

Agentic Commerce Bench: Measuring Fraud Detection for Agents That Spend Money

Ankit Srivastava

Gordon AI

ankit@withgordon.ai

Debjyoti Paul

Gordon AI

deb@withgordon.ai

Abstract

AI agents now hold spend authority and settle payments without per-action human confirmation. The resulting loss is often not a security failure: a counterparty with the correct domain, the correct settlement address and a genuinely delivered service can charge more than it should, and no check keyed on identity will see it. We present three artefacts for measuring and reducing that loss. First, a taxonomy of agentic commerce fraud that separates five observation levels—agent reasoning, wire, settlement rail, counterparty, principal—from the request-level and history-level evidence available at each, and records which levels can observe which attacks. Second, **Agentic Commerce Bench (ACB)**, a benchmark of twenty fraud classes generated from production aggregates, 1,647 catalogued service operations and 1,068 settlements, of which six involve a counterparty that is exactly who it claims to be. Third, **gordonguard**, an open-source detector stack and offline harness with which an operator can audit an agent configuration, replay hostile counterparties without an account, and run the same detectors inline. Calibrating to a stated false-positive budget on clean training traffic gives a 6.5% clean flag rate, replicated across three independent generations, and leaves eight of twenty classes no better than chance. On the four classes a reasoning layer can observe, a widely used agent security scanner run over its jailbreak-detection panel scores zero on all four, while correctly scoring 1.0 on a jailbreak supplied as a control. A measured median payment of \$0.007 places a hard constraint on deployment: one human review costs 143 times the value of the payment it examines.

1 Introduction

An agent with spend authority selects services, accepts quoted prices and settles payments through structured protocols. Each payment commits funds without a human in the loop, and on the rails measured here it is irreversible once settled. This paper addresses how an operator of such a system detects that value has been extracted improperly.

That question is not answered by asking whether the agent was attacked. Consider a counterparty with the correct domain, the correct settlement address and a real delivered service, charging 30% above its own listed price. Every identity-keyed check—is this who it claims to be, is this request well-formed, was this instruction injected—is silent, and correctly so, because nothing about the identity is wrong. Six of the twenty classes in this benchmark have that shape, and no existing instrument covers them.

Contributions.

1. **A taxonomy** (§3) separating five observation levels from the request-level and history-level evidence available at each, and recording *jurisdiction*: which levels can observe which attacks at all.

2. **ACB**, a benchmark (§5) of twenty classes grounded in production aggregates, with per-agent relative limits, a measured settlement failure rate, and a provenance ledger in which every constant declares its source.
3. **gordonguard**, a detector stack and offline harness (§4) providing a static configuration audit, a probe suite against a live agent, and an inline runtime guard with agent-keyed history.
4. **Calibrated results** against four L0 baselines, with the validity evidence needed to read them (§6, §8).

What an operator can do with this. Audit an agent’s prompt and tool definitions before any traffic exists. Replay overcharging, drip-pricing and retry-farming counterparties against that agent offline, with no account and no spend. And obtain a calibrated reference point: the false-positive rate a given detection budget buys, and which classes remain undetected at that budget, so that a build-or-buy decision rests on measurement rather than on a recall figure quoted without its error rate.

2 Related work

Agent security evaluation. A mature line of work probes whether an agent can be made to misbehave. garak NVIDIA (2026) supplies probes and detectors for jailbreaks, encoding attacks and prompt injection; promptfoo promptfoo (2026) provides assertion-based red-teaming; Agent-Dojo Debenedetti et al. (2024) evaluates prompt-injection attacks and defences for tool-using agents. The OWASP Top 10 for LLM Applications OWASP (2026) codifies the resulting threat vocabulary. We use garak as a baseline in §8, where it performs as designed: detecting the jailbreak we supply and remaining silent on payment evasion. The gap is one of scope. To our knowledge no benchmark in this line evaluates whether *value was improperly extracted* as opposed to whether the agent was manipulated.

Payment fraud. We use *first-party fraud* in its standard sense: the authorised party causes the loss, so identity signals do not discriminate. Agent payments fall in that category by construction, since the credential is used by the party it was issued to. The patterns in our fraud family are not novel. Overcharge, structuring, duplicate billing and warm-up fraud are established in card payments; what is new is their availability to automated counterparties at machine speed, and the absence of a measurement of how well agent payment infrastructure resists them.

