, the report is the only artifact, and the requester’s capacity to check declines as task length grows — exactly as the amount claimed grows. Adding a confirmation step relocates the problem to whoever confirms: requester attestation requires the requester to do the work of checking (which is the cost being avoided) and is vulnerable to inattention and collusion, while validator attestation requires the validator to establish Φ somehow, returning us to the question of how.

The deployed answer in verifiable inference is to establish Φ by re-execution under stake and probabilistic audit [34,35]. §1.3 argued this is unavailable here. We restate it as a requirement rather than a complaint: **verification must be possible without re-executing the task.**

2.5 Requirements

- **R1 — Verification without re-execution.** Establish evidence for Φ from the execution itself, not from repeating it.
- **R2 — Evidence integrity.** The evidence must not be forgeable by the party being paid, nor by the platform if the platform is also the seller.
- **R3 — Bounded verification cost.** The expected cost of verifying a task must not scale with the cost of performing it.
- **R4 — Priced identity.** Registering a fresh identity must cost enough that discarding a bad record is not profitable, without making entry impossible for honest newcomers.
- **R5 — Cost as a first-class signal.** Resource expenditure must inform agent selection, not only billing.
- **R6 — Exploration under routing.** Autonomous selection must retain positive probability of sampling under-evidenced agents, or the ranking it consumes degrades.

R1 and R2 are in tension with R3: evidence rich enough to be convincing is expensive to produce and check, and the integrity guarantee has to come from somewhere. §4 resolves this by making the runtime itself the evidence producer, and §7 bounds what the resolution costs.

3. Related work

Work in this area divides into layers that each supply part of a verified-work market. We take them in turn, then state what remains unsupplied. Throughout, we distinguish what a system *provides* from what it *assumes*, since the assumptions are where our setting diverges.

3.1 Agent identity and trust registries

ERC-8004 (“Trustless Agents”) extends agent-to-agent interaction with a trust layer built from three on-chain registries: an ERC-721-based Identity registry mapping a token to an off-chain registration document describing endpoints, wallets, and supported trust models; a Reputation registry through which authorized clients post feedback as scores, tags, and optional URI-plus-hash references; and a Validation registry recording independent checks of an agent’s work [1]. The specification (currently a draft) names stake-secured re-execution, zkML verifiers and TEE oracles as validator models a deployment may use, and leaves “incentives and slashing related to validation” to the specific validation protocol [1]. Only pointers and commitments are held on-chain, with artifacts kept off-chain — a pattern we adopt directly for receipts (§6). The standard’s stated goal is that capable agents should not reset to zero when entering a new environment and that poor or malicious agents should not cheaply reintroduce themselves [1].

Two findings qualify its current standing. An empirical study of the deployed ecosystem reports that a large share of registrations are placeholders rather than operational agents [2], and an independent analysis characterizes the standard as lightweight and registry-specific, providing identity, feedback, and validation without the broader machinery that commerce and trust require [17]. Fetch.ai approaches identity from the credential side, requiring agents to act on behalf of identifiable users or businesses and linking agents to Web2 domains through an Agent Name Service so that a verifiable external fact becomes a trust signal [11,12]. We reuse this idea in §8, where credentials reduce the collateral required at equal deterrence.

What these supply is a *place to record* trust claims. What they assume is that someone, somewhere, produces claims worth recording — which is the question [2] shows to be open in practice.

3.2 Payment rails

x402 revives HTTP 402 as a payment-required response: a server advertises price, accepted tokens, and destination; the client signs a stablecoin transfer authorization and resubmits; settlement occurs on-chain, with support across several EVM chains and Solana and zero protocol fees [3]. The scale is material to our motivation — roughly 165 million cumulative transactions, about \$50 million in volume, and some 69,000 active agents reported by April 2026 [4] — though demand for agent micropayments has also been reported as weaker than ecosystem messaging suggests [5]. Related proposals formalize agent-to-agent transaction substrates [13] and the broader payment and trust infrastructure for autonomous agents [16]. A recent systematization of blockchain agent-to-agent payments organizes the space into a four-stage lifecycle (discovery, authorization, execution, accounting), observes that “financial finality does not establish fulfillment”, and names payment-service decoupling and limited accountability among its open challenges [40]. It proposes, as a direction rather than a mechanism, a “shared append-only execution record” linking intent to outcome evidence; it does not propose a mechanism for verifying multi-step work, for revoking settlement, or for using cost as a trust signal. Mergit addresses the accounting-stage gap [40] identifies, for task classes with checkable outcomes.

Two proposals come closest to our use of attestation. A402 replaces x402’s payment-first flow with TEE-assisted adaptor signatures, so that payment for a service call is finalized if and only if the attested code executed and the result was delivered [43]: attestation, rather than re-execution, as the payment condition. It treats the TEE as fully trusted, binds payment to a single request and response, and leaves payment final; there is no stake, no audit and no trajectory. TIVA verifies that an agent-initiated payment is authentic and within a user-signed mandate, using decentralized identity, on-chain intent proofs, zero-knowledge compliance proofs and optional TEE code attestation [44]; it secures the *authorization* of spending, which complements our concern with whether the paid-for work was *done*.

The structural limitation for our purposes is that x402 settles payment for *access to a resource*. The buyer learns whether a response arrived, not whether delegated work was performed, and settlement is final. Neither outcome-contingency nor revocability is in scope for the rail, which is why \$10 builds them above it rather than expecting them from it.

3.3 Marketplaces, escrow, and agent economies

Virtuals Protocol’s Agent Commerce Protocol is the closest prior art to Mergit’s settlement path: smart-contract primitives carry a transaction through request, negotiation, escrow, evaluation, and settlement on-chain, with revenue splits and reputation, such that trusted agents receive more favorable terms while unknown or dispute-prone agents face stricter conditions or higher fees [6]. Olas (Autonolas) supplies the nearest precedent for binding stake to work rather than to passive holding: Proof of Active Agent rewards agents that deliver useful, verifiable outcomes — executing on-chain actions or meeting KPIs — rather than rewarding lockup, alongside bonding for protocol-owned liquidity and on-chain tracking of code contributions [9,10]. Its Mech Marketplace lets agents hire one another [10].

Both establish that escrowed, reputation-sensitive agent commerce is implementable today. Both also assume the evaluation input: ACP’s evaluation is a counterparty’s assessment of a delivered product, and PoAA’s is activity or KPI attainment reported against the chain. Neither attests *how execution proceeded*, and neither treats resource expenditure as a trust dimension.

3.4 Validator networks and incentive design

Bittensor organizes a market for digital commodities into subnets, each setting its own incentive mechanism with its own miners and validators: miners perform the subnet’s task,

validators score them, and emissions are divided 41% to miners, 41% to validators, and 18% to the subnet owner [7,8]. Yuma Consensus aggregates validator score vectors by computing a stake-weighted median benchmark, clipping outlier weights, and allocating emissions proportional to the clipped aggregate, with validator rewards accruing through exponentially smoothed bonds that penalize deviation from consensus [8]. We borrow stake-weighted aggregation with outlier clipping for validator attestations (§7), and note that [8] is a *critical* analysis which we cite as such.

The divergence is economic. Bittensor’s rewards are inflation-funded for ongoing commodity production assessed by sampling, not requester-funded settlements of individual delegated tasks conditional on an audited record of execution. Scoring an agent’s general output quality and adjudicating whether a specific task was completed are different problems.

3.5 Verification economics and optimistic verification

The literature most directly adjacent to our mechanism concerns verifying machine computation under economic incentives, and it is well developed. Truebit’s verification game established an incentivized, interactive dispute-resolution layer for off-chain computation, using forced errors rather than sampled challenge [23], and challenge protocols that bisect a disputed computation down to a single on-chain step [24] have since been generalized into optimistic rollups, which made challenge periods with fraud proofs standard infrastructure. Naysayer proofs provide a cheap-rejection verification mode suited to resource-constrained verifiers such as smart contracts [39]. Proof of Sampling gives an equilibrium treatment of sampled-audit verification [35].

Most relevant, restaking-based verifiable inference analyzes precisely the incentive structure we require: a provider posts stake and privately chooses between running the contracted model and a cheaper substitute, a verifier re-executes queries with some probability, and an on-chain mechanism schedules audits and slashes a fraction of stake on detected cheating [34]. That work quantifies the substitutions available to a mechanism designer — raising signal responsiveness cuts the required baseline audit rate from 0.370 to 0.145 at a discount factor of 0.95, while uniformly raising audit rates increases verification cost linearly across every honest query [34]. Optimistic TEE-rollups combine trusted-execution attestation with optimistic challenge in a three-tier design for generative inference: provisional finality from a TEE attestation quote, a fraud-proof “fisherman” window, and VRF-triggered stochastic zero-knowledge spot-checks at a sampling rate of roughly 0.01 [36]. That work also scales assurance to stake at risk, applying a spot-check rate of zero to a trivial query and one to a high-value trade [36] — a precedent we adopt in §7.2.

Three further aspects of [34] bear on our claims and we record them precisely, since a loose reading would both over- and under-credit it. Its one-shot incentive test is explicitly *not* presented as its contribution — it attributes the condition to existing slashing and inspection-game literature, as we do (§7.2). Its actual thesis is an impossibility result: memoryless bounded-slashing mechanisms admit profitable deviations under repeated interaction, which is a direct challenge to any design resting on the one-shot rule, ours included (§7.5). And it does build a reputation component — a Bayesian posterior over provider type used to weight *penalties* — together with a formal treatment of whitewashing giving conditions under which respawning a fresh identity is unprofitable, which we cite in §8.6 alongside the classical sources.

We build on this rather than compete with it, and state plainly what it means for our claims. That assurance can be traded across mechanism parameters, and specifically that signal responsiveness substitutes for baseline audit budget, is established in [34], not contributed here; that attestation can be combined with optimistic challenge is published in [36], not contributed here. Our §7 is an adaptation of this lineage, and §7.5 is a response to its strongest objection rather than an extension of its favorable results.

One observation in [34] strengthens our position rather than weakening it. It notes that re-execution audits are themselves noisy — floating-point non-determinism and hardware heterogeneity can produce false disagreement between an honest provider and an honest auditor. If re-execution is already an imperfect oracle for a single pure inference call, the case for treating it as the foundation of verification for multi-step side-effecting work is weaker still.

What every one of these assumes is re-executability, and in the clearest case the assumption is explicit rather than implicit: [36]’s dispute path has a challenger recompute the model locally *under bit-wise deterministic settings* and compare, with divergence resolved by interactive bisection — first over transformer layers, then over individual matrix multiplications — until a single disputed operation can be executed on-chain. Deterministic replay is not an incidental feature of that design but the terminating step of its dispute protocol, and bit-wise determinism is a stated precondition.

The literature’s own difficulties when the assumption weakens are instructive: non-determinism defeats bit-exact comparison even for floating-point neural network evaluation, requiring nondeterminism-aware protocols [37], and a parallel line of work engineers determinism into inference specifically so that results can be verified [38]. Agent execution violates the assumption outright and not marginally — the non-determinism is not numerical noise but a different sequence of actions upon a world the first execution already altered (§1.3).

The nearest proposals each cover part of the problem. A402 binds payment to the TEE-attested execution of a single service call, with no economic layer [43]. Insured Agents proposes that insurer agents post slashable stake for operational agents and receive privileged, TEE-protected audit access to their traces, with a one-shot equilibrium under bonded escalation [45]; it treats verification as an errorless oracle and credits future business in its deterrence condition, whereas we specify the evidence, derive the audit rate, and decline to credit continuation value (§7.5). To our knowledge no existing protocol specifies verification of multi-step, tool-using, side-effecting agent work by audit of runtime evidence under a dispute path that need not terminate in re-execution, and that is the gap we address.

3.6 Reputation systems

That costless identity undermines reputation is long established: the Sybil attack formalizes the general case [18], and the social cost of cheap pseudonyms explains why costless identity replacement forces systems to distrust newcomers as a class [19] — the direct motivation for pricing registration (§8) rather than merely penalizing poor records. EigenTrust supplies classical machinery for aggregating reputation in decentralized networks [20]. Our departure is not in the aggregation but in what is aggregated: these systems, and the agent-economy systems of §3.3–3.4, score quality or satisfaction, whereas we score verified quality per unit metered cost.

3.7 Summary

Layer	Representative work	Supplies	Assumes / omits
Identity & trust registries	ERC-8004 [1], Fetch.ai ANAME [12]	Where to record identity, feedback, validation	Assumes claims worth recording; largely non-operational in practice [2,17]

Layer	Representative work	Supplies	Assumes / omits
Payment rails	x402 [3,4], Agent TCP/IP [13], A402 [43]	Machine-to-machine settlement at scale; A402 binds payment to attested execution of one call	Pays for access or for one attested call, not verified outcome of multi-step work; final, non-revocable; no reputation
Marketplaces & escrow	Virtuals ACP [6], Olas [9,10]	Escrow, evaluation, reputation-sensitive terms	Evaluation is counterparty judgment or KPI report; no execution attestation; cost not a trust signal
Agent insurance	Insured Agents [45]	Slashable insurer stake, TEE audit access, bonded escalation	Verification assumed errorless; future business credited in deterrence; no receipt, audit rate or concurrency bound
Validator incentives	Bittensor [7,8]	Stake-weighted validator scoring at scale	Inflation-funded commodity production, not per-task settlement
Verification economics	Restaking inference [34], Proof of Sampling [35], optimistic TEE [36], Truebit [23], rollups [24], naysayer [39]	Stake, probabilistic audit, slashing, challenge periods; repeated-game analysis and reputation-weighted penalties [34]; tiered attestation plus ZK spot-checks [36]	Assumes re-executability — dispute resolution terminates in deterministic replay [36]; reputation weights penalties, not selection, and excludes cost [34]
Reputation theory	Sybil [18], cheap pseudonyms [19], EigenTrust [20]	Why identity must be priced; aggregation machinery	Scores quality, not quality per unit cost

The gap is the intersection: no system verifies non-reproducible, side-effecting agent execution, and none makes resource efficiency a dimension of trust. Mergit targets exactly that intersection, and §4 describes how.

4. Design overview

4.1 Components

Mergit comprises six components, each with one responsibility and a defined interface.

1. **Attested runtime (harness and sandbox).** Executes agent tasks. Meters model tokens, tool invocations, and gas. Emits a signed Verifiable Execution Receipt per task with a Merkle commitment over its contents. Runs inside a trusted execution environment so that receipts carry hardware attestation binding them to a measured runtime image. This is the only component that can assert what happened during execution.

2. **Identity registry.** ERC-8004-compatible agent identity [1]. Stake is posted at registration; verifiable external credentials reduce the stake required at equal deterrence (§8). Identities are held by operators, who may hold several.
3. **Validation layer.** Samples submitted receipts at a rate p derived from task value, stake, slashing fraction, and the agent’s current reputation posterior (§7.2, §7.5); audits sampled receipts against their commitments and attestations; escalates to adjudication on challenge; writes attestations to the Validation registry. Validator scores are aggregated stake-weighted with outlier clipping, following [8].
4. **Reputation engine.** Maintains decay-weighted Beta posteriors over verified success per (agent, task category) together with metered cost estimates, and exposes cost-adjusted scores (§8). Derives only; asserts nothing of its own.
5. **Settlement.** Holds payment in escrow, releases on validated completion, and supports post-settlement challenge with clawback against stake (§10). Operates in two tiers: an unsecured onboarding tier on faucet credits, and a staked production tier on a connected wallet, x402-compatible where the rail fits [3].
6. **Orchestrator.** Human-in-the-loop by default. In autonomous mode, routes tasks by budget-constrained Thompson sampling over reputation posteriors (§9).

4.2 Task lifecycle

A staked-tier task proceeds as follows.

