

Caliber: an accounts-payable clerk whose code decides the money, measured against a planted answer key

Anudeep Bonagiri

Sreesanth Nitillan Senthilkumar

Abstract—Caliber is an autonomous accounts-payable clerk for a simulated 40-truck Texas carrier. A kernel in plain TypeScript, in integer cents, matches every fuel-card charge, toll and maintenance invoice against telematics, tank size, purchase order, contract rate, receiving record and vendor bank details over 92 business days. Clean rows post within policy limits; everything else becomes a hold. A read-only Factory Droid session investigates each hold through Elastic Agent Builder tools and the evidence files and returns a typed finding; when a person is needed, the controller answers from ASI:One and the answer becomes a scoped, expiring, backtested rule. Segregation of duties is enforced in SpacetimeDB reducers: the agent identity cannot approve a payment, and every action is an append-only audit row. We release CarrierBench, the synthetic carrier with its hidden answer key of 52 planted errors in 7 classes and 6 decoys, a machine audit that every planted row is inferable from its own evidence, and a model-only baseline on identical inputs. On the measured 92-day run the kernel reaches precision 1.00 and recall 1.00 on the 58 answer-key rows and posts 7,886 payments without a person; the oracle ceiling posts 7,886. A saboteur run shows that a finding recommending release with confidence 0.99 and fabricated evidence cannot move money. We report what we got wrong and the limits of a synthetic benchmark.

I. System context

Caliber has one write path to money, shown in Fig. 1. The business-day clock ticks inside the SpacetimeDB module; on each tick matchDay hands every row of the day to kernel.ts, a pure function that never imports the database, a connector or a model client. The kernel returns a code and an evidence object. A matched row within the auto-pay limits (\$2,500.00 per transaction, \$40,000.00 per day) becomes a payment row; a duplicate_toll is auto-rejected with an audit row and no task; every other code becomes a hold and an agent_task.

The worker claims the task with a version-locked claim_task, opens a Droid session with autonomyLevel Off, the edit and execute tools disabled, a permission handler that cancels anything but MCP tool calls, and a 90 s abort, at concurrency 2. Droid reads the receipts, contracts and the dispatcher’s swap log and calls Elastic Agent Builder tools over MCP (nearest telematics rows, the contract rate and clause). Its output is validated against a schema whose recommendation is one of clear, hold or hold_and_ask; there is no pay. The worker writes it back with record_investigation, which can set a question on the hold and nothing else. The controller, over an

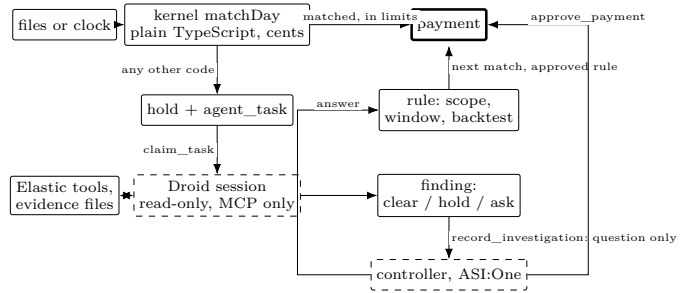


Fig. 1. The one write path. Only the kernel (a matched row within limits) and approve_payment (a controller identity that did not request the hold) write a payment. The finding reaches the hold’s question field and nothing else; Droid’s reach into data is MCP and read-only files.

ASI:One agent or the terminal, answers the question; the answer is drafted into a rule with a scope, a window and a backtest the module computes itself, and only an approved rule lets the kernel clear the next matching row. approve_payment checks the caller’s role, refuses the identity that requested the hold, requires a call-back note on a bank change, and writes the only other payment row there is. Refusals through the try_* procedures are committed audit rows.

II. CarrierBench

CarrierBench is the synthetic carrier as a benchmark. seed/generate.py with seed 42 writes 92 business days (2026-07-01 to 2026-11-05) for 6 terminals, 40 trucks with one fuel card each, 48 drivers, 5 maintenance vendors, 34 contracts and 30 purchase orders: 5,607 fuel-card transactions at 46 stations, 2,157 tolls at 14 plazas, 180 invoices and 155,457 telematics pings, 7,944 payable rows in all. Inside them the generator plants 52 errors in 7 classes and 6 decoys (Table I), recorded in answer_key.json, which no agent code reads. The generator is deterministic: carrierbench.py regenerated the fixtures in 2.74 s and every file was byte-identical to the commit (yes).