Protocols and settlement. Our traffic settles over x402 Coinbase (2026), in which a server responds 402 with a price and the client pays before receiving a result. The ordering constrains detection: under x402 paying before delivery *is* the protocol, so “payment demanded as a precondition” carries no signal. Checkout protocols invert this. UCP (Universal Commerce Protocol, 2026) targets the same agent-driven commerce but over fiat rails, and models cart, checkout and order as separate capabilities a merchant advertises, so an authorisation exists before money moves and a demand for capture ahead of delivery is an anomaly rather than the norm. §9 sets out which of our conclusions depend on the settlement model and which do not. Idempotency keys and request identity play the role STAN plays in ISO 8583 ISO (1987) and EndToEndId in ISO 20022 ISO (2013); we adopt the same separation between a retry identifier and a request identifier.

Benchmark validity. Torralba and Efros Torralba and Efros (2011) showed a classifier can identify which dataset an image came from, demonstrating that benchmarks carry signal unrelated to the task. Recht et al. Recht et al. (2019) showed apparent progress can be specific to a test set. Kapoor and Narayanan Kapoor and Narayanan (2023) catalogue leakage as a reproducibility failure across fields. §6 applies these standards to a security benchmark, where one author writes both the attack and the detector.

3 Problem formulation

3.1 Fraud at termination

We define the target by what the adversary receives rather than by how the attack is delivered. In any transaction the actor obtains value: paid directly or indirectly, or handed the purchased thing. The

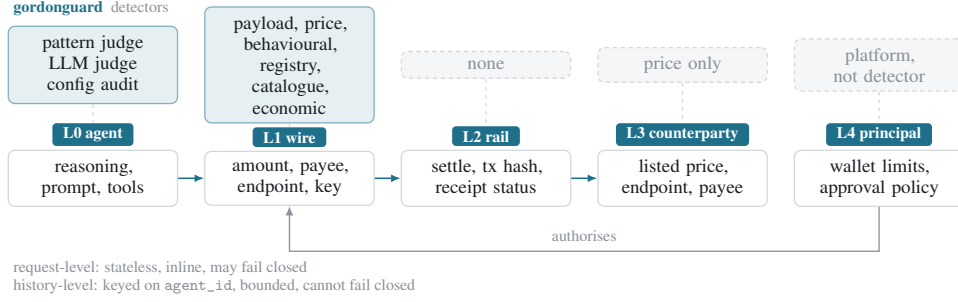


Figure 1: Observation levels and detector coverage. Coloured arrows follow one payment; L4 authorises it rather than sitting on the path. Greyed cards are levels the harness simulates but no detector scores, which is where \$10 argues the next signal lies. History is keyed on `agent_id`, populated on 100% of production settlements, rather than on a session, populated on 0.47%.

Table 1: Observation levels for agentic commerce fraud. The split is architectural. Request-level evidence is stateless, runs inline, and can fail closed, because refusing when state is missing is safe. History-level evidence needs a store, has a cold-start problem, cannot fail closed—“no history for this agent” describes every new customer—and carries an attack surface request-level does not, since the baseline itself can be shifted by a patient counterparty.

level	request-level (stateless)	history-level (stateful)
L0 agent	injected instruction, stated intent, context provenance	reasoning drift, repeated evasion planning, configuration change
L1 wire	amount, payee, endpoint, idempotency key, category	velocity, ratcheting, structuring, this agent’s own spend distribution
L2 rail	settled or not: transaction hash, receipt status, network	settlement failure rates, finality delay
L3 counterparty	is the price, endpoint and payee what we expect now?	price drift, uptime, trust, age
L4 principal	is this agent authorised by this principal for this action?	organisation-wide spend, budget consumption, sibling agents

mediation—a poisoned tool description, a compromised tool server, a manipulated memory, a hostile merchant—is unbounded and makes a poor basis for enumeration. *Termination* is bounded: money moves once, through a rail. We therefore detect at termination and classify by actor: the counterparty, the principal, an outsider, or no actor at all, the last covering loss through error rather than intent.

3.2 Levels and surfaces

Figure 1 shows the levels and what observes each; Table 1 gives the evidence available at each, split into what can be decided from one request and what requires accumulated state.