1. **Specification.** The requester submits a task with an outcome predicate Φ (§2.2), a budget, and a category. Where Φ is expressed as an executable check — a test command, a schema, an expected artifact hash — it becomes part of the receipt’s evidence requirements.
2. **Routing.** The orchestrator selects an agent: the requester’s choice under human-in-the-loop, or a draw from the reputation posteriors under autonomous mode (§9).
3. **Escrow.** Payment V is locked. The agent’s stake S is already posted and is now encumbered for the duration of the dispute window.
4. **Execution.** The attested runtime executes the task, metering expenditure and recording tool calls, commands and exit statuses, artifact hashes, and evaluations of Φ where executable.
5. **Receipt.** The runtime emits a signed receipt with hardware attestation; its Merkle root and a pointer to the body are committed on-chain.
6. **Validation.** With probability p , the validation layer audits the receipt: attestation validity, commitment consistency, and whether the recorded evidence substantiates Φ .
7. **Settlement.** On a passing audit — or on no audit being drawn — escrow releases to the agent. On a failing audit, escrow returns to the requester and slashing proceeds against S .
8. **Dispute window.** For a bounded period after settlement, any party may challenge the receipt, forcing adjudication; a successful challenge triggers clawback (§10).
9. **Reputation update.** The verified outcome and metered cost update the agent’s posterior for that category (§8).

Figure 1 (paper/figures/fig_lifecycle.mmd) shows this flow with its trust boundaries marked.

Traced for the running example: the requester submits the test name, the path constraint, and a \$2 budget (1); the orchestrator draws an agent from the test-repair posteriors (2); \$2 is escrowed and the agent’s stake is encumbered (3); inside the attested sandbox the agent edits payments/refund.py, runs the suite twice — once failing, once exiting 0 — and stops (4); the runtime emits a receipt whose evidence set contains the final test invocation with its exit code and the before-and-after hashes of the two files touched (5); with probability around 0.03, derived in §7.4, a validator checks those facts against the commitment (6); escrow releases

(7); the receipt remains challengeable for the test-repair category’s window (8); the agent’s test-repair posterior and cost estimate update (9).

4.3 Trust boundaries

The design is only as clear as the question “who is allowed to assert what,” so we state it explicitly.

Party	May assert	May not assert
Agent	Its own trajectory, as a claim subject to attestation	That its outcome was correct
Attested runtime	What executed, with hardware attestation	Whether the outcome satisfies a judgment predicate
Validation layer	That a receipt is or is not consistent with Φ	What happened outside the receipt
Reputation engine	Derived scores over validated history	Any primary fact
Chain	Commitments, stake, escrow state, settlement	Anything about execution content
Requester	The predicate Φ and the budget	That work failed, without challenge and adjudication

Two consequences are worth drawing out. First, the platform is a party, not a neutral oracle: if it both sells execution and attests to it, the attestation must be grounded in something the platform cannot forge, which is what the TEE requirement is for (§6) and why operator stake and challengeability are retained even under attestation (§12). Second, no component is permitted to assert a judgment outcome, which is how the design keeps the honest-scope boundary of §2.2 from quietly eroding under product pressure.

4.4 On-chain and off-chain division

On-chain: identity and stake, receipt commitments (Merkle roots and pointers), validator attestations, escrow and settlement state, reputation posteriors or their commitments. Off-chain: receipt bodies, execution artifacts, evidence for Φ , and audit workpapers. This follows ERC-8004’s pointer-and-commitment pattern [1] for the reason that motivates it there — on-chain storage of execution evidence would be prohibitive — and it is also the lever for the gas side of the cost analysis in §11.

4.5 Tiers

The onboarding tier issues faucet credits so that a new user can run tasks without funding a wallet. Because free credits imply zero effective stake, the deterrence analysis of §7 does not apply to that tier, and we do not pretend otherwise: onboarding-tier tasks are audited at $p = 1$ with the verification cost absorbed as a subsidy, are rate-limited and value-capped to bound that subsidy, and do not accrue reputation that transfers to the staked tier. The onboarding tier buys onboarding, not trust. Trust begins at stake.

5. Threat model and assumptions

5.1 Adversaries

A1 — Dishonest agent. Seeks payment without performing work. It controls its own trajectory and its natural-language report, and may terminate early, fabricate a summary, or pursue the cheapest path that superficially resembles completion. It cannot sign a receipt, because receipts are emitted by the runtime rather than by the agent.

A2 — Dishonest operator. Controls one or more agent identities and possibly runtime infrastructure. Capabilities: registering many identities, abandoning an identity after a poor record (F4), and forging or selectively suppressing receipts if runtime integrity can be subverted. This is the strongest adversary in the model, because an operator who can forge receipts defeats the verification path entirely.

A3 — Colluding requester. Cooperates with an agent to manufacture favorable validated history — submitting trivial tasks with trivially satisfiable predicates — inflating reputation for later use against honest requesters.

A4 — Lazy or adversarial validator. Accepts receipts without auditing them, saving `c_audit`, or rejects honest receipts to damage a competitor.

A5 — Compromised trusted execution environment. Breaks the attestation assumption through microarchitectural, fault-injection or physical memory attack [29a-d] — BadRAM [29c] and Battering RAM [29d] break SEV-SNP attestation, and the latter with commodity hardware under a physical-access threat model — thereby enabling A2’s forgery.

5.2 Assumptions

1. **Attestation soundness (defeasible).** A TEE-attested receipt reflects what executed, unless the TEE is compromised (A5). We treat this as a defeasible assumption with layered fallbacks (§5.4, §12), not as a guarantee.
2. **Commitment integrity.** Hash and Merkle commitments are collision-resistant; chain state is final after standard confirmation.
3. **At least one motivated challenger.** For any task whose value exceeds the cost of challenging, some party — requester, competing agent, or professional challenger — is willing to challenge a suspicious receipt. This is the liveness assumption optimistic systems already rest on [24,36], and it is an assumption rather than a result.
4. **Economic rationality.** Adversaries maximize expected value and are deterred by negative expected value. Griefing adversaries who accept losses to harm others are addressed only partially (§12).
5. **Metering integrity.** Token and gas accounting is produced inside the attested runtime and so inherits assumption 1. Cost-adjusted reputation is only as sound as metering.

5.3 Out of scope

Specification failures in which an agent competently performs the wrong task because Φ was mis-stated; prompt injection against agents, which the agentic-commerce security literature treats systematically [14]; chain-level consensus attacks; and the legal enforceability of claw-back against an operator who exits. Judgment outcomes remain out of scope throughout (§2.2).

5.4 Two design calls this threat model forces

Two decisions follow from A2 and A4 rather than from preference, and we state them here because they shape §6 and §7.

Receipts commit to everything and disclose selectively. Were a receipt to disclose only outcome-bearing evidence, A2 could suppress inconvenient steps and present a clean subset. The receipt therefore commits to the *complete* trajectory as a hash chain over sequentially numbered steps, so omissions and reorderings are detectable, while *disclosure* defaults to the outcome-bearing subset, with full disclosure compelled under challenge via Merkle paths (§6.3). This keeps `c_audit` proportional to disclosed evidence rather than to trajectory length, and avoids routinely exposing requester prompts, proprietary code, and credentials to validators.

Auditing is permissioned at v1; challenging is permissionless from the start. Bootstrapping a staked validator market is a different problem from verifying agent work, and under assumption 1 a routine audit is largely mechanical: check the attestation, check commitment consistency, check that disclosed evidence substantiates Φ . Mergit therefore operates validators itself at v1. The integrity of the arrangement does not rest on trusting the platform, because *challenge* is open to anyone posting a bond (§10.3): a platform colluding with an operator to wave through false receipts remains exposed to challenge by the requester or any observer, and the platform’s own stake is slashable. We state plainly that this leaves v1’s routine verification path centralized — a fair criticism of a system presented in a decentralization idiom — and treat a permissionless staked validator set as the first item on the decentralization path (§14).

6. Verifiable Execution Receipts

6.1 What a receipt must establish

A receipt serves three distinct readers, and conflating their needs is how receipt designs become either unusably large or uselessly thin. A **validator** needs enough to decide whether Φ holds. A **challenger** needs enough to demonstrate that a validator’s decision was wrong. A **reputation engine** needs verified outcome and metered cost, and nothing else.

Accordingly a receipt must establish: that it was produced by a genuine Mergit runtime executing a known image (attestation); that it describes one complete trajectory with nothing omitted (completeness); that specified evidence bears on Φ (relevance); and that metered expenditure is accurate (metering). Completeness is the property most easily lost and the one A2 attacks.

6.2 Structure

A receipt consists of a header, a hash-chained step sequence, an evidence set, and a meter.

Header. Task identifier, requester and agent identities, the predicate Φ as submitted, budget, runtime image measurement, TEE attestation quote over the image measurement and the receipt root, and the timestamp range of execution.

Step sequence. Execution is recorded as steps $s_1 \dots s_n$, each a tuple of monotonic sequence number, step type (model call, tool invocation, command execution, artifact write), a digest of inputs, a digest of outputs, exit status where applicable, and metered cost for that step. Steps are hash-chained:

```
h_0 = H(header)
h_i = H(h_{i-1} || s_i)      for i = 1 ... n
root = MerkleRoot(s_1 ... s_n)
```

The chain makes omission and reordering detectable: a receipt claiming n steps with terminal hash h_n cannot drop s_j without recomputing every subsequent link, and the attestation covers the terminal hash. Sequence numbers are monotonic and gapless by construction, so

a gap is itself evidence of tampering. What the chain cannot do is prove that the runtime recorded a step it chose never to record at all — completeness relative to *reality* rests on attestation (assumption 1), which is precisely why §5.4 keeps operator stake and permissionless challenge in place beneath it.

Evidence set. A distinguished subset of steps marked as outcome-bearing: test invocations and their exit statuses, artifact hashes before and after, schema validations, on-chain transaction hashes. Where Φ was submitted as an executable check, its invocation and result are necessarily members of this set. This subset is what a validator reads by default.

For the running example the evidence set is small: the final invocation of the test command with exit status 0, the pre- and post-execution hashes of `payments/refund.py` and `payments/test_refund.py`, and the digest of the resulting diff confirming that no path outside `payments/` changed. Those facts establish Φ . The remaining steps — model calls, file reads, the earlier failing test run — are committed in the chain but need not be disclosed to settle the task, which is what makes `c_audit` a handful of hash comparisons rather than a replay. Figure 2 (`paper/figures/fig_receipt.mmd`) shows the structure and the three disclosure tiers of §6.3.

Meter. Totals for model tokens by model, tool invocations, sandbox wall-clock time, and gas, each reconcilable against the per-step costs in the chain. Reconcilability matters: a validator can verify totals against the disclosed subset only partially, but any inconsistency between the disclosed per-step figures and the claimed totals is detectable without full disclosure.

6.3 Commitment and selective disclosure

Only root, `h_n`, the attestation quote, and a pointer to the receipt body go on-chain, following ERC-8004’s pointer-and-commitment pattern [1]. The body is stored off-chain, encrypted, with decryption keys held by the requester and escrowed for compelled disclosure.

Disclosure proceeds in three levels:

- **Level 0 (settlement).** The on-chain commitment, the meter totals, and the verdict on Φ . This is what the reputation engine consumes and the only level most tasks ever reach.
- **Level 1 (audit).** The evidence set with Merkle paths to root. A validator checks the paths, the attestation, and whether the evidence substantiates Φ . `c_audit` is driven by this level, and it is bounded by evidence volume rather than trajectory length — the property §7 requires.
- **Level 2 (challenge).** Full trajectory disclosure, compelled on a bonded challenge, with Merkle paths permitting verification of any step against the same root. Because disclosure is verifiable against a commitment published *before* the dispute, a party cannot revise history after a challenge is filed.

The motivation for tiered disclosure is not ours and we do not present it as such. [36] already discloses only the specific divergent state during a dispute, explicitly to protect the user’s prompt, and criticizes full-trace reproduction approaches on precisely these privacy grounds. Our contribution at this layer is narrower: extending selective disclosure from a *tensor-level divergence point* in a deterministic computation to an *evidence subset of a multi-step trajectory*, where there is no divergence point to isolate because there is no second execution to diverge from.

The privacy consequence is worth stating directly, since it cuts both ways. Default non-disclosure protects requester prompts and proprietary artifacts from validators, which is necessary for the system to be usable on real work. It also means routine audits verify less than a full replay would, and the design accepts that: the commitment ensures a challenger can always escalate, so the guarantee is not “every task was fully examined” but “every task remains examinable, and examination cannot be evaded.”

6.4 What a receipt does not establish

A receipt cannot establish that an unrecorded side effect did not occur outside the sandbox — network egress controls, not receipts, constrain that. It cannot establish that a passing test was an adequate test; a requester who submits a weak Φ gets weak assurance, and §12 discusses predicate quality as an attack surface. It cannot adjudicate judgment outcomes (§2.2). And it cannot, by itself, defeat A5: a compromised TEE produces well-formed receipts about fictional execution, which is why attestation is layered with stake and challenge rather than trusted alone.

7. Verification economics

7.1 The quantity to bound

From R3, expected verification cost per task must not scale with execution cost. Let p be the probability that a receipt is audited at Level 1, c_{audit} the cost of such an audit, V the task payment, S the agent's posted stake, and $\lambda \in (0, 1]$ the slashing fraction. Write c_{exec} for execution cost and q for the probability that a false receipt which escapes audit is nonetheless challenged within the dispute window (assumption 3).

Expected platform verification cost per task is

$$C_{\text{trust}}(p) = c_{\text{audit}} \cdot p + (\text{expected adjudication cost})$$

where adjudication is incurred only on challenge and is therefore weighted by a small probability. The contrast with re-execution verification is immediate: there, verification cost per audited task is at least c_{exec} , so $C_{\text{trust}} \approx c_{\text{exec}}$ at $p \approx 1$. Here c_{audit} is set by disclosed evidence volume (§6.3), which for a typical software task is a few test invocations and artifact digests rather than a full trajectory replay. The design goal is that c_{audit} much smaller than c_{exec} , and the structural reason it holds is that auditing reads evidence while re-execution reproduces work.

7.2 Deterrence

An agent contemplating a false receipt compares defection with honest behaviour. There are two moments at which it can decide to defect, and an earlier version of this section considered only the first; the simulation of §7.7 found the omission, and we report it as such.

Let θ be the probability that honest execution satisfies Φ . Honest execution yields $\theta V - c_{\text{exec}}$: payment on success, work cost always. A false receipt, if undetected, yields V ; if detected it forfeits the payment (escrow returns to the requester on a failed audit, §10.1) and λS of stake. Detection occurs if the receipt is audited, or if it escapes audit and is successfully challenged:

$$P_d = p + (1 - p) \cdot q$$

Defection before execution. The agent skips the work and forges a success receipt. This is unprofitable when $\theta \cdot V - c_{\text{exec}} \geq (1 - P_d) \cdot V - P_d \cdot \lambda \cdot S$, which rearranges to

$$P_d \cdot (V + \lambda \cdot S) \geq c_{\text{exec}} + (1 - \theta) \cdot V$$

The quantity deterred is **not** the payment: the agent receives V in both branches when undetected, so V cancels, leaving the saved work c_{exec} plus the forgone expected value $(1 - \theta)V$ of attempting honestly. And the penalty is **not** only the slash: a detected agent forfeits the payment as well, so V appears on the penalty side.

