Ledger-Mediated Orchestration: Vendor-Independent Autonomous Software Delivery over a Pool of Coding Agents

Adam Matthew Steinberger

Abstract—A single autonomous coding session is not autonomous software delivery: sessions lose context across crashes, exhaust one vendor's capacity mid-task, and carry no structure for the human decisions delivery legally and practically requires. We present a ledger-mediated orchestration model and the family of components it is built from. All delivery state is a function of an append-only ledger, $\text{state}(t) = f(\text{ledger}_{\leq t})$, which makes engine handoff a scheduling event rather than a loss of state. Work items are claimed from a PostgreSQL queue with FOR UPDATE SKIP LOCKED under renewable leases; a handoff between engines is admitted only by a pure, model-free no-loss predicate; and delivery proceeds through a six-phase state machine whose four human gates each require an explicit recorded verdict. Beneath the orchestrator, five session runners share one bounded, never-blocking core that never gives a credit balance a clock and lets a capacity verdict outrank a completion claim. Above it, an exact-head release calculus binds every automated verdict to the revision it evaluated and terminates within a bounded number of repairs. Beside it, a deterministic retrieval engine and a fail-closed bootstrap layer apply the same append-before-act discipline. Finally, we report a measured regularity from the project's own tracked records: on one machine, with the model and deadline fixed, successful throughput stayed between 0.99 and 2.00 generations per minute while offered concurrency rose sixteenfold. The regularity yields a zero-shortfall time-to-completion as a testable prediction. We state what would falsify it and what it does not cover, and we argue from the same records that the scarce inputs were governance and correct judgment, not production.

Artifacts. This paper is typeset from docs/paper.md and published as PDF[1] and HTML[2]. The complete documentation is published as a book: PDF[3], EPUB[4] and print HTML[5]. Every empirical figure in the section on production rate is recomputed from tracked sources by scripts/paper_evidence.py.

I. Introduction

Let an engine be an autonomous coding session runner over one vendor's model, and let $E = \{e_1, \ldots, e_m\}$ be a pool of such engines with independent failure and capacity behavior. The delivery problem is to carry a specification from human intent to deployed software using E, under three constraints that single-session tooling violates: (i) no vendor session may be the source of truth; (ii) human decisions must occur at defined points, not wherever a session happens to stall; (iii) exhaustion of one vendor's capacity must not lose work.

The system that answers these constraints is one repository holding a family of packages: the orchestrator vibey; five session runners, claudeloop, codexloop, cursorloop, agyloop and the local qwenloop; vibey-gh, which owns provenance, merging and release; vibey-skills, a retrieval engine over a skill library; and vibey-bootstrap, a bootstrap layer for cloud workloads. Each package once carried its own paper. This paper consolidates them. Its contributions are:

- the ledger invariant, the no-loss handoff gate, the queue semantics and the gated six-phase machine, with their soundness arguments;
- a session-runner core shared by five engines, whose capacity taxonomy never gives a credit balance a clock and whose completion rule a capacity verdict outranks;
- the exact-head release calculus, with a termination bound and a recorded production counterexample;
- deterministic, fail-closed retrieval and bootstrap components built on the same append-before-act discipline;
- a measured production-rate regularity, its modulators, the time-to-completion prediction it enables, and the observations that would falsify it.

II. The ledger invariant

Invariant 1 (Write-ahead intent). Every decision, finding, and handoff is a row in an append-only ledger before it takes effect; consequently $\text{state}(t) = f(\text{ledger}_{\leq t})$ for a pure f, independent of any vendor session.

The invariant is enforced, not conventional: the event relation carries database rules that turn every UPDATE and DELETE into a no-op, and the per-project sequence number is claimed inside the same transaction as the insert, so every ledger range has a well-defined digest. Corrections are new events that supersede prior ones. The function f is a set of pure projections (open items, the decision log, the cost report) computed from the event sequence alone.

Crash recovery and engine handoff follow as corollaries: a successor engine re-derives context from the ledger alone,

[1] https://the-vibey-project.github.io/vibey/main/paper.pdf
[2] https://the-vibey-project.github.io/vibey/main/paper/
[3] https://the-vibey-project.github.io/vibey/main/book.pdf
[4] https://the-vibey-project.github.io/vibey/main/book.epub
[5] https://the-vibey-project.github.io/vibey/main/book-print.html

so a credit exhaustion on e_i between turns of an item reschedules the item onto e_j with the open-question set intact.