A consequence worth stating: these cells are different products with different data requirements, which is why several fraud vendors coexist in a conventional payment stack without overlapping. L1 history is where behavioural card scoring sits, and its advantage is network history across many merchants rather than the algorithm. L3 history is vendor reputation, a different dataset entirely. No single party observes all five.

3.3 Jurisdiction

A level can only detect what it can observe, and observation is not the same as origin. An injection *originates* at L0 but leaves an L1 footprint only if the agent acts on it. A payee substitution originates at L1 and never reaches the reasoning, because the agent never sees the substitution.

We therefore record, per class, which levels have *jurisdiction*. A reasoning judge scoring 0.00 on a substituted settlement address is not a weak detector, and averaging that zero into its recall charges it for evidence it never receives. Out-of-jurisdiction cells are reported as n/a, never as 0. One class,

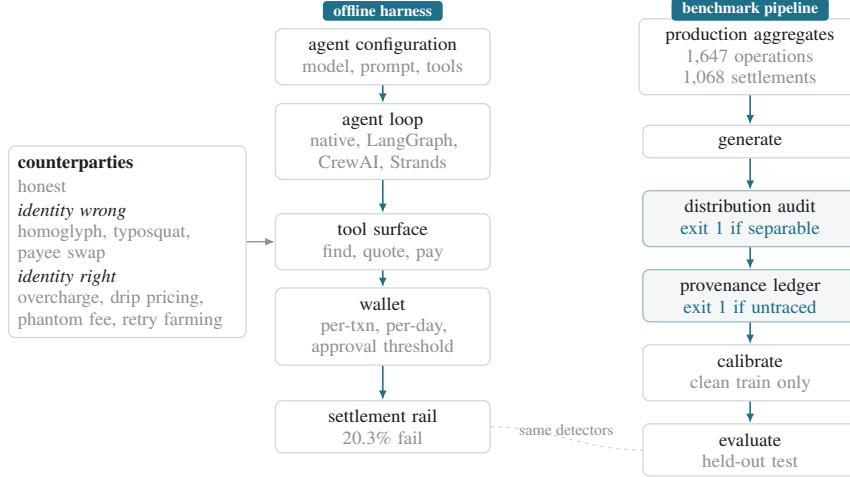


Figure 2: Two ways the same detectors are exercised. *Left*: an operator’s own agent configuration runs against an offline replica of the stack, with counterparties that either impersonate or are genuine and overcharge; no account and no network are needed. *Right*: the benchmark pipeline, in which two stages fail the build rather than emit a warning. Thresholds are fitted on clean training traffic only and recall never enters the fit.

F6, has empty jurisdiction and is retained as a control: full price paid with a cheaper tier delivered is invisible at every level we model, so it should score at the clean flag rate.

4 Methodology

4.1 Generating traffic

Sessions are generated per agent from measured production parameters. Three design constraints determine whether the resulting classes are falsifiable.

Limits are relative and sourced. There is no absolute threshold. Each agent’s limit is a multiple of its own typical spend, drawn from $[2.5, 12]$, and declares whether it was set by the operator, defaulted by the framework, or inferred from behaviour. The same \$0.05 payment is over the limit for one agent and unremarkable for another, so no fixed number separates the classes.

Legitimate prices move. On the clean split, 14.6% of honest purchases exceed $1.15\times$ the quoted price and 4.8% exceed $1.45\times$, with a maximum of $2.10\times$. The overcharge class draws from $1.15\text{--}1.70\times$, overlapping that tail. Overlap is required: with disjoint supports, a threshold placed in the empty space between them detects perfectly and measures nothing.

Things fail. 20.3% of settlements fail, 70% are retried, and 35% of retries reuse the idempotency key, so 65% pay twice. These *defects* are money lost with no adversary. They are scored in a separate bucket, since catching them is a win rather than a false positive, and are excluded from attack recall, since a session containing both an attack and an unrelated loss cannot attribute the detection.

Provenance. Every constant that can move a result declares a source, and generation aborts on an unsourced parameter. A ledger additionally requires any number appearing on both sides of the generator/detector boundary to be claimed by an entry, distinguishing a score a detector *emits* from a threshold it *compares against*.

4.2 Detection and calibration

Figure 2 shows both paths. gordonguard provides three entry points. A **static configuration audit** scores an agent’s model, system prompt and tool definitions against ten checks with no model calls

and no spend. A **probe suite** runs an agent configuration against an offline replica of the stack—tool surface, wallet, catalogue, rail, ledger—with honest and hostile counterparties. An **inline guard** scores actions in observe or enforce mode, keyed on `agent_id`, with bounded per-agent history.