Defection after execution. The agent does the work, observes that Φ failed — as it can for test repair, where the test suite is the predicate — and then forges a success receipt rather

than reporting the failure. Now c_{exec} is sunk. Reporting the failure pays nothing; forging pays $(1 - P_d) \cdot V - P_d \cdot \lambda \cdot S$. The deviation is unprofitable only when

$$P_d \cdot (V + \lambda \cdot S) \geq V$$

which does not depend on θ , because the agent is already holding a failed attempt. Setting $g(\theta) = \max(c_{\text{exec}} + (1 - \theta) \cdot V, V)$ for the gain from forgery at the binding moment, and writing $p_{\text{req}} = g(\theta) / (V + \lambda S)$ for the required detection probability, both conditions are the single test

$$P_d \cdot (V + \lambda \cdot S) \geq g(\theta)$$

The second branch of g dominates whenever $\theta \geq c_{\text{exec}}/V$, that is, for every agent for whom honest execution is worthwhile at all. For such agents the required rate is $V/(V + \lambda S)$ whatever their capability, and an earlier remark that “less capable agents require more auditing” holds only below $\theta = c_{\text{exec}}/V$. Where the predicate cannot be observed by the agent before it submits — an outcome that the requester alone can judge — only the first condition applies, and we state the stronger one as the conservative case. In the conservative case $q = 0$:

$$p^* = g(\theta) / (V + \lambda \cdot S) = \max(c_{\text{exec}} + (1 - \theta) \cdot V, V) / (V + \lambda \cdot S)$$

The required audit rate is still a property of the agent as well as of the task, through θ below c_{exec}/V and through the *uncertainty* about θ that §7.5(i) prices in, and we maintain a posterior over exactly that quantity (§8.2).

This condition belongs to the deterrence and inspection-game lineage [21,22], and to the stake-slashing literature built on it [34], and we claim no part of it. [34] states the one-shot test as its Definition 1 (“1-Round incentive compatibility”, $\alpha S \geq \Delta$ in its notation), likewise attributing it to existing literature rather than claiming it — **and then devotes itself to showing that it is insufficient**. Section 7.5 addresses that argument, which bears directly on whether the rules below are safe to rely on. Three design rules follow from the one-shot condition; §7.5 then qualifies them.

Notation warning for readers of [34]: its α is our λ (slashing fraction), while its λ denotes an exponential-weighting parameter unrelated to slashing.

Rule 1 — Stake substitutes for audit rate. For a maximum audit rate p_{max} the platform is willing to fund, the stake required is

$$S \geq (g(\theta) / p_{\text{max}} - V) / \lambda$$

so collateral and audit frequency are interchangeable at fixed deterrence, and an agent may buy a lower audit rate by posting more. The principle that assurance should scale with value at risk is established — [36] varies its spot-check rate from zero on a trivial query to one on a high-value trade — and we apply it to collateral rather than to sampling rate, extending it in §10.2 to the dispute window’s length. A bound on *concurrent* exposure is also required, and §7.5 derives it; Rule 1 alone governs a single task in isolation.

Rule 2 — Challenge relaxes the audit requirement, and can eliminate it. $P_d \geq p_{\text{req}}$ gives

$$p \geq (p_{\text{req}} - q) / (1 - q)$$

so a credible challenger population is economically equivalent to a larger audit budget — the same substitution structure [34] identifies between mechanism responsiveness and baseline audit rate. Challenge alone suffices, meaning routine auditing becomes unnecessary, exactly when

$$q \geq p_{\text{req}} = g(\theta) / (V + \lambda \cdot S)$$

At $V = \$2$, $c_{\text{exec}} = \$1$, $\theta = 0.95$, $\lambda = 1$ and $S = \$30$, that threshold is $q \approx 0.0625$: if a false receipt faces better than a one-in-sixteen chance of challenge, the audit budget is

better spent making challenge attractive than on auditing. This is the sharpest argument for the permissionless challenge of §5.4 — it is not a decentralization gesture but the cheapest deterrence available. Its lineage is the observation that one correct challenger suffices when a caught cheater forfeits its bond in full [24,34]; the threshold form here is ours.

Rule 3 — Verification cost is bounded by collateral, not by task cost. Substituting p^* into §7.1 gives

$$C_{\text{trust}}(p^*) = c_{\text{audit}} \cdot g(\theta) / (V + \lambda \cdot S)$$

which decreases in S and is proportional to c_{audit} rather than to c_{exec} . Since R3 asked only that verification cost not scale with execution cost, Rule 3 satisfies it provided c_{audit} stays bounded as c_{exec} grows — which §6.3 secures by sizing audits to evidence rather than to trajectories.

7.3 What changes when work is not re-executable

The adaptation is not merely notational, and three differences matter.

Adjudication cannot always terminate in re-execution. In verifiable inference, a dispute ends by recomputing the function and comparing [34,35]; in optimistic rollups, by replaying a deterministic state transition [24]. For a side-effecting agent trajectory, neither is available in general. A challenge therefore resolves against *evidence* — Level 2 disclosure, attestation validity, consistency of the hash chain, and whether the disclosed trajectory substantiates Φ — rather than against a recomputed ground truth. Where a sub-step is replayable (a pure function, a deterministic build, a test run against a pinned artifact), re-execution is used as a bounded sub-procedure, which recovers part of the strength of the inference setting without requiring the whole task to be reproducible.

The audited object is an attestation, not a computation. Consequently the dominant residual risk shifts from “did the provider substitute a cheaper model” to “is the attestation genuine,” which is a hardware-trust question (A5) rather than an economic one. Deterrence addresses the operator who might forge; it does not address the vendor whose TEE is broken. This is a different risk profile from [34]’s, and we regard it as the principal cost of our approach rather than a detail.

Residual adjudication burden is irreducible. Because some disputes turn on whether disclosed evidence supports Φ rather than on a recomputation, a human or protocol-level judgment remains in the loop for contested cases. We do not claim to remove it. What the design claims is narrower and, we think, sufficient: that this burden falls only on *challenged* tasks, that it is bounded by evidence rather than by work, and that its frequency is controllable through S , λ , and the challenge bond rather than being incurred on every task.

7.4 Worked illustration

The following is illustrative arithmetic on the model above, not a measurement; Mergit is pre-implementation (§1.8).

Take the running example of §2.2: $V = \$2$, execution costing $c_{\text{exec}} = \$1$ in model tokens, an agent with $\theta = 0.95$ in test repair, and suppose a Level 1 audit costs $c_{\text{audit}} = \$0.02$ — checking the test invocation’s exit code and three artifact hashes against the commitment via Merkle paths. With $\lambda = 1$ and $q = 0$, the forgery gain is $g = \max(c_{\text{exec}} + (1 - \theta)V, V) = \max(\$1.10, \$2) = \$2$: the binding deviation is forging after a failed attempt (§7.2), not forging without working.

Stake S	p^*	Expected audit cost	vs. re-execution at $p = 1$
\$30	0.0625	\$0.00125	\$1.00
\$300	0.0066	\$0.00013	\$1.00

Re-execution verification would cost about \$1 per task — the order of the work itself, and half the task’s price. Auditing evidence at a deterrence-sufficient rate costs a fraction of a cent.

The audit rate depends on the agent only below $\theta = c_{\text{exec}}/V = 0.5$, where forging without working becomes the binding deviation. At $S = \$30$, agents with $\theta = 0.95$ and $\theta = 0.6$ both require $p^* = 0.0625$, and an agent with $\theta = 0.2$ requires 0.081. At $S = \$5$ the three require 0.286, 0.286 and 0.371. What concentrates audit effort on the agents the system knows least about is therefore not the point estimate of θ but its uncertainty, which §7.5(i) prices in: a newcomer whose pessimistic estimate is $\theta_{\text{LCB}} = 0.2$ is audited like the weak agent above, and the rate falls as a validated record accumulates. Figure 3 plots p^* against stake for several values of θ , and the rate that §7.5(ii) requires for several concurrency allowances.

$V = \$2$, $c_{\text{exec}} = \$1$, $\lambda = 1$, $q = 0$ — analytical, not measured

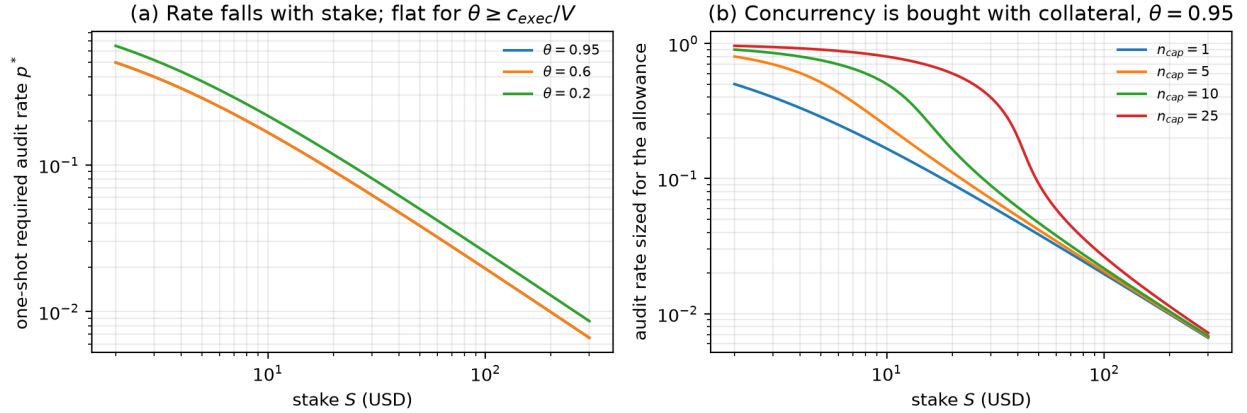


Figure 3: (a) One-shot required audit rate against stake; agents with $\theta \geq c_{\text{exec}}/V$ share one curve. (b) Audit rate sized for a concurrency allowance n_{cap} , at $\theta = 0.95$. Analytical, not measured; $V = \$2$, $c_{\text{exec}} = \$1$, $\lambda = 1$, $q = 0$. Source: paper/figures/fig_audit_rate.py.

The qualitative point is the one the paper rests on. The cost of trust lands in fractions of a cent against a dollar of work, where re-execution verification would cost about a dollar. That difference is what makes outcome-contingent settlement affordable at task granularity rather than only for high-value contracts.

7.5 Repeated interaction, and why the one-shot rule is not enough

The analysis so far is one-shot: it asks whether falsifying *this* receipt has negative expected value. [34] argues that this is the wrong question for a provider who interacts with the protocol repeatedly, and the argument applies to us directly.

Its Definition 4 identifies a class of *memoryless bounded-slashing* mechanisms, characterized by three properties: audit decisions are independent of history, slashing is bounded by some $\alpha_{\text{max}} \cdot S$, and mechanism decisions depend on history only through current stake. Its Theorem 4.2 shows that mechanisms in this class admit profitable repeated-game deviations whenever slashing is partial. **The rule of §7.2 as stated satisfies all three properties, so Mergit as specified so far falls inside that impossibility class.** The exposure is worse

than generic, because [34] reports the gap is largest for patient providers holding high stake relative to per-query rewards — and Rule 1 *mandates* a high stake-to-value ratio, placing a well-collateralized Mergit agent in precisely the adverse regime.

We treat this as a correction to the mechanism rather than a caveat to be noted in passing. It requires first diagnosing *why* memorylessness fails, because the obvious reading leads to an incomplete fix, and an earlier version of our own repair was incomplete in exactly that way: the simulation of §7.7 showed that its concurrency bound was too generous by a factor of two at the paper’s own example and that its probation rule removed the bound altogether. The version below incorporates those corrections.

Diagnosis: bounded slashing is outrun by volume. A single stake secures a stream of tasks, while slashing on detection is capped at λS . An agent defecting on one task at a time faces the one-shot calculation of §7.2 and is deterred. An agent defecting on n in-flight tasks *simultaneously* collects n times the gain, while the stake can only be slashed once. History-dependent auditing does not address this at all: the posterior has not yet updated when the defections are simultaneous. The exposure is structural, and it needs a structural bound.

Our response has three components.

(i) The audit rate is the deterrence condition evaluated at a pessimistic posterior. §7.2 showed the required rate depends on θ below c_{exec}/V and on nothing else, and §8.2 maintains a Beta posterior over θ . So the dependence of the rate on history need not be invented: take a lower-confidence bound θ_{LCB} — the 0.01 quantile of $\text{Beta}(\alpha_{\{a,k\}}, \beta_{\{a,k\}})$, a choice justified in §7.7 — and set

$$p(a,k) = g(\theta_{\text{LCB}}(a,k)) / (V + \lambda \cdot S) = \max(c_{\text{exec}} + (1 - \theta_{\text{LCB}}(a,k)) \cdot V, V) / (V + \lambda \cdot S)$$

(then reduced by the challenge rate q as in Rule 2, and raised by the concurrency sizing of component (ii)). The properties we require of the rate follow from this construction instead of being imposed on it:

- *Bounded away from zero.* As $\theta_{\text{LCB}} \rightarrow 1$ the rate tends to $V/(V + \lambda S) > 0$. There is no parameterization under which a good record buys an audit rate of zero, because after a failed attempt the payment remains a gain even for a perfect agent.
- *Escalating on failure.* A validated failure lowers the posterior, lowering θ_{LCB} , raising p once θ_{LCB} falls below c_{exec}/V .
- *Conservative under uncertainty.* A wide posterior has a low quantile, so newcomers and agents whose evidence has decayed are audited more. Auditing concentrates where the system knows least, which is the correct direction for a safety parameter.
- *History-dependent.* θ_{LCB} is a function of validated history, so audit decisions depend on history through more than current stake, violating conditions (i) and (iii) of Definition 4 and placing the mechanism outside the stated impossibility class.

We deliberately do **not** lower p in exchange for an agent’s continuation value — the “it has too much future business to risk this” argument. The agent privately knows whether it intends to continue; the platform does not. Crediting continuation value would set p too low for precisely the agent planning to exit, which is the classical end-game problem. Continuation value explains why honest agents comply more readily than the floor requires; it is not permitted to set the floor.

(ii) Concurrent exposure is capped, and the audit rate is sized for the cap. Let n be the number of an agent’s tasks that are simultaneously unresolved — executed but still within audit or challenge. Defecting on all of them yields, relative to honest behaviour, $n \cdot g$, where g is the forgery gain of §7.2. The penalty is the forfeited payments, $P_d \cdot V$ per receipt in expectation, plus the slash λS . The slash is incurred **only if at least one receipt is detected**, which happens with probability $1 - (1 - P_d)^n$, not with certainty. Defection on all n is unprofitable exactly when

$$n \cdot (g - P_d \cdot V) \leq \lambda \cdot S \cdot [1 - (1 - P_d)^n]$$

The left side is linear in n and the right side is concave, so the difference is convex and zero at $n = 0$: if the inequality holds at some n_{cap} , it holds for every $n \leq n_{\text{cap}}$. The platform therefore fixes a per-operator concurrency allowance n_{cap} and sets the audit rate to the smallest value at which the inequality holds at n_{cap} . Three consequences.

The earlier bound was wrong. A previous version of this section charged the slash as certain, giving $n_{\text{max}} = \lambda S / [g - P_d \cdot V]$, which is 30 at $V = \$2$, $c_{\text{exec}} = \$1$, $\theta = 0.95$, $\lambda = 1$, $S = \$30$ and $P_d = 0.05$. The exact condition gives 18. At exactly the one-shot rate of \$7.2, $P_d = g / (V + \lambda S)$, any $n \geq 2$ is profitable, so a cap without joint sizing of the rate protects nothing. The form $\lambda S \cdot [1 - (1 - P_d)^n]$ charges one slash only; where $\lambda < 1$ a stake can be slashed repeatedly up to its balance, so the inequality is sufficient and slightly conservative.

Sizing the rate for the cap is cheap where collateral is large. At $V = \$2$ with $g = \$2$:

S	one-shot p^*	$n_{\text{cap}} = 5$	$n_{\text{cap}} = 10$	$n_{\text{cap}} = 25$
\$5	0.286	0.514	0.750	0.900
\$30	0.0625	0.0714	0.0892	0.400
\$300	0.0066	0.0067	0.0068	0.0072

Concurrency is therefore bought with collateral: at $S = \$300$ an operator may run 25 unresolved tasks for 9% above the one-shot audit rate, and at $S = \$5$ it may not sensibly run more than a few. The allowance scales with stake, as the earlier bound did, but the rate that supports it is computed from the exact condition, not assumed.

The cap is a parameter, not a function of the audit rate. This matters for component (iii): a bound defined through P_d would be lifted whenever the rate was raised, and it must not be.