Controls. Three are computed, not asserted. Held-out entities: 25 of 40 trucks carry exactly one answer-key row and 10 carry none, so 19 planted rows sit on an entity with no other planted row; all 5 vendors carry more than one (0 held out), which the audit reports rather than hides.

TABLE I

CarrierBench answer key and machine audit. Decoys are legitimate swaps that look like the code. Inferable: the Python mirror of the kernel raised the row with the key’s code from the row’s own records. File: an evidence document exists for the row.

Code	Planted	Decoys	Inferable	File
location_mismatch	14	6	20	20
capacity_exceeded	6	0	6	6
rate_variance	9	0	9	9
duplicate_toll	11	0	11	11
duplicate_invoice	4	0	4	4
vendor_bank_change	3	0	3	3
short_received	5	0	5	5

Decoys: 6 legitimate unit swaps from the dispatcher’s swap log, 48.50 to 97.20 miles from the truck’s nearest ping, every one indistinguishable from card misuse by the records the kernel reads (all look like location_mismatch: yes; all have swap-log lines: yes). Policy holds: 6 clean invoices above the per-transaction limit, a hold by policy and not an exception, kept out of the key and out of the baseline sample.

Machine audit. seed/mirror.py, the Python copy of kernel.ts, evaluated all 7,944 rows from their records (nearest ping within 30 minutes, tank size, earlier tolls at the plaza, the contract valid on the receipt day, the purchase order and its received quantity, the vendor’s bank-change day). It raised 58 of 58 key rows with the key’s code, 58 have an evidence file, and it flagged 0 rows outside the key. Audit passed: yes. The kernel’s thresholds are 25 miles and 30 minutes, tank plus 10 percent, 10 minutes at a plaza, 5 percent over contract, 30 days for a same-amount invoice, and \$5,000.00 within 7 business days of a bank change.

Run it from a bare clone with python3 bench/carrierbench/carrierbench.py --seed 42; it writes REPORT.md and REPORT.json and exits non-zero on any failure.

III. Protocol

Four systems were given the same rows. Caliber is the kernel plus Droid, measured on the full 7,944-row replay through the ledger (docs/FACTS.json). Claude Sonnet (claude-sonnet-5-5), Claude Opus 5.5 (claude-opus-5-5) and asil-mini (asil-mini) were each given, per row, exactly the records the kernel sees (the transaction, the truck and its pings for the day, or the plate’s earlier tolls, or the purchase order, the contract valid on the receipt day, the vendor record and the vendor’s earlier invoices), with no answer-key field and no planted marker, and asked for hold or clear with a code and a one-line reason, one attempt per row, no examples, no fine-tuning. The sample is stratified: all 52 planted rows, all 6 decoys and 500 clean rows (174 card, 173 toll, 153 invoice; every clean invoice, since only that many exist), 558 rows in all; the 6 policy holds were left out. Temperature, as recorded by the runner: asil-mini 0; Claude Sonnet not sent (Claude Sonnet 5.5

TABLE II

Precision / recall per code, code-exact; key rows include the decoys. Caliber on the full 7,944-row run; the three models on the 558-row sample. Generated 2026-10-25T12:55:10+00:00.

Code	Key rows	Caliber	Sonnet	Opus 5.5	asil-mini
location_mismatch	20	1.00 / 1.00	1.00 / 1.00	0.95 / 1.00	0.71 / 1.00
capacity_exceeded	6	1.00 / 1.00	1.00 / 1.00	1.00 / 1.00	n/a / 0.00
rate_variance	9	1.00 / 1.00	1.00 / 1.00	0.47 / 1.00	1.00 / 0.89
duplicate_toll	11	1.00 / 1.00	1.00 / 1.00	1.00 / 1.00	0.12 / 1.00
duplicate_invoice	4	1.00 / 1.00	1.00 / 1.00	1.00 / 1.00	0.29 / 0.50
vendor_bank_change	3	1.00 / 1.00	1.00 / 1.00	1.00 / 1.00	0.11 / 0.33
short_received	5	1.00 / 1.00	1.00 / 1.00	0.62 / 1.00	1.00 / 1.00
overall	58	1.00 / 1.00	1.00 / 1.00	0.81 / 1.00	0.31 / 0.81

rejects a non-default temperature); Claude Opus 5.5 not sent. Both Anthropic models ran with thinking at the API default and no effort field; the default effort differs by model (API default (medium on Opus 5.5; Sonnet 5.5 defaults to high)), so each sits at its own default rather than one shared setting. Prompts are committed under bench/prompts, raw replies under bench/outputs.