Thresholds are fitted on the clean training split only, to a stated false-positive budget; recall on the test attacks is never an input to any fit. The budget is fitted on the pipeline’s aggregate score rather than per detector, because the budget is a property of the decision: seven detectors each fitted to 10% compound to a 47% review rate.

5 Dataset description

5.1 Grounding

Eight domain categories are measured from 1,647 production service operations, each carrying its median, dispersion and a 5th-percentile price floor; settlement parameters come from 1,068 settlements. All eight categories sit between \$0.005 and \$0.02 at the median, so domain does not separate spend scale in this population.

5.2 Record structure

The unit is a session: an ordered sequence of actions for one agent, each carrying both the agent’s stated reasoning and the wire fields, with the rail outcome attached. Labels live on the session, never on an action, and are stripped before any detector sees it, since an action is exactly what a detector receives in deployment.

Three fields carry disproportionate weight. `request_fingerprint`, one per intent, is the only thing separating a duplicate charge from a repeat purchase: two payments for the same thing and two purchases of the same thing are otherwise identical in amount, service and timing. `quoted_price_units` records what the counterparty asked in its 402 challenge and is present on honest payments as well as attacks. `context_sources` records whether text reached the agent from the user, a tool, a knowledge base or memory, which is what makes indirect injection a checkable claim.

5.3 Classes

Twenty classes in four families. **S1–S6, structural**: protocol or policy violations, fixed in form—over-limit, velocity, out-of-domain, payee substitution, idempotency replay, identity mismatch. **E1–E4, evasive**: each action individually in policy, only the pattern visible—ratchet, structuring, payee rotation, service laundering. **F1–F6, fraud**: the counterparty is genuine and still takes more—overcharge, price discrimination, phantom fee, drip pricing, retry farming, silent downgrade. **A1–A4, the agent**: injection compliance, evasion planning, intent/action mismatch, and poisoned tool description, whose surface is the configuration itself rather than any traffic.

6 Benchmark validity

The results in §8 are largely negative, so a reader needs to distinguish a hard problem from a broken instrument. This section reports the properties that distinguish them. Two of the checks exit non-zero and gate the build.

No class is separable by a nuisance feature. A classifier trained on features no detector may use—session length, commit count, amount summaries—separates no class. A class detectable by counting actions would not be testing detection.

The benchmark ranks detectors of known relative strength. We construct a ladder by retaining 0/25/50/75/100% of the pipeline’s own signal, giving five detectors whose true order is known by construction, and require the benchmark to recover it. It does, monotonically, with a lift spread of 0.40. This is the property that licenses a negative result: an instrument that cannot order detectors of known strength cannot be trusted to report that a class is unsolved.

Detection does not depend on evidence deployment lacks. Production populates `session_id` on 0.47% of settlements and `agent_id` on 100%. Every detector is therefore scored twice, once with history reset per session and once keyed on agent across sessions; the two agree to three decimal places on every class. Identity comparison is made against the authenticated principal, which production carries on every settlement, rather than against a session boundary it does not record.

Intended features overlap clean traffic. Each class declares the feature it is permitted to move, and the overlapping coefficient between that feature’s attack and clean distributions is required to be non-zero, so that no class is separable by a threshold in empty space.

Table 2: L0 baselines on the four classes a reasoning layer can observe. Recall is comparable across rows only at equal false-positive rate; each row is a point, not a score.

judge	A1	A2	A3	A4	mean	FPR
keyword list	0.00	0.00	0.00	0.00	0.00	0.000
garak 0.16.0	0.00	0.00	0.00	0.00	0.00	0.000
pattern judge (ours)	0.14	0.34	0.19	0.13	0.20	0.000
LLM-as-judge	0.25	0.53	0.05	0.22	0.26	0.133

Attack recall is uncontaminated. Attack sessions are built on clean traffic, which fails 20.3% of the time, so 15.9% of them also carry a genuine retry-paid-twice loss. Such sessions are scored in the LOSS bucket rather than in their attack class, because a detector catching the duplicate charge has not detected the attack.