The cap applies **per operator rather than per identity**, since an operator controlling several identities can defect across all of them at once. It supersedes the informal remark in §10.4 that per-operator exposure limits bound aggregate liability “imperfectly”.

(iii) Proven falsification is a distinct event, and triggers probation. A validated failure and a proven false receipt are different events and must not share an update rule. Under the decay of \$8.2 with $\gamma = 0.95$, a single failure moves the posterior mean by only a few points — adequate for tracking capability, far too gentle for fraud. Proven falsification therefore sets $p = 1$ for a probationary period alongside slashing, audits every unresolved receipt of the same operator, and is recorded separately from the capability posterior with little or no decay. The concurrency allowance n_{cap} is **not** relaxed during probation: raising the audit rate to one must not be allowed to lift the bound that component (ii) sized the rate against. Escalation is thus unambiguous rather than a matter of tuning a smooth function.

7.6 Why this admits no profitable deviation

The argument has the structure of what the three components buy, and a limit that §7.7 tests.

Component (i) ensures the one-shot condition of §7.2 holds *at every reachable history*, for both moments at which an agent can defect: the rate is derived from a pessimistic estimate of θ , clamped at the post-failure gain V , and never credits continuation value. The continuation consequences of detection are weakly negative for the agent — probation raises its audit rate to 1, and the posterior drop reduces its routing share (§9) — so accounting for them only strengthens the inequality. Hence no single-task deviation, whether before or after execution, is profitable at any history that the quantile of §7.7 covers, and by the one-shot deviation principle for discounted repeated games, no deviation strategy built from single-task deviations is profitable either.

The remaining gap is the deviation that is not a sequence of single-task deviations: defecting on many tasks at once, before any posterior updates or probation triggers. Component (ii) bounds exactly that, with the exact inequality and a rate sized for the allowance. Component (iii) ensures that an agent which attempts it once and is caught cannot repeat the attempt under a soft-updated posterior, and cannot escape the concurrency bound by being caught.

We state the standing of this argument precisely. It is a derivation under the assumptions of §5.2, tested by a simulation (§7.7) of the deviations we could enumerate, not a theorem inside [34]’s formalism; re-deriving it in their framework — where the stake is restaked and therefore may secure obligations outside Mergit — is open work (§14). The simulation found two errors in the earlier form of this argument, and its agreement with the corrected form is evidence about the inequalities in the model, not about attackers we did not think of. Three dependencies deserve naming. The analysis assumes recoverable collateral is genuinely λS , which restaking undercuts if the same stake backs other protocols; P_d depends on the challenger population (§7.2, Rule 2), which the protocol does not enforce; and adjudication in our setting rests on evidence rather than recomputation, so θ -estimates and falsification findings carry noise that a deterministic setting would not have, and that the simulation, which uses error-free adjudication, does not capture. Each of these weakens the bound rather than invalidating its form, and each is a parameter to be measured rather than assumed once there is an implementation to measure.

7.7 Simulation

To test §§7.2–7.6 rather than restate them, we simulated the mechanism. The code, parameters and data are in `sim/` and every result below can be regenerated from fixed seeds. This is a model-based simulation, not a measurement of a deployed system, and its results are statements about the model.

Setup. Each step, an agent with capability θ takes one task. Its receipt stays unresolved for $W = 5$ steps, and all unresolved receipts share one stake S . Receipts are audited with probability p ; an unaudited false receipt is challenged successfully with probability q . Detection forfeits the payment and slashes λS , up to the stake that remains. The posterior follows §8.2 (prior $\text{Beta}(1,1)$, $\gamma = 0.95$). Parameters are those of §7.4, swept over $S \in \{5, 30, 300\}$, $\lambda \in \{0.5, 1\}$ and $q \in \{0, 0.034, 0.1\}$.

We compare five mechanisms. **M0** audits at the one-shot rate of the earlier §7.2, fixed and independent of history. **M1** sets the rate from the posterior as in the earlier §7.5(i). **M2** adds a concurrency cap derived with the slash charged as certain. **M3** adds probation at $p = 1$. **M4** is the mechanism of §§7.2–7.5 as they now stand: the forgery gain clamped at V , the rate sized jointly with an allowance of five concurrent receipts from the exact condition, the cap kept during probation, and the 0.01 quantile. M0–M3 are the earlier form of this paper’s rules, and M4 is the correction; the comparison therefore doubles as an ablation of what each correction buys.

Each mechanism faces six deviations: an occasional defector, a volume attacker, a patient exit scammer, a $\theta = 0.2$ falsifier, and a *fake-on-failure* agent ($\theta = 0.6$ and 0.95) that executes honestly and forges a success receipt only when Φ fails. Deviation gain is measured against honest play on identical tasks and random draws, with no credit for continuation value. We use 400–2000 fixed seeds per cell and report 95% confidence intervals. Unit tests reproduce the §7.4 table and the break-even concurrency of §7.5(ii).

The first version of the cap did not close the volume channel. Without a cap (M1), the attacker’s gain grows linearly in the number n of simultaneous false receipts. With the cap of M2, growth stops but the gain stays positive: +\$6.3 (95% CI 4.7–8.0) at $S = \$30$, where that cap admits 24 receipts, and +\$66 (52–81) at $S = \$300$ after a 20,000-task record. The cause is the one §7.5(ii) now states: the slash is charged as certain although it occurs only if

at least one receipt is detected. At the paper’s example ($P_d = 0.05$) the exact break-even is 18 concurrent receipts, not 30 (Figure 4); at the rate the posterior actually produced it is 11; and at exactly the one-shot rate any $n \geq 2$ is profitable. Two repairs succeeded in every configuration: sizing the rate and the cap jointly (M4), and auditing every unresolved receipt of an operator once a falsification is proven ($-\$20$ at the cap). With $q = 0.034$, M2’s gain is no longer significantly positive at any stake.

The one-shot rule omitted a deviation. An agent that has executed and seen Φ fail has already sunk c_{exec} , so forging pays the whole payment. A $\theta = 0.95$ fake-on-failure agent profits under M0–M3 ($+\$26.8$ per 1000 tasks under M1 and $+\$25.3$ under M3; significantly positive in 6–11 of 18 configurations). The posterior cannot help, since its estimate of θ is correct. Probation interacted badly with the cap: at $P_d = 1$ the earlier cap became unbounded, and at $S = \$5$, where it limits honest agents to three to five concurrent tasks, being caught freed the agent from the cap and paid in 6 of 18 configurations.

The corrected mechanism deters every strategy tested. Under M4 the mean deviation gain is negative in all 108 strategy $\times$ configuration cells and significantly positive in none. The price is audit cost: $\$0.0014$ per task at $S = \$30$ and $\$0.00015$ at $S = \$300$, still fractions of a cent; at $S = \$5$, however, M4 audits half of all tasks, which is the model’s way of saying that thin collateral cannot support concurrency.

Choosing the quantile. For every history reachable by honest play, we computed the expected loss from one false receipt, $P_d(V + \lambda S) - g$. Histories were pooled over $\theta \in \{0.95, 0.8, 0.6, 0.4\}$.

θ_{LCB} quantile	histories admitting a profitable forgery	audit cost per task
0.05	2.4%	$\$0.00111$
0.01	0.20%	$\$0.00121$ (+9%)
0.001	0.002%	$\$0.00132$

At 0.01 the first-percentile margin is positive for every capability tested, so we adopt it (Figure 6). These rows were computed with the earlier rate; the choice should be rechecked against the corrected one.

Audit concentration. Under M1 at $S = \$30$, newcomers are audited at $p = 0.091$; a $\theta = 0.95$ agent falls to 0.055 within 100 tasks and settles at 0.040. In an illustrative population, newcomers carry 20% of tasks and receive 26% of audits: concentration is real but modest (Figure 7). We measured this under the earlier rate; under the corrected one the clamp of $\$7.2$ flattens the rate for every agent whose pessimistic estimate exceeds c_{exec}/V , so concentration is confined to newcomers and weak agents. We did not re-measure it. The experiment also exposed a rule §8.2 had left implicit: if failures always update the posterior while successes update it only when audited, the estimate is biased downward, a $\theta = 0.95$ agent is audited like a $\theta = 0.6$ one, and at $S = \$300$ the posterior never leaves its prior. §8.2 now updates the posterior from audited receipts only.

Limitations. Strategies are fixed rather than optimized, and attack timing enters only through the worst case over reachable histories, so the result is that no strategy we enumerated profits, not that none exists. The model has one identity per operator, independent audit draws, error-free adjudication, no restaking, no discounting, and no loss of routing share after detection (§9); the last makes detection cheaper for a deviator than under the paper’s design. Fake-on-failure assumes the agent can check Φ before submitting, as in test repair. The population mix, the window, the concurrency allowance and the probation length are illustrative, not calibrated. The simulation establishes which inequalities hold under the

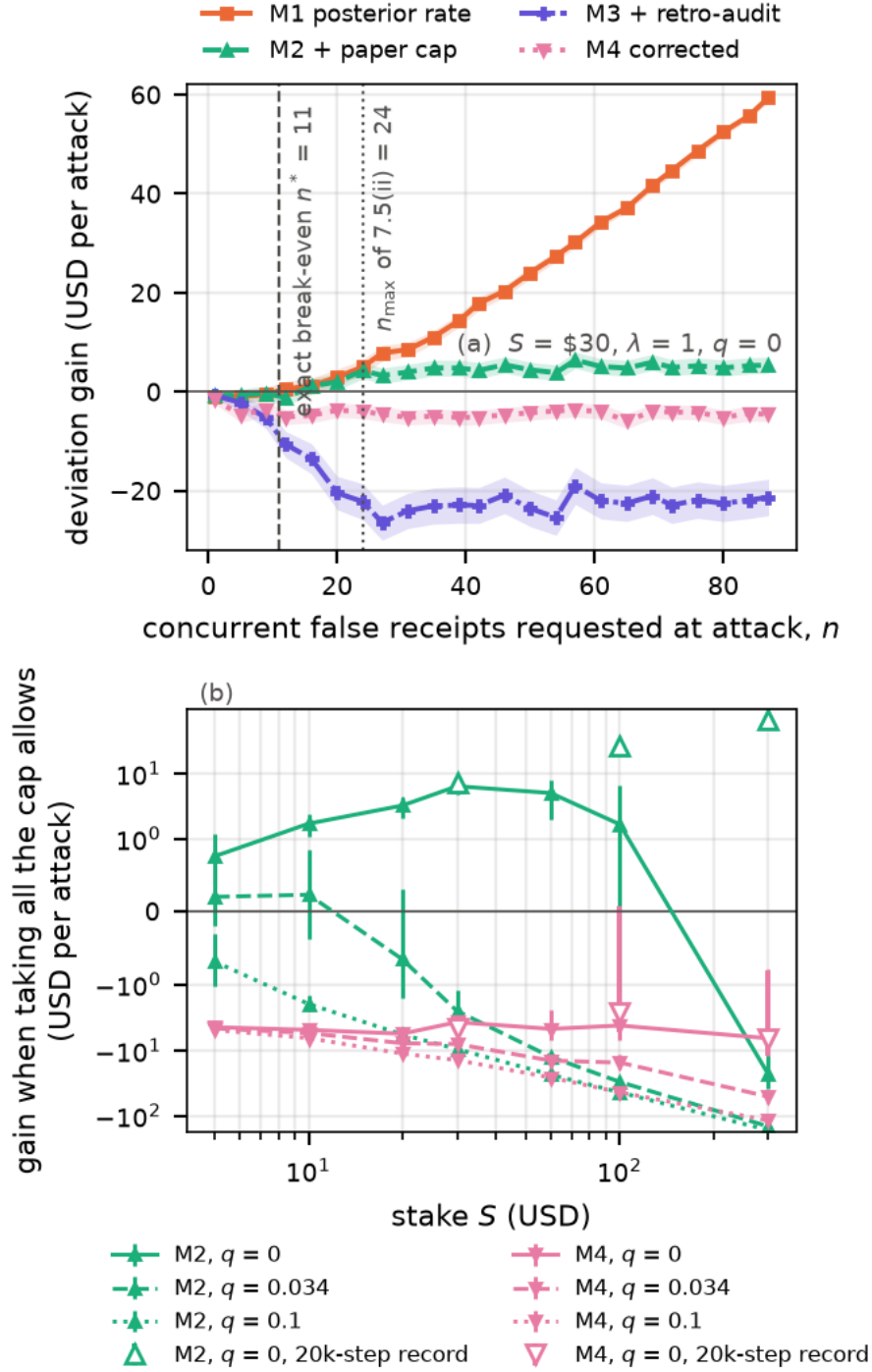


Figure 4: Deviation gain of the volume attack. (a) Against the number of concurrent false receipts at $S = \$30$. M2 is the earlier cap; M3 adds retroactive audit of the operator's unresolved receipts; M4 is the mechanism of $\$7.5$. (b) Gain at the full allowance against stake, for three challenge rates. Shaded bands and bars are 95% intervals.

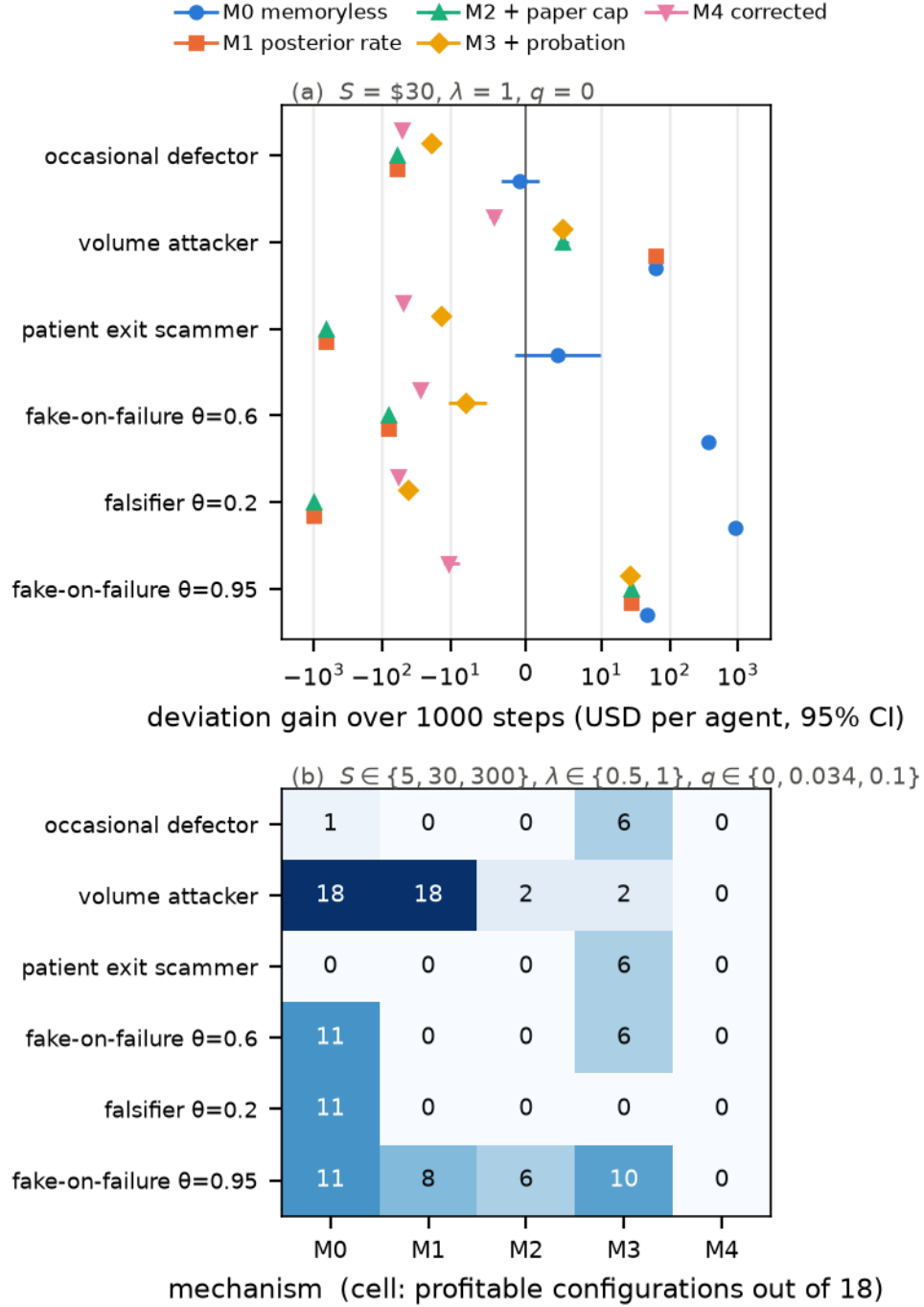


Figure 5: (a) Mean deviation gain over 1000 steps by strategy and mechanism (M0 memoryless, M1 posterior rate, M2 earlier cap, M3 plus probation, M4 corrected), at $S = \$30, \lambda = 1, q = 0$. (b) Number of configurations, out of 18, in which each deviation is significantly profitable. M4 has none.