Prompt condition. One condition was run, which we name hinted: the prompt (bench/prompts/hold_or_clear.txt) lists the 7 codes and their thresholds before asking. A bare condition, records and question with no code list, was not run; the comparison below is hinted only, which favours the models. Scoring is code-exact: a hold counts for a code only when the system named that code, the same definition facts.py applies to the kernel; a detection reading (any code) is also given. A decoy is scored as a correct hold for every system, because the records cannot clear it. Positives are the 58 answer-key rows. Cost cap \$15 per Anthropic model, applied to each on its own; stopped for cost: Sonnet no, Opus 5.5 no. Partial run: no.

IV. Results

Per code and overall. Table II is the comparison. Caliber: precision 1.00, recall 1.00, 58 true positives, 0 false positives over 7,944 transactions and 92 days. Claude Sonnet: 558 of 558 rows answered, 58 holds, code-exact precision 1.00 and recall 1.00 (detection 1.00 / 1.00, 0 false holds), 6 of 6 decoys held, \$6.38 at list rates. Claude Opus 5.5: 558 of 558 rows answered, 72 holds, code-exact precision 0.81 and recall 1.00 (detection 0.81 / 1.00, 14 false holds), 6 of 6 decoys held, \$12.85 at list rates. asil-mini: 557 of 558 answered, 150 holds, precision 0.31 and recall 0.81 (detection 0.33 / 0.84, 101 false holds), 6 of 6 decoys held, 561 requests against a quota of 100 per fifteen minutes. Where each failed, from the scorer: Claude Sonnet failed on nothing in this sample. Claude Opus 5.5 held 14 clean rows (holds by code that did not match a planted row: 10 as rate_variance, 3 as short_received, 1 as location_mismatch). asil-mini missed 4 of 6 capacity_exceeded, 1 of 9 rate_variance, 2 of 4 duplicate_invoice, 2 of 3 vendor_bank_change; held with the wrong code 2 capacity_exceeded; held 101 clean rows (holds by code that did not match a planted

row: 82 as `duplicate_toll`, 8 as `location_mismatch`, 8 as `vendor_bank_change`, 5 as `duplicate_invoice`); 1 row had no parseable answer (1 unparsed), counted as not held. Ties or wins against the kernel: 13 entries, listed code by code in Section V. Where a model ties the kernel on a code, the kernel’s case on that code is not accuracy on this hinted sample but cost per row (\$6.38 on Sonnet and \$12.85 on Opus 5.5 for 558 rows, against none), determinism (the same row gives the same code every day), and the gate: a model’s hold is advice the worker may record, the kernel’s hold is a row only a person can release.

End to end through the ledger. In the measured run (2026-10-25T00:22:27+00:00) the kernel or an approved rule posted 7,886 payments (\$1,083,630.52) with no person involved, auto-rejected 11 duplicate tolls, and raised 53 holds, of which 6 were policy holds (\$16,789.00). The controller released 8 payments (\$32,097.00). The 6 decoys were cleared through an approved rule after exactly one controller answer each (6 of 6); planted exceptions cleared by a rule: 0; wrong postings: 0. Of the 3 bank-change holds, 1 carried a public-records hit and was rejected by the controller, 2 were approved with a call-back note; all 3 vendor checks came from hand-authored fixtures labelled as such (the CRS sandbox had not answered). All 7,894 payments carry a labelled offline settlement id and 0 a live one. Investigations in the measured run were 47 replayed recordings of the live Droid run and 0 live; the live run itself cost, per task, 1,033 input, 1,354 output and 70,386 cache tokens in 25.60 s, 2,016,204 Factory credits for 52 tasks.