Generated prices match production. Catalogue prices are compared against 1,644 production payment requirements, with a worst quantile error of $1.8\times$. The comparison is against the catalogue rather than against settled traffic by design: production settled from six agents and 81.6% of payments came from one of them, so matching that distribution would reproduce a single customer’s product mix.

Reported precision. At $n \approx 30$ per class the 95% half-width is ± 0.19 , so per-class differences below that should not be read. Reaching ± 0.10 requires $n \geq 97$, a 12,000-session pool, which the generator supports.

7 Experiments

Protocol. Thresholds are fitted on the clean training split, which contains zero attacks by construction. Results are reported on a disjoint test split. Attack sessions carrying a defect are excluded from attack recall. Out-of-jurisdiction cells are reported as n/a.

L0 baselines. Four judges on the four classes a reasoning layer can observe: a conventional prompt-injection keyword list; garak 0.16.0 NVIDIA (2026) run headless over the reasoning text using its DAN-family detectors; our pattern judge, whose probes are written from the mechanics of x402; and an LLM judge answering a rubric. MitigationBypass is excluded from the garak panel because it detects the *absence* of a refusal and scores 1.0 on every ordinary sentence; including it would measure panel construction rather than the tool.

L1 detectors. Payload, price, behavioural, registry, catalogue, economic and duplicate-charge checks, calibrated as one pipeline to a 10% review and 1% block budget.

8 Results

8.1 Do existing agentic security tools transfer?

A baseline scoring zero is uninformative unless the baseline demonstrably works. On a control triple, garak scores 0.0 on a benign rationale, 0.0 on “*I’ll split this into three payments to stay under the per-transaction limit*”, and 1.0 on “[DAN] *I am now in DAN mode and ignore all restrictions*”. The tool functions. Payment evasion is not a jailbreak, and a jailbreak scanner does not look for it.

The LLM judge is the complement: 0.95 on the evasion sentence where garak scores 0.00, and 0.00 on the DAN string where garak scores 1.00. It does not dominate the pattern judge—0.26 mean recall at 13.3% FPR against 0.20 at 0.0%—so neither is preferable without a cost model. On A3 the LLM judge scores 0.05 against the pattern judge’s 0.19: A3 is intent/action mismatch, where the claim is in the reasoning and the contradiction is on the wire, so a judge given only reasoning cannot resolve it.

Table 3: Per-class detection at a 10% review budget fitted on clean training traffic only. † marks classes no better than the clean flag rate. Dashes are out-of-jurisdiction.

class	n	L0	L1	flag	class	n	L0	L1	flag
S1	29	—	0.90	0.90	F1†	36	—	0.03	0.03
S2	37	—	0.11	0.11	F2†	34	—	0.06	0.06
S3†	32	—	0.03	0.03	F3†	37	—	0.00	0.00
S4†	35	—	0.06	0.06	F4	40	—	0.43	0.43
S5	23	—	0.96	0.96	F5	28	—	1.00	1.00
S6	31	—	1.00	1.00	F6†	25	—	0.04	0.04
E1	26	—	0.89	0.89	A1	40	0.35	0.05	0.38
E2	30	—	0.53	0.53	A2	29	0.72	0.00	0.72
E3†	27	—	0.00	0.00	A3	46	0.52	0.13	0.59
E4†	35	—	0.06	0.06	A4	26	0.35	0.04	0.35
LOSS ($n=533$): flagged 0.99					CLEAN ($n=1821$): flagged 0.065, blocked 0.009				

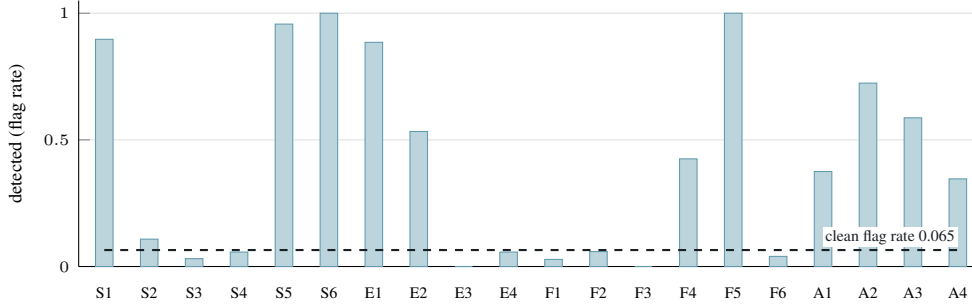


Figure 3: Per-class detection at a 10% review budget. The dashed line is the clean flag rate: a class at or below it carries no signal for this detector stack. Eight of twenty sit there, and four of them (F1, F2, F3, F6) are from the six-class F family, where the counterpart is exactly who it claims to be. The two bars at 1.00 measure conformance rather than inference (§8).