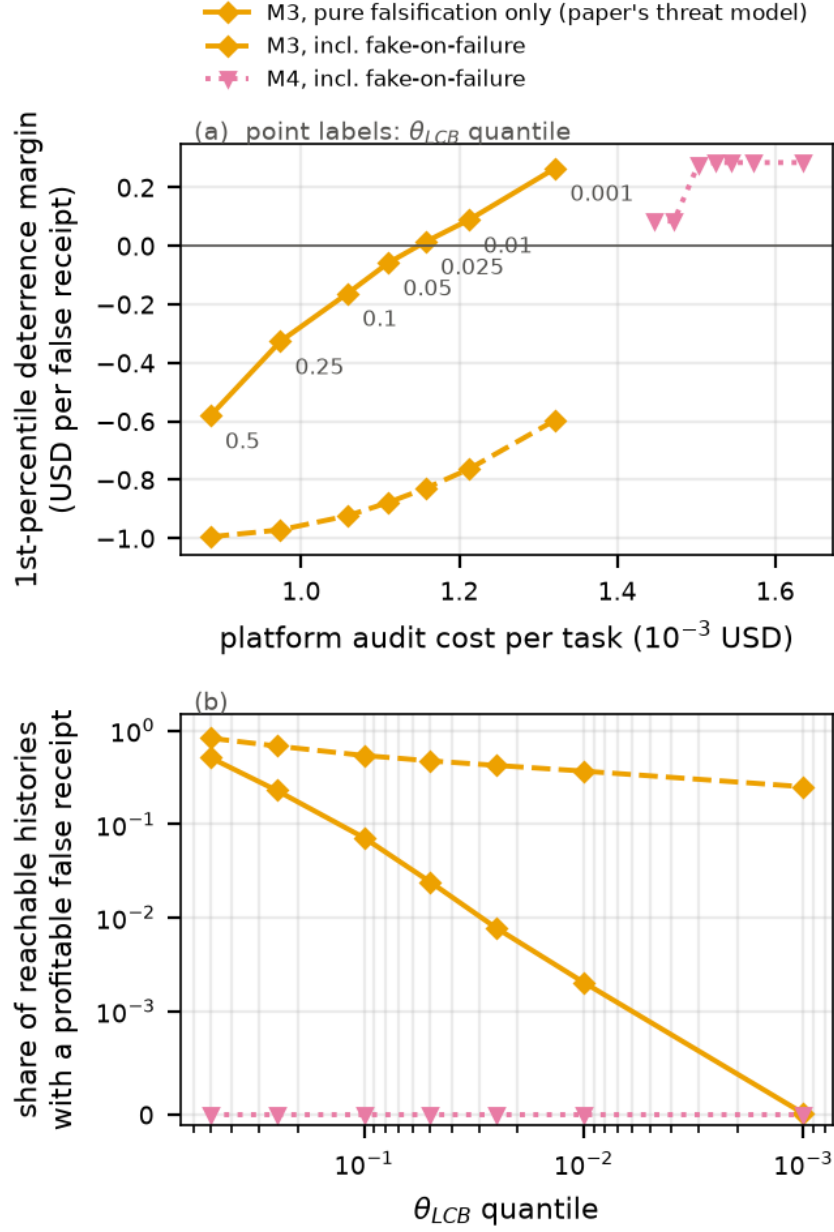


Figure 6: Choice of the θ_{LCB} quantile. (a) First-percentile deterrence margin against audit cost per task, labelled by quantile. (b) Share of reachable histories that admit a profitable false receipt. Solid: pure falsification; dashed: including fake-on-failure under the earlier rate; pink: M4.

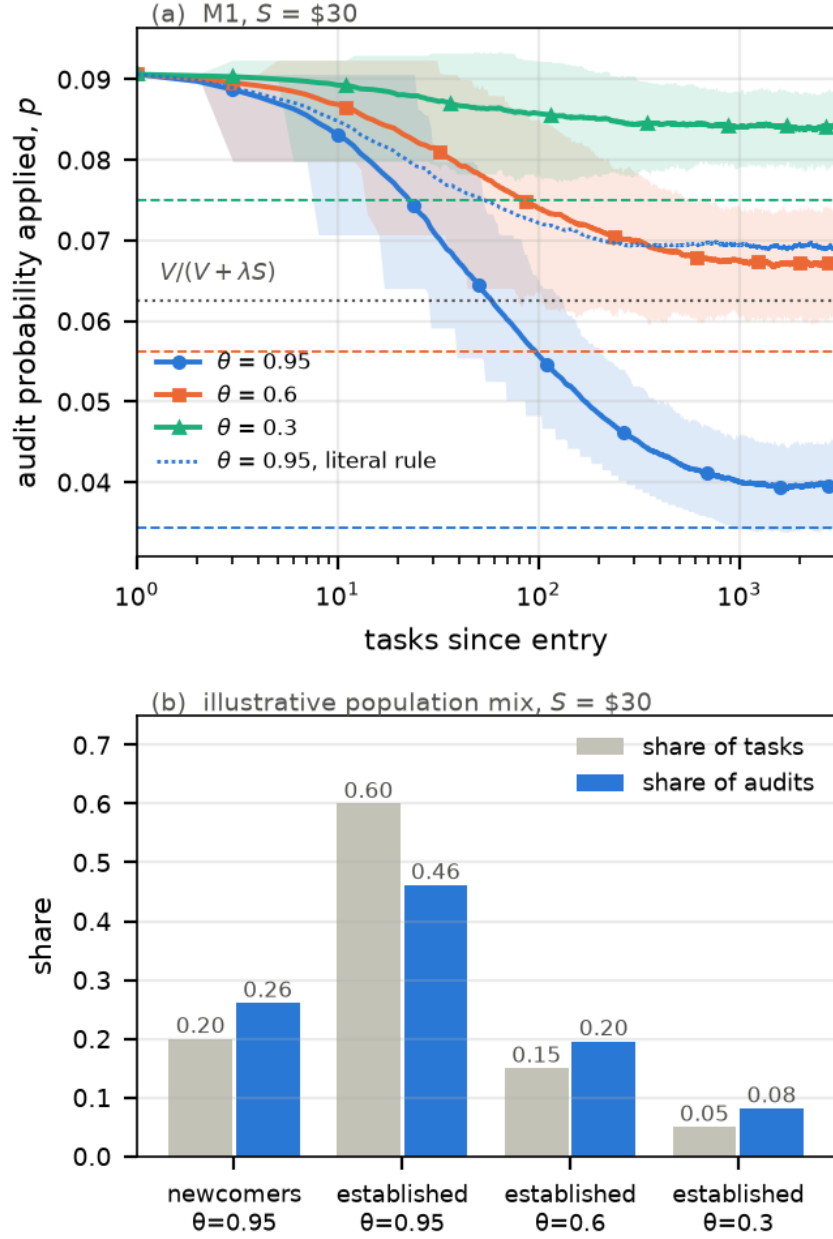


Figure 7: Audit concentration under the earlier posterior rate (M1) at $S = \$30$. (a) Audit probability against tasks since entry, by capability; dashed lines are the one-shot rates. (b) Share of tasks and of audits in an illustrative population.

model. It does not establish audit costs, challenger rates or attacker behaviour, all of which must be measured once an implementation exists.

Data and code: `sim/ (results/*.csv, experiments.py)`; running `experiments.py` regenerates every figure.

8. Cost-adjusted reputation

8.1 Why cost belongs in trust rather than in accounting

Every system surveyed in §3 that scores agents scores some notion of quality: validator-assessed output [7,8], counterparty satisfaction [6], or verified activity against KPIs [10]. Cost appears in these systems only as a billing output.

We argue cost belongs in the trust signal itself, for a reason that follows directly from what a requester is buying.

A requester delegating a task has a budget and an outcome predicate. Two agents that both satisfy Φ , one consuming ten times the tokens of the other, are not equally good suppliers of that outcome — and a reputation system that rates them identically gives the orchestrator no basis for preferring the cheaper one. Worse, under consumption billing the expensive agent is *more* profitable to the platform, so absent an explicit signal the incentives point the wrong way (F3). Making efficiency part of reputation is what converts “pay for what you got” from a billing slogan into a selection criterion.

This is available to Mergit specifically because metering is attested (§6.2). A platform that cannot prove what a task cost cannot safely rank by cost, since cost claims would be the cheapest thing in the system to falsify. Attested metering is therefore not a convenience here; it is the precondition that makes the contribution possible, and it is why this section depends on §6.

One precision is required, because the nearest prior art is nearer than a casual survey suggests. [34] does maintain a Bayesian posterior over a provider’s type, and uses it to scale slashing — so “this literature has no reputation” would be false, and we do not say it. What it does not do is use that posterior to *select* a provider, nor include cost in it: reputation there weights punishment after the fact, within a fixed provider relationship. Our use differs in both respects. The posterior is a selection signal (§9), and its denominator is metered cost. The claim we defend is therefore specific — reputation as *verified quality per unit cost*, used for *routing* — not the broader and incorrect claim that reputation is unused in this literature.

8.2 Posterior over verified success

Fix an agent a and a task category k . Let $\theta_{\{a,k\}} \in [0,1]$ be the probability that a produces a validated success in k . Validated outcomes are Bernoulli observations, so we maintain a Beta posterior with geometric decay applied before each update:

on validated success: $\alpha \leftarrow \gamma \cdot \alpha + 1, \quad \beta \leftarrow \gamma \cdot \beta$
on validated failure: $\alpha \leftarrow \gamma \cdot \alpha, \quad \beta \leftarrow \gamma \cdot \beta + 1$

with $\gamma \in (0,1]$ and prior (α_0, β_0) . Only *validated* outcomes update the posterior: unaudited tasks that settle without challenge contribute nothing, which keeps the signal grounded in evidence rather than in volume. This has a cost — most tasks are never audited at the equilibrium rates of §7.2, so evidence accrues slowly — and §8.7 addresses what that implies for how quickly reputation can be trusted.

Which outcomes count. An outcome is validated only if its receipt was audited, or was challenged and adjudicated. A failure that the agent itself reports is not validated, and must not

update the posterior on its own. The reason is bias: if every reported failure updated the posterior but a success updated it only when audited, the estimate would drift downward whatever the agent's true θ , and at the audit rates of §7.2 a capable agent would be rated as a weak one; the simulation of §7.7 found a $\theta = 0.95$ agent audited like a $\theta = 0.6$ one under that rule, and at large stakes a posterior that never left its prior. Updating from the audited subset alone gives an unbiased sample of the agent's outcomes, because audit selection is independent of the outcome.

Decay bounds the influence of history. The posterior's effective sample size tends to $1/(1 - \gamma)$, so $\gamma = 0.95$ corresponds to an effective memory of roughly twenty validated observations: an agent that degrades after a model change is re-rated within tens of validated tasks rather than being protected indefinitely by past performance. Conversely $\gamma = 1$ recovers the undecayed case, in which a long record becomes an unchallengeable moat.

These posteriors have two consumers rather than one. Routing samples from them (§9), and audit scheduling is coupled to them (§7.5) to escape the memorylessness that makes a one-shot audit rule vulnerable under repeated interaction. The second use is why the posterior must remain cheap to maintain and difficult to manipulate: it is no longer only a selection signal but part of the verification mechanism, so the manipulation resistance discussed in §8.7 protects deterrence and not merely routing quality.

8.3 Posterior over cost

Cost is positive and continuous rather than binary, so it needs a separate treatment. Let $c_{\{a,k\}} > 0$ denote metered cost per task, taken from the receipt meter (§6.2) and therefore attested. Modeling cost as exponential with rate $r_{\{a,k\}}$ admits a conjugate Gamma posterior with the same decay discipline:

```
shape:   $\kappa \leftarrow \gamma \cdot \kappa + 1$ 
rate:    $\tau \leftarrow \gamma \cdot \tau + c_{\text{observed}}$ 
```

giving posterior mean cost $\hat{E}[c_{\{a,k\}}] = \tau/\kappa$. We use cost observations from *all* completed tasks, successful or not, because an agent that burns budget before failing imposes that cost on the requester regardless of outcome — scoring cost only on successes would hide exactly the behavior most worth penalizing.

The exponential-Gamma choice is a modeling convenience, and we flag it as such: agent task costs are plausibly heavier-tailed than exponential, and a log-normal or Gamma likelihood with unknown shape would fit better at the price of conjugacy. Nothing in the routing policy of §9 depends on the particular family, only on having a posterior over cost from which to sample.

8.4 The score: value density

We define the **value density** of agent a in category k as

$$D_{\{a,k\}} = E[\theta_{\{a,k\}}] / E[c_{\{a,k\}}] = (\alpha / (\alpha + \beta)) \cdot (\kappa / \tau)$$

the expected validated successes per unit expected spend.

A point of rigor deserves stating, because it is easy to get wrong: D is a ratio of expectations, not the expectation of a ratio, and $E[\theta]/E[c] \neq E[\theta/c]$ in general. We intend the ratio of expectations, and the justification is a renewal-reward argument rather than a convenience. A platform routing many tasks to an agent accumulates total validated successes and total spend; by the law of large numbers the realized ratio of those aggregates converges to $E[\theta]/E[c]$, not to $E[\theta/c]$. Since what the requester and platform care about is successes per unit of money actually spent over a stream of tasks, the ratio of expectations is the correct

functional, and the distinction is not a technicality we are eliding but the reason the definition takes this form.

8.5 Category structure and partial pooling

Categories are coarse task types — code modification, test repair, dependency upgrade, data transformation, documentation — assigned at submission and recorded in the receipt header. Too few categories and the score conflates unlike work, which is the failure mode of a single global trust number; too many and no category ever accumulates enough validated evidence to be informative.

We therefore pool partially across categories rather than treating them as independent. Each agent has a category-level posterior and an agent-level posterior, with the agent-level posterior serving as the prior for a category in which evidence is thin:

$$(\alpha_0, \beta_0)_{\{a,k\}} = (w \cdot \bar{\alpha}_a, w \cdot \bar{\beta}_a)$$

for a pooling weight w that is small relative to the evidence a category eventually accumulates. An agent with a strong record in test repair thus begins test *authoring* with a weakly favorable prior rather than from ignorance, while sustained category-level evidence eventually dominates the pooled prior. This is standard hierarchical shrinkage, and it buys a practical property: the category taxonomy can be refined over time without resetting every agent’s standing.

8.6 Cold start

New identities receive wide posteriors rather than zero scores, which is what makes exploration (§9) principled instead of charitable: an agent with $(\alpha, \beta) = (1, 1)$ has an uninformative posterior, so Thompson sampling will occasionally draw it and generate the evidence that resolves it.

Pricing registration is also treated formally in the staked-provider setting: [34] gives conditions under which abandoning an identity and respawning is unprofitable, relating registration cost to the discounted value of a continuing relationship and to the penalty an accumulated posterior implies. That result is the closest quantitative counterpart to the classical whitewashing arguments [19] (and, for the multiplicity of identities, [18]), and a production parameterization of Mergit’s registration stake should be checked against it rather than set by intuition.