III. The no-loss handoff gate

A handoff from e_i to e_j carries a brief β: objective, constraints, done and remaining work, open questions, decisions, assumptions, findings, artifacts, and a reference ρ to the ledger range it summarises. The brief is admitted only if a pure predicate gate(ledger$_\rho, \beta$) holds, evaluated without any model call.

Invariant 2 (No loss). Let $\text{open}_k(\rho)$ be the ids opened by event kind k in range ρ and not closed or superseded. The gate holds iff $\text{open}_k(\rho) \subseteq \text{ids}_k(\beta)$ for $k \in \{$questions, decisions, assumptions, findings, artifacts$\}$ (R2–R5, R7); every remaining-work item of the last verdict appears in β (R1); the digest of ρ recomputed from the ledger equals the digest carried by β (R6); the budget carried by β agrees with the ledger's (R8); every hard constraint of the specification is restated (R9); and no free-text field of β carries a tool grant, a permission change, or an acceptance-criteria mutation (R10).

The gate has modes. In strict mode all ten rules apply, and a failing brief gets up to three strict attempts, each regeneration fed the specific violations. It then escalates to full-transcript mode, which makes the brief advisory: the successor is to work from the whole range ρ, so the gate stops checking R1–R5, R7 and R9, while R6, R8 and R10 still run, because they check facts about the range and the brief's containment rather than its completeness. In the current implementation the range reaches the successor as a file rather than inline: the full ledger is written into the receiving worktree and the seed prompt names it, so in this mode completeness rests on the successor reading that file, not on the gate. Inlining the range into the seed, or keeping the closure rules enabled until it is inlined, is open work. Further failure parks the item on a human gate; a fourth, forced mode is reserved for an explicit operator override. A handoff that fails the gate is therefore a retry, an escalation, or a human decision, never a silent partial.

IV. Queue semantics

Work items form a relation Q in PostgreSQL. Workers claim with

```
SELECT ... FOR UPDATE SKIP LOCKED
```

which yields two properties without any global lock: mutual exclusion per item (at most one worker holds item w at any instant) and non-blocking progress (a worker never waits on a peer's claim, so throughput scales as $\min(|Q|, |\text{workers}|)$).

Within a project, claims are ordered by priority descending, then by the earliest permitted run time, then by id, and they exclude items whose dependencies have not succeeded. Within one priority class an item can be bypassed only while it is held, and every hold is bounded by a lease: a claim sets the lease expiry to $\text{now} + L$, a live worker renews it, and a reaper returns any expired lease to the ready state. A crashed worker therefore costs at most L of delay and never a lost item. Across priority classes the discipline is strict priority, not arrival order. Because workers die and leases expire, every job is idempotent under replay.

V. The six-phase machine

$$\Sigma = \langle D, B, R, D_d, D_e, D_r \rangle$$

design, build, review, deploy-design, deploy-execute, deploy-review, with the human-gated subset $G = \{D, R, D_d, D_r\}$.

Invariant 3 (Gate soundness). For every $\sigma \in G$ the exit guard is a conjunction of an explicit human verdict recorded as a ledger row and, where the phase accumulates open items, the emptiness of a ledger-derived open set. $D \to B$ requires at least one acceptance criterion, every criterion mapped, and an accepting verdict; $R \to$ Done requires $\text{open}_{\text{findings}}(R) = \varnothing$ and an accepting verdict; $D_d \to D_e$ requires the deployment specification accepted and consent recorded; $D_r \to$ Done requires the demonstration accepted. Emptiness is never sufficient: no gate exits on the absence of objections alone.

A review finding therefore cannot vanish into a transcript: it is a ledger row that holds $\text{open}(R) \neq \varnothing$, and the machine cannot leave R for completion until a human closes it. Loop-back is a routing decision over the findings: an unambiguous finding routes $R \to B$, and a finding that needs clarification, or a project configured for strict loop-back, routes $R \to D$. Both are ledger transitions, not ad-hoc prompts.

A gate is not a blocked thread. A handler that needs a human returns a park value; the worker records a gate row, marks the item as awaiting a human, releases its lease, and claims the next item. The answer is itself a ledger row that re-readies the item. Human latency therefore never holds a queue slot, and the non-blocking progress of the previous section survives humans in the loop.