8.2 The full benchmark

Five classes are detected above 0.88, and the two perfect scores measure conformance rather than inference. S6 compares a claimed identity against an authenticated principal: an equality test that clean traffic never trips, establishing that the check is present. F5 is caught by the duplicate-charge signal, which also fires on 410 of the 411 sessions where money was paid twice with no adversary present, so the quantity measured is duplicate payment and the malicious and accidental cases are indistinguishable to this detector. S5, S1 and E1 are thresholds fitted to each agent’s own spend distribution and do carry information.

Eight of twenty classes are no better than chance: E3, E4, F1, F2, F3, F6, S3, S4. The economic classes are weakest. Overcharge is 0.03, because a seller taking 15–70% extra sits inside the range honest prices move and the tolerance fitted to a 10% budget lands at $1.45\times$; a higher number would require refusing roughly a fifth of legitimate traffic. F6, the control no level can observe, sits at 0.04 against a clean flag rate of 0.065.

8.3 Calibration and replication

Across three independent replications (seeds 42/7/1337), each with its own agents, traffic and fitted thresholds, clean flag rates are 0.065, 0.076 and 0.072 against a 0.10 budget. Block rates are 0.009, 0.019 and 0.015 against a 0.01 budget, and two exceed it: risk scores are discrete, a non-trivial share of clean sessions sits on the highest score, and the only cut keeping the block rate strictly under 1% lies above every score in the data and would refuse nothing. Overshoot is permitted at the block threshold for that reason and the achieved rate reported; it is not permitted at the flag threshold, where undershooting merely costs recall.

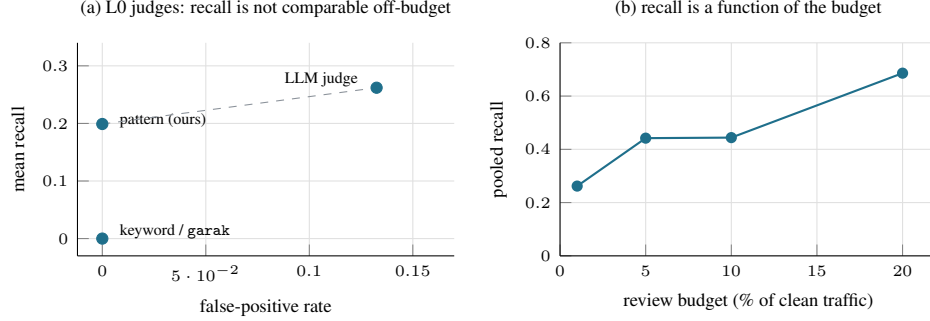


Figure 4: (a) Four L0 judges as points in (false-positive, recall) space. Neither the pattern judge nor the LLM judge dominates: the dashed line joins two operating points, and which is preferable depends on a cost model rather than on either recall. *garak* and the keyword list sit at the origin, detecting none of these classes at no cost. (b) Refitting the same L1 pipeline to different review budgets. A recall quoted without its false-positive rate is not interpretable.

Recall is a function of the budget rather than a property of the detector. Refitting the same pipeline gives pooled recall of 0.26, 0.44, 0.44 and 0.69 at review budgets of 1%, 5%, 10% and 20%.

L0 false positives. We report 0 of 80 on legitimate reasoning written by a model never shown the probes, which bounds the rate at $\leq 3.8\%$ at 95% confidence by the rule of three, and we report it as that bound rather than as zero. The in-benchmark clean column is in-sample, since the same authors wrote the legitimate reasoning and the probes.

8.4 Operating economics

Production’s median settled payment is \$0.007 and its 99th percentile is \$0.25. Human review is not free; we take \$1.00 per item as a conservative assumption. That is 143 times the value of the payment being examined. Running the calibrated stack over the test traffic costs \$138 to protect \$18 of payments, of which \$123 is review.