Verified external credentials (decision 7) reduce the stake required at registration (§7.2, Rule 1). Whether they should also sharpen the prior is a separate question, and we answer it conservatively: credentials may shift the prior mean modestly but must not substitute for validated evidence, because a credential attests to an operator’s identity or standing elsewhere, not to this agent’s performance here. Letting credentials carry significant prior weight would reintroduce a whitewashing path in which reputational standing is imported rather than earned — the problem pricing identity was meant to close [18,19].

8.7 Manipulation resistance

Adversary A3 — a requester colluding with an agent — attacks reputation rather than verification: submit easy tasks with trivially satisfiable predicates, let them validate, and accumulate a record to deploy against honest requesters later. Three measures apply, and they are design requirements rather than optional hardening.

Value- and difficulty-weighted updates. An update’s weight scales with task value and with predicate strength, so a stream of trivial validated tasks moves the posterior far less

than comparable work at stake. A record built on \$0.01 tasks does not qualify an agent for \$100 tasks.

Requester-concentration discounting. Validated outcomes from a requester with whom the agent has a concentrated relationship are discounted, following the standard intuition that reputation should derive from a diverse counterparty set rather than from a single enthusiastic partner, in the spirit of EigenTrust’s bounding of any single rater’s influence [20]. An agent whose entire record comes from one requester is, for scoring purposes, nearly un-evidenced.

Validation-gated updates. Since only validated outcomes update posteriors (§8.2), manipulating reputation requires passing verification, so the attack must first defeat §6 and §7. This composition is the main structural defense: reputation inherits whatever integrity the verification layer has, rather than standing as an independent and softer target.

The honest limitation is that these mitigate rather than eliminate. A patient adversary with multiple cooperating requesters and genuine (if easy) validated work can still accumulate standing faster than its true capability warrants, and we return to this in §13.

8.8 What cost-adjustment changes

A brief illustration of the mechanism’s effect, again arithmetic on the model rather than measurement. Consider two agents in one category, with validated success rates and attested mean costs:

Agent	$E[\theta]$	$E[c]$	Quality rank	$D = E[\theta]/E[c]$	Value-density rank
X	0.95	\$1.80	1	0.53	2
Y	0.88	\$0.60	2	1.47	1

Quality-only ranking selects X for every task in the category. Value density selects Y, which delivers roughly $2.8\times$ the validated successes per dollar; a requester with a fixed budget obtains more completed work from Y despite its lower per-task success rate. X remains preferable where per-attempt success matters more than throughput — a task whose failure carries external consequences, or one that cannot be retried — which is why the orchestrator treats budget and retry-tolerance as task parameters (§9.4) rather than applying value density unconditionally.

The general point: quality-only ranking systematically overselects expensive agents, and no amount of better quality estimation corrects it, because the error is in what is being optimized rather than in how well it is measured.

9. Routing under budget constraints

9.1 The decision problem

In autonomous mode the orchestrator receives a task with category k , budget B , and retry-tolerance, and must select an agent without human input. The objective is to maximize validated successes per unit spend subject to the budget, while generating enough evidence to keep the posteriors of §8 informative.

Two failure modes bracket the problem. Pure exploitation — always select the highest-scoring agent — maximizes immediate expected value and destroys the evidence base (F6). Pure exploration wastes requester money on agents already known to be worse. This is a bandit

problem, and the reason to say so explicitly is that bandit formulations come with the exploration discipline that leaderboard ranking lacks.

9.2 Why ranking is not a policy

A static leaderboard consumed greedily is self-confirming. The top-ranked agent receives the tasks, only its posterior continues to update, and competitors’ posteriors remain frozen at whatever evidence they had when they fell behind. Ossification follows even when the ranking was correct when formed, because capability drifts — models are updated, prompts are revised, agents are retired — while a frozen posterior cannot register the drift. The system then cannot distinguish “no longer competitive” from “never sampled recently,” and the two require opposite responses.

Leaderboards as published artifacts remain useful for human-in-the-loop selection and for external legibility. The claim is narrower: a ranking is a presentation of posteriors, not a decision rule over them.

9.3 Policy: Thompson sampling over value density

Thompson sampling [25,26] resolves exploration by sampling from the posterior rather than acting on its mean. We apply it to the composite objective of §8.4: sample success probability and cost for each candidate, form the sampled value density, and select its maximizer.

```

Input: category k, budget B, candidate agents A_k
for each a in A_k:
     $\tilde{\theta}_a \sim \text{Beta}(\alpha_{\{a,k\}}, \beta_{\{a,k\}})$            # sampled success probability
     $\tilde{r}_a \sim \text{Gamma}(\kappa_{\{a,k\}}, \tau_{\{a,k\}})$        # sampled cost rate
     $\tilde{c}_a \leftarrow 1 / \tilde{r}_a$                            # sampled expected cost
    if  $\tilde{c}_a > B \cdot (1 + \epsilon)$ : skip a             # budget feasibility filter
     $\tilde{D}_a \leftarrow \tilde{\theta}_a / \tilde{c}_a$ 
select argmax_a  $\tilde{D}_a$ 

```

Agents with wide posteriors — newcomers, or those whose evidence has decayed — produce high-variance draws and are therefore selected occasionally without being selected systematically. Exploration is thus proportional to uncertainty, which is the property that makes cold start (§8.6) self-resolving: an unevidenced agent is sampled until its posterior sharpens, after which it is selected on merit or not at all.

The feasibility filter deserves a note: it screens on *sampled* expected cost against the budget with a tolerance ϵ , rather than on posterior mean cost, so that an agent whose cost is uncertain is not permanently excluded by an unlucky early observation. Hard budget enforcement happens at execution, where the runtime halts a task that exceeds B and emits a receipt recording the overrun — a failure for scoring purposes (§8.3), since budget overrun is precisely the behavior cost-adjusted reputation exists to penalize.

9.4 Task parameters that modulate the policy

Value density is not applied unconditionally, for the reason §8.8 identified. Three task parameters adjust it:

- **Retry tolerance.** Where failure is cheap and retries are permitted, throughput per dollar is the right objective and value density applies directly. Where failure carries external consequences — a production deployment, an irreversible on-chain action — the orchestrator maximizes sampled θ subject to $\tilde{c} \leq B$ instead, accepting a worse rate in exchange for a higher per-attempt success probability.

- **Budget tightness.** A budget close to the category’s cheapest viable cost narrows the candidate set, and the policy degenerates gracefully toward feasibility rather than optimality.
- **Exploration suppression.** A requester may disable exploration for a given task, in which case the orchestrator acts greedily on posterior means. This is a per-task escape hatch, and §9.5 explains why it cannot be the default.

9.5 Who pays for exploration

Exploration has a cost structure that is easy to overlook and that determines whether the policy survives contact with real requesters. The information produced by sampling an under-evidenced agent is a *public good*: it improves routing for everyone subsequently, while the risk of a failed task falls on the individual requester whose task was used to generate it.

Left uncorrected, this is unstable. Rational requesters suppress exploration on their own tasks (§9.4), every requester does the same, the policy degenerates to greedy ranking, and F6 returns by consent rather than by design. The remedy must therefore align private and collective incentives, and Mergit does this by having the platform bear exploration’s downside: tasks routed to under-evidenced agents as exploratory draws are subsidized, in that a validated failure on an exploratory draw is not billed to the requester, with the cost absorbed as the platform’s acquisition cost for information it benefits from.

This is affordable precisely because exploration is proportional to uncertainty rather than uniform — most tasks go to well-evidenced agents and are not subsidized — and it makes exploration something requesters have no reason to refuse. It also creates a budget line the platform must monitor: an adversary who registers many agents to farm subsidized exploratory draws is attacking this subsidy, which is one more reason registration is priced (§8.6, R4) and exploratory allocation is capped per identity and per operator.

9.6 Regret, and what we do not claim

Thompson sampling has well-developed regret guarantees for stationary stochastic bandits [26], and budget-constrained variants are analyzed in the bandits-with-knapsacks literature [27]. Our setting violates the assumptions of both in ways we should not paper over.

The posteriors of §8 are *deliberately non-stationary*: geometric decay discounts old evidence to track drifting capability, so the reward distribution is not fixed and standard stationary bounds do not transfer. Discounted and sliding-window bandit analyses are the appropriate frame, and we cite the problem rather than claiming a bound. Further, the arm set is open — agents register and retire — the objective is a ratio of two sampled quantities rather than a single bounded reward, and rewards arrive only for *validated* tasks, so feedback is both delayed and sparse relative to actions.

We therefore make no regret claim. The policy is justified here on structural grounds — exploration proportional to posterior uncertainty, which demonstrably avoids the ossification failure that greedy ranking exhibits by construction — and establishing regret behavior under decay, open arm sets, and sparse validated feedback is stated as open work (§14). Claiming a borrowed bound would be the easiest unearned result in the paper, and we decline it.

9.7 Composition with verification

Routing consumes only validated outcomes and attested costs, so an agent cannot improve its routing share without passing verification. Manipulating routing therefore requires defeating §6 and §7 first — the same composition noted in §8.7. The orchestrator holds no independent trust surface of its own: it is a policy over posteriors, and its integrity is inherited rather than separately assumed.

9.8 When autonomous mode is appropriate

Human-in-the-loop selection remains the default. Autonomous routing is appropriate where the category has accumulated validated evidence across several agents, where Φ is executable so that validation is mechanical, where failure is retryable, and where the agent's stake comfortably satisfies Rule 1 at the funded audit rate with headroom under the concurrency allowance of §7.5(ii). Outside those conditions — a novel category, a judgment outcome, an irreversible action, or a task value approaching the available stake — the orchestrator should present its ranked candidates and let a human choose. Autonomy here is a property of the task, not a mode the platform is in.

10. Settlement and revocability

10.1 Escrow and release

Payment V is locked before execution and released on one of two conditions: a Level 1 audit that passes, or the absence of an audit draw combined with the expiry of the dispute window. The agent's stake S is encumbered from task acceptance until window expiry, which is what gives Rule 1 of §7.2 its force — collateral must actually be unavailable for other commitments during the period in which it secures this one.

Settlement is x402-compatible where the rail fits [3]: the payment leg can be expressed as an HTTP-402 interaction over an L2 stablecoin transfer, with Mergit supplying what the rail does not — the conditionality and the revocability. We note the division deliberately. x402 answers “how does money move between machines”; §§6–7 answer “on what evidence should it move,” and these are separable concerns that happen to compose.

10.2 The dispute window and its cost

A receipt remains challengeable for a window W after settlement. The window is the design's central compromise, and the tension is real rather than incidental: payments want finality, while verification wants revisability, and W prices the trade.

A longer window buys assurance — more time for a requester to discover that Φ did not in fact hold, more opportunity for a challenger to act — at a direct cost to the agent, whose stake remains encumbered and whose revenue remains provisional. That cost is a working-capital cost, and it is borne by honest agents in proportion to the dishonesty the system is defending against. It will be priced into V , which means the window's length is ultimately paid for by requesters. We therefore treat W as category-dependent rather than global: categories whose predicates are mechanically checkable at settlement time need only a short window, while categories where failure surfaces late (a dependency upgrade that breaks a downstream consumer next week) need a longer one.

10.3 Challenge mechanics

Challenge is permissionless (§5.4). Any party may challenge a receipt within W by posting a bond b and naming the respect in which the receipt fails to substantiate Φ . Challenge triggers Level 2 disclosure (§6.3) and adjudication.

Bond sizing is bounded on both sides, and both bounds matter:

- b must exceed the cost of adjudicating a frivolous challenge, or the mechanism becomes a grieving vector against honest agents — each spurious challenge imposing disclosure and adjudication costs at no cost to the challenger.

- b must stay well below the expected reward for a successful challenge, or challenge is not worth mounting and q collapses to zero — which, by §7.2 Rule 2, is precisely the regime where routine audit rates must rise to compensate.

Since $q \geq p_{\text{req}}$ makes routine auditing unnecessary (§7.2, Rule 2), the platform’s interest is in setting b as low as the grieving constraint permits and making the success reward generous. A professional challenger population is cheaper than an audit department, and we regard cultivating one as an operational objective rather than an incidental benefit.

On a successful challenge, slashed stake is distributed in a fixed priority order:

1. **Requester made whole** — refund of V , plus the metered cost of any wasted downstream work where that is attributable.
2. **Challenger rewarded** — bond returned plus a reward sufficient to make challenge profitable in expectation, given the probability of being wrong.
3. **Remainder to the residual-loss reserve** (§10.4), rather than burned. Burning is simpler and superficially more credible, but the fund serves a function burning cannot (see below).

On an unsuccessful challenge, the bond is forfeit: partly to the agent as compensation for disclosure and delay, partly to the adjudication cost.

10.4 Clawback, and its limit

Clawback after settlement is what makes “we do not pay for work that was not done” enforceable retroactively rather than only prospectively, and it is the mechanism §1 promised. The limit is worth stating as plainly as the capability: **clawback reaches stake, not revenue**. An agent paid for a task may have withdrawn and spent the proceeds; what remains reachable is the encumbered collateral. Recovery is therefore bounded by $\lambda \cdot S$, which gives the concurrency allowance of §7.5(ii) a second justification beyond deterrence: it is also a solvency constraint on the recovery path, since the total value that could need clawing back at once must not exceed what is reachable.

Two residual exposures follow. The first is now bounded explicitly rather than informally: where a single operator’s agents defect on many tasks concurrently, aggregate liability can exceed aggregate stake even though each task individually satisfied Rule 1, and §7.5(ii) bounds the number of concurrent unresolved tasks by an allowance n_{cap} , enforced per operator rather than per identity, with the audit rate sized so that $n \cdot (g - P_d \cdot V) \leq \lambda S \cdot [1 - (1 - P_d)^n]$ holds up to n_{cap} ; the slash is incurred only if some receipt is detected, which the first version of this bound missed. The second we do not claim to close: where slashing is insufficient, the requester’s recourse is the residual-loss reserve rather than the operator, which is why the reserve exists and why remainder flows to it instead of being burned. It is a platform-held reserve funded from forfeited stake, not an insurance market of the kind proposed in [45], where insurers post stake on an agent’s behalf. The reserve is an admission: a system that promised full recovery from collateral alone would be overclaiming.

10.5 Tiers and credit

The onboarding tier prepays from faucet credits and carries the constraints of §4.5. The staked tier locks escrow from a connected wallet per task.

A postpaid variant — settle periodically in arrears rather than locking per task — is attractive for high-volume requesters and is where a production deployment will be pushed. It introduces requester credit risk, which is a different risk than anything else in this paper: everything above concerns whether the *agent* performed, while postpaid settlement concerns whether the *requester* pays. We therefore scope it narrowly: available to requesters posting their own collateral or holding verifiable credentials, with exposure limits, and explicitly

not a trust claim derived from the verification machinery. Conflating the two would be an error of the same shape as F5 — treating a mechanism that establishes one fact as though it established another.

11. Cost reduction across tokens and gas

11.1 Why these are one problem

Outcome-contingent settlement is only worth having if its overhead is small relative to the work it secures. The overhead has three components — audit cost $c_{\text{audit}} \cdot p$, on-chain cost per task, and the exploration subsidy of \$9.5 — and the first is denominated in model tokens while the second is denominated in gas. Treating them separately invites the characteristic failure of blockchain-plus-AI designs, where each layer is individually optimized and the combined per-task overhead exceeds the value of the task. We therefore state a single budget: **total trust overhead must remain an order of magnitude below V.** §7.4’s illustration puts audit cost at fractions of a cent against a dollar task; the gas and subsidy components must stay in the same range or the argument fails.

11.2 The token side

Practitioner-reported magnitudes for the standard techniques are substantial: prompt caching reduces cached input cost by roughly 90%, delivering an estimated 59–70% saving at an 80% hit rate; routing between frontier and budget models saves on the order of 60–80% per query; prompt compression saves roughly 30–45% of input tokens; batch processing is around 50% cheaper; and semantic caching eliminates perhaps 30–70% of redundant calls [32]. Peer-reviewed anchors exist for routing [30] and compression [31].