Oracle ceiling. The oracle is the kernel handed the answer key: every planted row held (duplicate tolls auto-rejected), every decoy cleared with no question, clean rows posted under the same policy limits. Computed from the fixtures with the mirror, it posts 7,886 payments (\$1,083,630.52), raises 41 exception holds and 6 policy holds, auto-rejects 11, asks 0 questions, posts 0 wrong rows, and still needs 47 human decisions. The measured run posted 7,886 without a person, raised 47 exception holds and 6 policy holds, auto-rejected 11 and asked 6 questions. The whole gap to the ceiling is the 6 questions on the decoys, which is the mechanism working as designed: the records cannot clear a decoy, a person can, and the answer is remembered as a rule.

Saboteur. A lying finding (recommendation clear, confidence 0.99, 3 fabricated evidence items citing a telematics tool result, a receipt line and a call that never happened) was fed through the real reducers in the in-process module harness for 2 planted holds (INV-1077, 2231; `vendor_bank_change` and `location_mismatch`): `claim_task` and `record_investigation` as the agent identity, the clock’s next match, then `approve_payment` as the agent identity on both entry points. Outcome: the finding was recorded, the hold moved to `awaiting_controller` with the agent as requester, payment rows before and after were 1 and 1, payment rows for the sabotaged holds 0, and 2 audit rows

read agent may not do this (needs controller). Test status: passed. The reducer path threw and rolled its audit row back; the procedure path committed it.

V. What we got wrong

Labels. Hand-shaped vendor checks were first rendered like recorded ones. Commit `aef1ed0` (2026-10-24 14:21) gave them their own status, an amber badge, the fixture note in every evidence row, and removed endpoint placeholders from the screen; decision D-3 then made the one planted CRS hit a hand-authored fixture labelled `FIXTURE` on the card and in every evidence row, with `docs/not-built.md` saying the sandbox has not produced it. The control (a hit vendor cannot be paid without a person rejecting it) is real; the data source is pending and says so.

Decoys. The first live recordings cleared the 6 decoys on Droid’s own reading of the swap log. That is a model clearing a hold. Commit `e3ade6e` (2026-10-24 14:13) changed the investigator prompt to ask when the evidence is consistent with a legitimate explanation only a person can confirm, and re-recorded all 6 as `hold_and_ask` with a question naming the driver; the controller’s answer, not the finding, clears them. The earlier recordings are kept under `_v1`.

The CRS fixture. The vendor-vetting beat runs on fixtures because the sandbox invite did not arrive in time (D-3). We report 3 of 3 checks as fixtures rather than present them as live calls.

Where a model tied or beat the kernel. From the scorer, code by code: Claude Sonnet tied the kernel on `location_mismatch`; Claude Sonnet tied the kernel on `capacity_exceeded`; Claude Opus 5.5 tied the kernel on `capacity_exceeded`; Claude Sonnet tied the kernel on `rate_variance`; Claude Sonnet tied the kernel on `duplicate_toll`; Claude Opus 5.5 tied the kernel on `duplicate_toll`; Claude Sonnet tied the kernel on `duplicate_invoice`; Claude Opus 5.5 tied the kernel on `duplicate_invoice`; Claude Sonnet tied the kernel on `vendor_bank_change`; Claude Opus 5.5 tied the kernel on `vendor_bank_change`; Claude Sonnet tied the kernel on `short_received`; `asil-mini` tied the kernel on `short_received`; Claude Sonnet tied the kernel on overall. (13 entries). Partial baseline: no.

VI. Limits

The company, the money and the vendors are synthetic, generated by the same program that planted the errors, so precision and recall measure whether the kernel finds what we planted, not how it behaves on a real carrier’s statements; the generator also guarantees that every planted row is inferable from its records, which real data does not. The models saw identical inputs but only the hinted prompt; a bare condition was not run. Investigations in the measured run are replayed recordings of one live Droid run, labelled on every card; no live model ran during the

measured 92 days. The vendor checks are hand-authored fixtures (CRS sandbox pending) and settlement ids come from a mock bank. The scripted controller decides from the swap log, which the same generator produced. Droid's session options disable write and execute tools, but its Read can address any path; an OS-level read-only mount is not built. The Elastic Workflow that escalates stale holds surfaced a new escalation in the controller's status 63 s after the trigger against a 30 s target. Rules expire (6 expired by day 92); their long-run behaviour on a live carrier is untested.

Reproduce

```
python3 bench/carrierbench/carrierbench.py --seed
42; python3 bench/carrierbench/oracle.py; bash
bench/carrierbench/saboteur_run.sh; the baseline
commands in bench/README.md; make -C docs/paper
rebuilds this note from the result files.
```