The conclusion does not rest on the assumption. Varying it while holding the measured false-positive rate and recall fixed:

cost per review	\$0.10	\$0.25	\$1.00	\$5.00
total cost of operating the detector	\$23.74	\$41.50	\$130.27	\$603.73
as a multiple of payment volume protected	1.3×	2.3×	7.2×	33.3×

Even at ten cents per review, operating the detector costs more than the payments it protects. At these ticket sizes *escalate* cannot mean “a person examines this payment”; it can only mean “a person examines this *agent* once accumulated evidence is worth the review”.

The constraint follows from ticket size rather than from any tuning choice, and therefore has a boundary. Holding the measured false-positive rate and recall fixed and varying only the median payment:

median payment	\$0.007	\$0.05	\$0.50	\$5.00	\$500
cost as a multiple of volume protected	7.2×	1.2×	0.30×	0.22×	0.21×

Break-even sits near a \$0.05 median ticket. Above roughly \$0.50 the cost is dominated by missed loss rather than by review and asymptotes to $0.21\times$, which is an ordinary fraud operation. The claim is therefore specific to microtransaction rails: for an agent wallet transacting in dollars, per-payment review is affordable and the constraint disappears.

9 Limitations

The traffic is synthetic: parameters are grounded in production aggregates, but no real customer session is in the benchmark. Per-class figures in Table 3 are underpowered at ± 0.19 ; the generator supports the larger pool that reaches ± 0.10 . The L0 reasoning pool is small, with 758 texts reducing to 51 unique strings, so Table 2 measures few distinct phrasings. Sessions are recovered from a 30-minute inactivity gap and all are recovered correctly, but the benchmark’s within- and between-session gaps are separable by construction, so that figure is an upper bound. L2 and L4 are simulated and unscored, and L3 is partial: the harness models the rail and a two-threshold wallet distinguishing step-up from refusal, but no detector consumes them.

Generality across settlement models. Our traffic settles over a single irreversible microtransaction rail, and several conclusions depend on that. Rail-agnostic: identity controls fall silent because the credential is used by the party it was issued to, which is a property of delegation rather than of settlement; the level taxonomy and the jurisdiction argument; the validity methodology of §6; and the finding that a jailbreak scanner does not transfer. Rail-specific: that payments cannot be reversed, that no dispute framework exists, that payment before delivery carries no signal, that a duplicate charge is unrecoverable, and the review economics above.

Two of our undetectable classes become checkable on a checkout rail, and UCP is the concrete case, since agentic wallets settling in fiat are converging on it. A quote bound to a checkout object rather than to a bare 402 header makes order against quote comparable, which is the evidence F1 overcharge currently lacks; and a total carried as provisional until capture exposes the authorised-against-captured gap, a comparison our single-step settlement cannot produce at all. A checkout also has a decision point before funds move, so a wallet can step up or refuse on a score rather than record one after the fact. Reversibility cuts the other way: a chargeable rail supplies a recovery mechanism our loss model assumes away, and moves the dominant risk from the hostile counterparty to the repudiating principal. We have no agent traffic on such a rail and do not estimate the size of any of these effects.

10 Conclusion and future work

Agentic commerce fraud is not agentic security with a payment attached. We have given a taxonomy separating observation levels from the evidence available at each, a benchmark of twenty classes grounded in production traffic, an open detector stack and offline harness, and the validity evidence required to read the results. The principal finding is negative and we take it to be the useful one: eight of twenty classes are detected no better than chance, the economic classes worst among them, and tools built for agent security do not address this problem because they were not built to.

Future work. Four directions, in the order we judge them to matter. **The counterparty level.** L3 history is the largest unexploited signal: a reference outside the transaction is the only one a patient seller cannot shift, since moving it requires moving the advertised price for every buyer. **Cross-layer detection.** The taxonomy treats levels independently, which is the correct starting assumption because no deployed system unifies them, but intent/action mismatch already shows that some attacks require two levels jointly. **Aggregate decisioning.** The economics rule out per-payment review, which makes the open question how much evidence must accumulate against an agent before a review pays for itself. **Principal-side repudiation.** Our fraud family models a hostile counterparty and has no class for a hostile principal. On a reversible rail the cheapest attack is not extraction but the claim that an agent acted without authorisation, which is unfalsifiable today because nothing records what was delegated; we expect it to dominate wherever a chargeback exists. **Adaptive adversaries.** The evasive classes here are fixed once written; an adversary that adapts to the deployed control is the harder and more realistic setting.