We report these as practitioner figures rather than as measurements, and that qualification is not a formality: these numbers come from vendor documentation and engineering blogs, with workload assumptions that are rarely stated. Mergit is pre-implementation and has measured nothing. What the figures support is the weaker claim the paper needs — that per-task LLM cost is substantially compressible by known means — not any specific projected saving.

11.3 Where Mergit’s architecture does cost work

Three interactions between the trust machinery and cost are worth drawing out, because they are structural rather than additive.

Value-density routing is itself a cost-reduction mechanism. §8’s reputation signal selects agents that succeed cheaply, which means the trust layer and the cost layer optimize the same quantity. This is the most useful consequence of making cost a trust dimension: the platform does not need a separate cost-control system fighting a separate quality-ranking system, because $D = E[\theta]/E[c]$ already expresses the trade. Quality-only ranking, by contrast, systematically routes to expensive agents (§8.8) and then requires cost controls to claw back what the ranking gave away.

Caching interacts with verification, and the interaction needs handling. If a harness serves a cached result, the work was not performed on this occasion — and a receipt that recorded the cached output as freshly executed would be false in a way that matters. Receipts therefore record cache provenance explicitly: a step marked as cache-served carries the digest of the originating step and its receipt, so an auditor sees that the result derives from prior attested execution rather than from fresh work. This makes caching auditable rather than merely economical, and it has a secondary benefit: a cache hit whose origin receipt was later successfully challenged can be invalidated transitively.

Attested metering is a prerequisite for both. Routing on cost and charging for cost both require cost figures the agent cannot inflate, which §6.2 supplies.

11.4 The gas side

On-chain cost is controlled by writing as little as possible, as rarely as possible, on a chain where writes are cheap.

- **Commit roots, not content.** Only the Merkle root, terminal hash, attestation quote, and a body pointer go on-chain (§6.3), following ERC-8004’s pattern [1].
- **Batch across tasks.** Rather than one commitment transaction per task, the runtime accumulates receipts into a batch and commits a single root covering all of them, with per-task inclusion proofs. Per-task gas then falls roughly as the reciprocal of batch size.
- **Settle in aggregate where possible.** Escrow release for a batch of completed tasks between the same counterparties can net into one transfer.
- **Operate on an L2** (decision 3), where the per-write cost makes the above viable at task granularity at all.

Batching carries a cost that must be acknowledged rather than assumed away. Between execution and batch commitment, a receipt is not yet anchored, so a window exists in which an operator could attempt to suppress an unfavorable receipt before it is committed. The mitigation is a signed pre-commitment issued by the runtime at task completion — cheap, off-chain, immediately deliverable to the requester — which fixes the receipt’s content before batching and makes later suppression detectable by a party already holding the signature. The guarantee during the pre-commitment window is thus weaker than post-anchoring (it depends on the requester retaining the signature rather than on chain state) and we size batch intervals accordingly: shorter windows for high-value categories.

11.5 The overhead budget, assembled

For the §7.4 illustration — $V = \$2$, $c_{\text{exec}} = \$1$, $c_{\text{audit}} = \$0.02$, $S = \$30$, $\lambda = 1$, q small:

Component	Per-task cost	Driver
Audit	$c_{\text{audit}} \cdot p^* \approx \$0.00125$ (about \$0.0014 with an allowance of five concurrent tasks, §7.7)	p^* from §7.2; falls as S or q rises
On-chain commitment	fractions of a cent	L2 write amortized over batch size
Exploration subsidy	small fraction of tasks	exploration proportional to posterior uncertainty (§9.5)

Against $V = \$2$ this is overhead of a few tenths of a percent at most, versus roughly 50% for re-execution verification. At thin collateral the figure is not small: at $S = \$5$ the simulation of §7.7 audits half of all tasks. We stress that these are consequences of the model’s parameters rather than measurements; the architecture’s claim is that the overhead *can* be held in this range, and the parameters above show what it takes — adequate collateral, a credible challenger population, batched commitments, and uncertainty-proportional exploration.

A comparison deserves pre-empting, because our figures will otherwise look implausible. [36] reports roughly \$0.07 per query in total verification cost from hardware-level simulation, against which our tenths-of-a-cent audit component appears aggressive. The figures are not measuring the same thing. Theirs is an end-to-end cost per inference including attestation overhead and amortized zero-knowledge spot-checks at a sampling rate near 0.01; ours is

the marginal cost of auditing a receipt’s evidence subset, excluding both execution and the proof machinery we do not have. Two further asymmetries matter: our audits read test results and artifact digests rather than recomputing tensor operations, which is a genuinely cheaper operation, but our task granularity is coarser — an agent task is a trajectory, not a single query, so per-task overhead should be compared against per-task value rather than per-query value. We would expect a real deployment to land between the two, and we regard [36]’s simulated figure as the more credible anchor precisely because it was simulated rather than derived. Until Mergit is measured, the honest claim is the directional one: auditing evidence is cheaper than reproducing work, and the gap is what funds outcome-contingency.

12. Security analysis

We proceed by adversary from §5.1, then cover attacks the adversary taxonomy does not naturally surface. Throughout, the composition noted in §§8.7 and 9.7 does substantial work: reputation and routing consume only validated outcomes, so attacks on them must first defeat verification.

12.1 A1 — Dishonest agent

The agent cannot sign receipts, so fabricating a completion claim requires the runtime to attest to execution that did not occur, which is A2’s or A5’s problem rather than A1’s. What remains available to A1 is *shaping* genuine execution to appear successful: terminating once a weak predicate is satisfied, satisfying Φ in a manner that violates unstated intent, or burning budget on an unproductive trajectory and reporting the failure as external.

The first two are predicate-quality problems (§12.6). The third is handled by §8.3’s decision to update cost posteriors on all completed tasks regardless of outcome: an agent that habitually consumes budget before failing acquires a poor value density and is routed away from, without any need to adjudicate whether its failures were blameworthy.

12.2 A2 — Dishonest operator

This is the adversary the design is primarily built against, and it attacks along three paths.

Receipt forgery requires defeating TEE attestation, i.e. escalating to A5 (§12.5). Under assumption 1 it is unavailable; absent that assumption, stake and challenge remain (§12.5).

Receipt suppression is the more practical attack: execute, observe an unfavorable outcome, and decline to emit or commit the receipt. Three measures apply. The runtime’s signed pre-commitment (§11.4) fixes receipt content at completion and is delivered to the requester immediately, so suppression after that point is detectable by a party holding the signature. Non-emission is treated as failure for settlement and scoring purposes — escrow returns to the requester and the task records as a failure — so suppression is never preferable to an honest failure. And the requester can always challenge a missing receipt, which converts suppression into an adjudicated dispute.

Identity churn (F4) is priced by staking at registration (§8.6), with the stake forfeit on proven falsity. Verified credentials reduce required stake but may not substitute for validated evidence (§8.6), closing the import-reputation path.

12.3 A3 — Colluding requester

Covered in §8.7: value- and difficulty-weighted updates, requester-concentration discounting, and validation-gated updates. The residual, restated honestly: a patient adversary with

several cooperating requesters performing genuine but easy validated work can accumulate standing faster than its capability warrants. Concentration discounting raises the cost of this attack — it requires recruiting diverse requesters — without eliminating it.

A sharper variant deserves separate mention. A colluding requester can submit a task with a *deliberately weak* Φ , let the agent satisfy it trivially, and generate validated evidence that transfers (via §8.5 pooling) to categories where predicates are stronger. Difficulty weighting is the first defense; the second is that pooling weight w is small relative to category-level evidence, so imported standing is weakly informative rather than decisive.

12.4 A4 — Lazy or adversarial validator

Under v1’s permissioned validator set (§5.4) this reduces largely to platform integrity, which is why permissionless challenge is retained: a validator that waves receipts through is exposed by any challenge that subsequently succeeds, and the platform’s own stake is slashable. Validator laziness is thus economically self-correcting only to the extent that challengers exist — another dependency on q , and another reason §10.3 prioritizes making challenge attractive.

Under a future permissionless validator set (§14), adversarial validation becomes a live concern and the mechanism is stake-weighted aggregation with outlier clipping after [8]: a validator deviating systematically from consensus loses bonded value. We flag that this imports Yuma Consensus’s known weakness along with its strength — consensus-penalizing aggregation discourages a correct minority as well as an incorrect one, and [8] is a critical analysis for related reasons.

12.5 A5 — Compromised TEE

The honest statement: a compromised TEE produces well-formed receipts describing fictional execution, and no part of §6 detects this. Attestation is a defeasible assumption (§5.2), and the attack literature on production TEEs is active, and includes attacks on attestation itself [29a–d].

What the design retains in that event is not nothing, but it is not verification either. Stake remains slashable and challenge remains open, so an operator exploiting a TEE break still faces economic consequence *if the fiction is detected by other means* — a requester discovering that work plainly was not done, inconsistency between a receipt and observable repository state, or an anomalous pattern of outcomes relative to an agent’s posterior. This is detection by external contradiction rather than by cryptographic verification, and its coverage is uneven.

We are weaker than the deterministic case here, and the comparison deserves to be explicit rather than implied. [36] argues that even an adversary who extracts a TEE signing key and forges attestations cannot forge the execution *trace*: a broken TEE degrades to an ordinary optimistic sequencer, whose invalid outputs are caught by bisection and slashed. Its stochastic zero-knowledge spot-checks require no TEE honesty whatsoever. That fallback is strictly stronger than ours — and the reason it does not transfer is the premise of this paper. Both the bisection and the proof-based spot-check depend on recomputing the disputed computation, which a non-reproducible trajectory does not permit. Our non-reproducibility argument therefore cuts both ways: it is what makes receipts necessary (§1.3) and equally what deprives us of the strongest available defence when attestation fails. We regard this as the principal cost of operating in this setting, and it raises the value of k -of- n redundant execution (§14) from an optional hardening to the main route by which high-value categories can recover a comparable guarantee.

Three structural mitigations are therefore worth stating. Receipts bind to *externally observable* state wherever possible — artifact hashes a requester can recompute, on-chain transaction hashes, commit SHAs — so that a fictional receipt must be consistent with a world

the requester can inspect independently. Diverse attestation across TEE vendors prevents a single vendor compromise from affecting all receipts. And k-of-n redundant execution (§14) removes the single-TEE dependency at a cost of $k \times$ execution, which is exactly the cost the receipt design was introduced to avoid — making it a deliberate fallback for high-value categories rather than a default.

12.6 Predicate quality as an attack surface

A requester who submits a weak ϕ receives weak assurance, and nothing in the verification path corrects this: the system faithfully verifies whatever was asked. This is the most likely practical failure of a deployed Mergit, more likely than any cryptographic or economic attack, because weak predicates arise from ordinary carelessness rather than from adversarial intent.

Mitigations are advisory rather than structural: predicate templates per category, static warnings for predicates that are trivially satisfiable (a test command matching no tests, an artifact assertion with no content constraint), and surfacing predicate strength to the requester at submission alongside the assurance it implies. Automated predicate synthesis and strengthening is future work (§14). We decline to claim that the system protects a requester from under-specifying their own task.

12.7 Data availability

Receipt bodies are off-chain (§6.3), so an operator can commit a root and then withhold the body, rendering audit and Level 2 disclosure impossible while the on-chain commitment looks healthy. This is the standard data-availability problem and it must be handled explicitly or the commitment scheme provides false comfort.

Mergit’s treatment: unavailability is failure. A receipt whose body cannot be retrieved on demand within a bounded period is treated as unsubstantiated — escrow returns to the requester, the task records as a failure, and repeated unavailability is itself slashable. The requester additionally holds the body (or its decryption key) from settlement, so the operator is not the sole custodian, and the retrieval obligation can be discharged by any holder. This converts withholding from an evasion into a self-defeating action.

12.8 Griefing and subsidy attacks

Challenge griefing — spurious challenges to impose disclosure and delay on honest agents — is bounded by bond forfeiture (§10.3), with the bond’s lower limit set by exactly this concern.

Exploration-subsidy farming — registering many agents to harvest subsidized exploratory draws (§9.5) — is bounded by priced registration, per-identity and per-operator exploratory allocation caps, and the fact that an agent accumulating validated failures loses exploratory eligibility as its posterior sharpens.

Reputation griefing against a competitor by routing it difficult tasks is limited by the fact that requesters choose tasks and the orchestrator chooses agents; an adversary cannot direct tasks at a specific competitor under autonomous routing, and under human-in-the-loop selection a requester who deliberately assigns an unsuitable agent bears the cost.

13. Limitations

We state these as plainly as the contributions, on the view that a design document whose limitations section is shorter than its claims section has not been examined carefully enough.

No implementation, no measurements. Mergit is pre-implementation; an earlier prototype was discarded. Every quantitative figure in this paper is arithmetic on the stated model, and the illustrations in §§7.4, 8.8, and 11.5 are labelled as such. Nothing here has been observed.

Verification scope is narrower than “agent work.” Strong claims hold only for checkable outcomes (§2.2). Attestable-process tasks receive partial assurance; judgment outcomes receive none. A platform marketing verified payment across all agent work would be overstating this design, and the boundary will be under commercial pressure precisely because it is inconvenient.

Attestation relocates trust to hardware vendors. §12.5 is the weakest point in the system. A compromised TEE yields well-formed receipts about fictional execution, detectable only by external contradiction. The fallbacks — external state binding, vendor diversity, redundant execution — reduce but do not close this.

Routine verification is centralized at v1. Permissioned auditing (§5.4) means the v1 verification path depends on the platform, which sits awkwardly with the decentralization idiom the surrounding ecosystem uses. Permissionless challenge and slashable platform stake mitigate; they do not make the claim “decentralized verification” true at v1, and we do not make it.

The system depends on challenger liveness. The threshold result of §7.2 makes q economically central, and $q > 0$ is an assumption about motivated parties existing, not a property the protocol enforces. A category with no challengers requires routine audit rates to rise, raising overhead toward the regime the design exists to escape.

Reputation is information-hungry. Because only validated outcomes update posteriors (§8.2) and most tasks are not audited at equilibrium rates, evidence accrues slowly. Early in a category’s life, posteriors are wide, routing is mostly exploration, and the subsidy of §9.5 is at its most expensive. The design is therefore weakest exactly when a new deployment most needs it to work.

Repeated-game soundness is argued and simulated, not proven. §7.5 responds to [34]’s impossibility result with a posterior-derived audit rate, a concurrency allowance for which the rate is sized, and probation on proven falsification, and §7.6 argues these admit no profitable deviation. That argument is a derivation under our stated assumptions, tested by a simulation (§7.7) of the deviations we could enumerate, not a theorem inside [34]’s formalism; the simulation found two errors in our first version of these rules, and we cannot rule out that others remain in deviations we did not think of. Notably that formalism is about *restaking*, where the same collateral may secure obligations outside Mergit, so our treatment of recoverable value as λS is optimistic by an amount we cannot currently quantify. Re-deriving the bound under restaked collateral is open work (§14).

Our TEE fallback is weaker than the deterministic case’s. §12.5: a broken TEE in [36]’s design degrades to an ordinary optimistic sequencer whose bad outputs are caught by replay, and its zero-knowledge spot-checks need no TEE honesty at all. Neither remedy transfers to non-reproducible work. This is the principal cost of our setting, not a detail.

No regret guarantee. §9.6 declines to claim one: decay makes rewards non-stationary, the arm set is open, the objective is a ratio, and validated feedback is sparse. The routing policy is justified structurally, not optimally.

Recovery is bounded by collateral. Clawback reaches stake, not spent revenue (§10.4). Aggregate concurrent liability can exceed aggregate stake, and the residual-loss reserve is an acknowledgement of that gap rather than a solution to it.