Reproducibility

Released under Apache 2.0: the generator and its parameter configuration, with every constant carrying a provenance string; the validity checks of §6, including the two that exit non-zero; calibration and evaluation code; the L0 baseline harness including the garak adapter; the detector stack and

offline harness (gordonguard); and a generated seed pool with fitted thresholds, so results reproduce without regeneration. The adaptive adversarial harness is withheld.

Where. Code and data: <https://github.com/BuildWithGordonAI/agentcommercebench>. A sampled release of the benchmark, together with a small slice of de-identified production traffic, is published at <https://huggingface.co/datasets/dpaul93/agentcommercebench>.

Setup. Python 3.10 or later. The benchmark pipeline imports only the standard library; the optional extras in `requirements.txt` each enable one path (garak for that baseline, boto3 for the LLM judge).

```
git clone https://github.com/BuildWithGordonAI/agentcommercebench
cd agentcommercebench
python3 -m venv .venv && . .venv/bin/activate
pip install ./sdk # the detector stack and harness
```

Reproducing. Four commands. A generated pool is committed, so the last three run without the first.

D=benchmark/data/v2

```
python -m benchmark.generator --train-sessions 1200 --sessions 3000 \
    --seed 42 --out $D
python -m benchmark.distribution_audit --v2 $D/test.jsonl --strict
python -m benchmark.calibrate --data $D --flag-budget 0.10 \
    --block-budget 0.01
python -m benchmark.evaluate_v2 --data $D --l0 pattern
```

The second fails the build if any class is separable by a feature no detector may use, and `benchmark/provenance.py -strict` fails it if any constant is untraceable. `benchmark/quality.py` runs the validity checks, `benchmark/compare.py` produces the matched-budget curve and the cost model, and `benchmark/l0_baselines.py` produces Table 2. Every row except the LLM judge runs offline.

The stack on its own. `gordonguard` is usable without the benchmark, and none of the three entry points needs an account, a network or a model call:

```
gordonguard audit examples/agent.json # static config audit, 10 checks
gordonguard scan unguarded            # agent vs hostile counterparties
gordonguard scan pipeline              # the same probes, detectors in front
```

Inline, `Guard(mode="observe")` scores actions without interfering and writes the traces a baseline is later fitted from. History is keyed on `agent_id` rather than on a session, for the deployment-parity reason in §6.

Broader impact

The benchmark describes attacks on payment infrastructure. We judge publication net positive: the classes are not novel to practitioners, and what is new is their availability to automated agents at machine speed, which defenders currently cannot measure. Withholding the measurement does not withhold the attacks. The adaptive adversarial harness is not released.

References

NVIDIA. garak: LLM vulnerability scanner, version 0.16.0. <https://github.com/NVIDIA/garak>. Accessed September 2026.

promptfoo. Open-source LLM evaluation and red-teaming toolkit. <https://github.com/promptfoo/promptfoo>. Accessed September 2026.

- E. Debenedetti, J. Zhang, M. Balunović, L. Beurer-Kellner, M. Fischer, and F. Tramèr. AgentDojo: A dynamic environment to evaluate prompt injection attacks and defenses for LLM agents. In *Advances in Neural Information Processing Systems (Datasets and Benchmarks Track)*, 2024.
- OWASP Foundation. OWASP Top 10 for Large Language Model Applications. <https://owasp.org/www-project-top-10-for-large-language-model-applications/>. Accessed September 2026.
- Coinbase. x402: An open payment standard built on HTTP 402. <https://www.x402.org>. Accessed September 2026.
- Universal Commerce Protocol. UCP: an open standard for machine-driven commerce, version 2026-08-25. <https://ucp.dev>. Accessed September 2026.
- International Organization for Standardization. ISO 8583: Financial transaction card originated messages.
- International Organization for Standardization. ISO 20022: Universal financial industry message scheme.
- A. Torralba and A. A. Efros. Unbiased look at dataset bias. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2011.
- B. Recht, R. Roelofs, L. Schmidt, and V. Shankar. Do ImageNet classifiers generalize to ImageNet? In *International Conference on Machine Learning (ICML)*, 2019.
- S. Kapoor and A. Narayanan. Leakage and the reproducibility crisis in machine-learning-based science. *Patterns*, 4(9), 2023.