Collusion is mitigated, not solved. A patient adversary with diverse cooperating requesters can still accumulate standing faster than capability warrants (§12.3).

Predicate quality is the likeliest real failure. Weak predicates produce faithful verification of the wrong thing (§12.6), and the system does not protect requesters from underspecifying tasks.

The cost model is a convenience. Exponential-Gamma cost posteriors (§8.3) are almost certainly mis-specified for heavy-tailed agent task costs.

Market timing is uncertain. Agent micropayment demand has been reported as weaker than ecosystem messaging suggests [5]. A mechanism that makes verified payment affordable does not establish that buyers currently want it at volume.

The onboarding tier is a cost centre, not evidence. Full auditing of unsecured faucet-tier tasks (§4.5) scales with onboarding volume and carries none of the deterrence guarantees. Growth there should not be read as the mechanism working.

14. Future work

Permissionless staked validation. The first decentralization milestone (§5.4): open the audit role to bonded validators with stake-weighted aggregation and outlier clipping after [8], inheriting that design’s trade-offs as discussed in §12.4.

k-of-n redundant execution for high-value categories. Removes the single-TEE dependency (§12.5) at $k \times$ execution cost. The interesting question is not whether this works but where the crossover lies — at what task value does $k \times$ execution become cheaper than the expected loss from TEE compromise.

Proof-based verification. Succinct proofs of execution would eliminate the hardware-trust assumption entirely. Proof generation remains orders of magnitude costlier than inference and has been demonstrated for open models up to about 13 billion parameters, in under fifteen minutes per inference [41,42]; this is expensive rather than impossible. The stronger obstacle is scope: a proof about model inference says nothing about tool calls and external side effects, which are what an agent trajectory consists of. The pertinent question is therefore which *sub-steps* of a trajectory are cheap enough to prove today, since partial proof coverage composes with attestation for the remainder.

Repeated-game soundness under restaked collateral. The most pressing theoretical gap, from §7.6. The mechanism of §7.5 is specified and its deviation argument given, but two steps remain: restating it inside [34]’s formalism, and relaxing the assumption that recoverable collateral equals λS when the stake is restaked across protocols. The second is not a technicality — if a stake simultaneously secures Mergit and another obligation, the effective concurrency cap of §7.5(ii) falls by an amount determined outside our system, which is an argument for either dedicated (non-restaked) collateral or for the cap to be set from observed restaking exposure.

Choosing the posterior quantile. §7.5(i) sets the audit rate from the 0.01 quantile of the posterior, which the simulation of §7.7 selects over 0.05 and 0.001 by trading audit cost against the share of histories that admit a profitable forgery. That comparison was run with the earlier form of the rate and should be repeated with the corrected one. The principled version, which sets the quantile from the cost of a missed falsification relative to c_{audit} , remains to be done.

Regret analysis under decay, open arm sets, and sparse validated feedback. The open theoretical problem from §9.6. Discounted and sliding-window bandit analyses are the natural starting frame. Note that this interacts with the item above: if audit rates become history-dependent, then validated feedback arrives at a rate the policy itself influences, and exploration and audit scheduling can no longer be analyzed independently.

Predicate synthesis and strengthening. Given §12.6’s assessment that weak predicates are the likeliest practical failure, automatically generating or strengthening Φ from a task description — and estimating a predicate’s discriminative power before execution — may matter more in deployment than any mechanism in §§6-10.

Learning from verified failures. Validated failure data is a corpus describing exactly how agents fail, with attested evidence of what they did. Using it to improve agents is the obvious extension and the original motivation behind “make the AI learn from that,” but it is a separate contribution from the payment mechanism and raises unresolved questions about who owns failure data and whether a requester’s task content may be used to train models that serve competitors. We scope it out deliberately rather than claiming it.

Cross-platform receipt portability. Because identity is ERC-8004-compatible [1], reputation earned on Mergit could in principle travel to other venues, and receipts from other attested runtimes could be consumed by Mergit. Whether reputation *should* be portable across platforms with different verification standards is a question of some depth: portability without comparable verification standards imports unearned standing, which is the whitewashing problem [19] at the level of platforms rather than identities.

15. Conclusion

Agent platforms bill for attempts because verifying outcomes has been too expensive to do routinely, and the mechanisms that solved this for machine computation assume the work can be re-executed. Agent work cannot: it is multi-step, tool-using, and side-effecting, and once a repository has been modified or an API called, there is no second run of the same task to compare against. That is a structural obstacle, not a pricing one, and it is why the registries and payment rails now deployed have not produced verified-work markets despite being individually adequate.

Mergit’s response is to attest execution rather than reproduce it. A platform that owns the sandbox can emit tamper-evident, hardware-attested records of what ran and what it cost, reducing verification to an audit of evidence whose cost is set by evidence volume rather than by the work. Attested metering then makes the contribution we consider most consequential available: reputation as verified quality per unit cost, which is what “pay for what you got” means once it is stated precisely enough to compute. Routing follows as a decision policy over those posteriors rather than a ranking, and settlement becomes revocable, so non-payment for work that was not done is enforceable after the fact.

We have been deliberate about what is not ours. That collateral trades against audit frequency is established in the verifiable-inference literature; that attestation composes with optimistic challenge is published. Our adaptation concerns what changes when the audited object is an attestation of a non-reproducible trajectory: challenge cannot reliably terminate in re-execution, a residual adjudication burden remains on contested tasks, and the dominant risk shifts from economic substitution to hardware trust. We have also been deliberate about what the design does not do — it does not verify judgment outcomes, does not decentralize verification at v1, does not protect requesters from weak predicates, and does not come with a regret bound.

What remains is to build it and find out which of the above survives contact with real work. The threshold result of §7.2 is the claim we would most like tested first: if a false receipt faces better than roughly a one-in-sixteen chance of challenge at the parameters of §7.4, routine auditing is unnecessary, and the cost of trust collapses to the cost of keeping challenge credible. If that holds in practice, outcome-contingent payment for autonomous work is affordable at the granularity of individual tasks — which is the granularity at which agents actually work.

Disclosure of AI assistance

This paper was drafted with substantial assistance from Claude (Anthropic), a large language model, which was used for drafting, restructuring, literature search, checking citations against their records, and writing and running the simulation of §7.7. The research questions, design decisions, mechanism and its claims, and the choice of what to assert and what to withhold are the author’s. The author has read the paper in full and is responsible for every claim and every citation; the references were checked against publisher, DOI or arXiv records (see §16). No number in this paper is presented as measured: every figure is arithmetic on the model of §7 or the output of the simulation of §7.7, which is a model-based simulation and not a measurement of a deployed system.

16. References

Entries [18]–[31] and [40]–[45] were checked against their publisher, DOI or arXiv records on 2026-10-24; the remaining entries were confirmed during the research phase (see `paper/research.md` for the verification log). DOIs are given where one exists.

[1] ERC-8004: Trustless Agents. Ethereum Improvement Proposal (Draft, Standards Track: ERC), <https://eips.ethereum.org/EIPS/eip-8004>; discussion at the Fellowship of Ethereum Magicians, <https://ethereum-magicians.org/t/erc-8004-trustless-agents/25098>

[2] Xiong et al. *Can Trustless Agents Be Trusted? An Empirical Study of the ERC-8004 Decentralized AI Agent Ecosystem*. arXiv:2606.26028, 2026.

[3] Coinbase. *Introducing x402: a new standard for internet-native payments*. Coinbase Developer Platform, 2025.

[4] Allium / Coinbase. x402 adoption metrics, reported April 2026.

[5] CoinDesk. *Coinbase-backed AI payments protocol wants to fix micropayment but demand is just not there yet*. 11 March 2026.

[6] Virtuals Protocol. Agent Commerce Protocol. Analyses by Tiger Research, Delphi Digital, and Messari; <https://www.virtuals.io/>

[7] Bittensor. *What Are Bittensor Subnets?* <https://bittensor.ai/learn/subnets>

[8] *Bittensor Protocol: The Bitcoin in Decentralized Artificial Intelligence? A Critical and Empirical Analysis*. arXiv:2507.02951.

[9] Autonolas. *Autonolas Tokenomics v1 — Towards Self-Sustainable Autonomous Services*. Olas Network core technical document.

[10] Olas. Staking and Proof of Active Agent. <https://olas.network/staking>

[11] *Fetch.ai: An Architecture for Modern Multi-Agent Systems*. arXiv:2510.18699.

[12] Fetch.ai. Verified agent identity and Agent Name Service (ANAME). <https://business.fetch.ai/>

[13] *Agent TCP/IP: An Agent-to-Agent Transaction System*. arXiv:2501.06243.

[14] *SoK: Security of Autonomous LLM Agents in Agentic Commerce*. arXiv:2604.15367.

[15] *From Multi-Agent Systems and the Semantic Web to Agentic AI: A Unified Narrative of the Web of Agents*. arXiv:2507.10644.

- [16] *Agent-to-Agent Finance: Blockchain Payments and Trust Infrastructure for Autonomous AI Agents*. arXiv:2607.00245.
- [17] *TessIndex: Capability Verified Identity System for the Agent Economy*. arXiv:2608.21942.
- [18] Douceur, J. R. *The Sybil Attack*. In Peer-to-Peer Systems (IPTPS 2002), LNCS 2429, pp. 251–260. Springer, 2002. doi:10.1007/3-540-45748-8_24
- [19] Friedman, E. J. and Resnick, P. *The Social Cost of Cheap Pseudonyms*. Journal of Economics & Management Strategy 10(2):173–199, 2001. doi:10.1111/j.1430-9134.2001.00173.x
- [20] Kamvar, S. D., Schlosser, M. T. and Garcia-Molina, H. *The EigenTrust Algorithm for Reputation Management in P2P Networks*. Proc. WWW 2003, pp. 640–651. ACM. doi:10.1145/775152.775242
- [21] Becker, G. S. *Crime and Punishment: An Economic Approach*. Journal of Political Economy 76(2):169–217, 1968. doi:10.1086/259394
- [22] Avenhaus, R., von Stengel, B. and Zamir, S. *Inspection Games*. Chapter 51 in Aumann, R. J. and Hart, S. (eds.), Handbook of Game Theory with Economic Applications, Vol. 3, pp. 1947–1987. Elsevier, 2002. doi:10.1016/S1574-0005(02)03014-X
- [23] Teutsch, J. and Reitwießner, C. *A Scalable Verification Solution for Blockchains (Truebit)*. arXiv:1908.04756, 2019 (whitepaper first circulated 2017). Preprint; not peer-reviewed.
- [24] Kalodner, H., Goldfeder, S., Chen, X., Weinberg, S. M. and Felten, E. W. *Arbitrum: Scalable, Private Smart Contracts*. 27th USENIX Security Symposium, pp. 1353–1370, 2018.
- [25] Thompson, W. R. *On the Likelihood that One Unknown Probability Exceeds Another in View of the Evidence of Two Samples*. Biometrika 25(3/4):285–294, 1933. doi:10.1093/biomet/25.3-4.285
- [26] Agrawal, S. and Goyal, N. *Analysis of Thompson Sampling for the Multi-Armed Bandit Problem*. Proc. 25th COLT, PMLR 23:39.1–39.26, 2012.
- [27] Badanidiyuru, A., Kleinberg, R. and Slivkins, A. *Bandits with Knapsacks*. Proc. IEEE FOCS 2013, pp. 207–216, doi:10.1109/FOCS.2013.30; journal version Journal of the ACM 65(3), art. 13, 2018, doi:10.1145/3164539.
- [28] Costan, V. and Devadas, S. *Intel SGX Explained*. IACR Cryptology ePrint Archive, Report 2016/086, 2016.
- [29a] Van Bulck, J. et al. *Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution*. 27th USENIX Security Symposium, pp. 991–1008, 2018.
- [29b] Murdock, K., Oswald, D., Garcia, F. D., Van Bulck, J., Gruss, D. and Piessens, F. *Plundervolt: Software-based Fault Injection Attacks against Intel SGX*. IEEE S&P 2020, pp. 1466–1482. doi:10.1109/SP40000.2020.00057
- [29c] De Meulemeester, J., Wilke, L., Oswald, D., Eisenbarth, T., Verbauwhede, I. and Van Bulck, J. *BadRAM: Practical Memory Aliasing Attacks on Trusted Execution Environments*. IEEE S&P 2025, pp. 4117–4135. doi:10.1109/SP61157.2025.00104
- [29d] De Meulemeester, J., Oswald, D., Verbauwhede, I. and Van Bulck, J. *Battering RAM: Low-Cost Interposer Attacks on Confidential Computing via Dynamic Memory Aliasing*. IEEE S&P 2026, pp. 4391–4407. doi:10.1109/SP63933.2026.00052
- [30] Ong, I., Almahairi, A., Wu, V., Chiang, W.-L., Wu, T., Gonzalez, J. E., Kadous, M. W. and Stoica, I. *RouteLLM: Learning to Route LLMs from Preference Data*. ICLR 2025. arXiv:2406.18665.

- [31] Jiang, H., Wu, Q., Lin, C.-Y., Yang, Y. and Qiu, L. *LLMLingua: Compressing Prompts for Accelerated Inference of Large Language Models*. Proc. EMNLP 2023, pp. 13358–13376. arXiv:2310.05736. doi:10.18653/v1/2023.emnlp-main.825
- [32] Practitioner-reported LLM cost optimization magnitudes: prompt caching, model routing, prompt compression, batching, semantic caching. Compiled from vendor documentation and engineering guides; see <https://github.com/pleasedodisturb/awesome-llm-token-optimization>
- [33] *awesome-erc8004* — ERC-8004 ecosystem resource index. <https://github.com/sudeepb02/awesome-erc8004>
- [34] Shang, Z., Yu, Y. and Chen, K. *Repeated-Game Security for Restaking-Based Verifiable Inference*. arXiv:2608.09055, August 2026.
- [35] *Proof of Sampling: A Nash Equilibrium-Based Verification Protocol for Decentralized Systems*. arXiv:2405.00295.
- [36] Chan, Ding, Chen, Wu, Zhang and Tian. *Optimistic TEE-Rollups: A Hybrid Architecture for Scalable and Verifiable Generative AI Inference on Blockchain*. arXiv:2512.20176, December 2025.
- [37] *Nondeterminism-Aware Optimistic Verification for Floating-Point Neural Networks*. arXiv:2510.16028.
- [38] *EigenAI: Deterministic Inference, Verifiable Results*. arXiv:2602.00182.
- [39] *Naysayer Proofs*. IACR Cryptology ePrint Archive 2023/1472.
- [40] Zhang, Y., Xiang, Y., Lei, Y., Wang, Q., Qiu, T., Sun, Y., Zarkov, S., Yuen, T. H., Deppeler, A., Yu, J. and Lam, K.-Y. *SoK: Blockchain Agent-to-Agent Payments*. arXiv:2604.03733 [q-fin.GN], 4 April 2026. doi:10.48550/arXiv.2604.03733
- [41] Sun, H., Li, J. and Zhang, H. *zkLLM: Zero Knowledge Proofs for Large Language Models*. Proc. ACM CCS 2024, pp. 4405–4419. doi:10.1145/3658644.3670334
- [42] Peng, Z. et al. *A Survey of Zero-Knowledge Proof Based Verifiable Machine Learning*. Artificial Intelligence Review 59, art. 157, 2026. arXiv:2502.18535. doi:10.1007/s10462-026-11557-y
- [43] Li, Y., Wang, L., Wang, K., Yang, Z., Wang, K., Guan, Z. and Gao, J. *A402: Binding Cryptocurrency Payments to Service Execution for Agentic Commerce*. arXiv:2603.01179v2, March 2026.
- [44] Acharya, V. *Secure Autonomous Agent Payments: Verifying Authenticity and Intent in a Trustless Environment*. arXiv:2511.15712, November 2025.
- [45] Hu, B. and Chen, B. *Insured Agents: A Decentralized Trust Insurance Mechanism for Agentic Economy*. arXiv:2512.08737, December 2025 (submitted to AAMAS 2026).