Two refinements do not change the analysis. An optional visual-design interstitial V may be inserted between D and B on explicit opt-in; it is human-gated on the same terms, exiting only when the visual plan is accepted or explicitly waived. The deployment triple $\langle D_d, D_e, D_r \rangle$ is entered only on an explicit opt-in recorded in the ledger; declining records a successful local completion.

VI. The engine family

Five runners implement engines: claudeloop over Claude Code, codexloop over OpenAI Codex, cursorloop over Cursor's agent and its Cloud Agents API, agyloop over Gemini through the Antigravity SDK, and qwenloop over Qwen 2.5 Coder served on local hardware. claudeloop came

first; the others transplanted its core. The orchestrator depends on a narrow contract that all five honour: a bounded run, a done marker, an event vocabulary, a capacity mapping, and a shared wind-down exit code (75) meaning that the engine ran out of window capacity mid-item and stopped cleanly after writing its state. Capabilities beyond the contract (savepoints, unwind, mid-run prompts and others) differ by runner and are declared per engine.

A. Bounded runs that never block

Unattended operation imposes three requirements that interactive tooling never meets: no turn may wait on human input; every resource the vendor meters must be bounded above by the operator, not the model; and a run's outcome must be auditable from durable state rather than from a transcript inside a vendor session. A run is therefore admitted only under an explicit bound vector (turns, spend, wall clock, per-turn and output-silence watchdogs, and a ceiling $W_{\max}$ on any single capacity wait; the exact members vary by runner) and ends at the first bound reached, with that bound recorded. Budget enforcement is preemptive: the run stops before an overrun, never after.

Invariant 4 (No interactive waits). No execution path may block on standard input. Where the vendor's tool can prompt, managed hooks pre-answer it and the session preamble declares unattended operation; everywhere, the watchdogs convert any residual hang into a loud, bounded failure.

Every turn, verdict and spend entry of a run is one line of JSON in an append-only trail under the run directory, so the runner obeys the same write-ahead discipline as the orchestrator, at the scale of one session.

B. Windows versus credits

The discrimination the family is built around separates a window, a rate limit that will reopen, from exhausted credits, a balance that no amount of waiting refills.

Invariant 5 (Waitability). A window is waitable under a deadline: a bounded probe re-tests capacity and the excursion is capped by $W_{\max}$. A credits state carries no deadline at all. It may be re-probed on a bounded backoff in case a human tops the balance up, but it is never scheduled as if it will reopen at a time.

In the orchestrator, capacity is four-valued (available, a waitable window with an optional reset time, exhausted credits, failed authentication), and the rule that credits have no clock is enforced at three independent layers: the credits type has no reset field, a property test asserts that no credits state ever produces a deadline, and a database CHECK constraint rejects an engine-health row that pairs exhausted credits with a reset time. The runners classify in the same order of precedence. claudeloop checks credit signals before the window path, so a billing failure can never be read as a waitable window even when a stray reset time rides along with it. codexloop is taxonomy-faithful: classification keys on the vendor's machine-readable error code and type before anything else (insufficient_quota is an exhausted quota, never a window), consults the HTTP status only when neither decides it, and never lets a status, a retry header or a completion claim outrank a body-level billing marker.

Theorem 1 (Capacity outranks completion). A completion claim observed while capacity is not available is recorded but not believed: a starved model emits plausible final output, so a run that ends for capacity is always attributed to capacity and never laundered into success.

Completion is read from a structured per-turn verdict where the vendor supports one, with an explicit done marker as the fallback, and the capacity verdict is consulted first either way.

C. Quotas and hardware as capacity

Two runners stretch the taxonomy at its ends. Gemini meters by per-model quotas whose dimensions do not collapse, so agyloop carries a five-member state (available, transient throttle, an exhausted named window, exhausted credits, failed authentication) and computes quota boundaries from the vendor's Pacific-time day. Its probes may tune when the next probe fires, never whether a wait ends: every excursion stays capped by $W_{\max}$. Its generated REST surface is pinned by a drift test to a committed snapshot of the vendor's discovery document, so a regenerated client that diverges fails the suite instead of a run.

qwenloop has no vendor. Its capacity is hardware, with the states available, locally busy and misconfigured, and nobody refills it by adding a payment method. Model acquisition is explicit: the doctor never downloads weights, and installing a model is a separate, deliberate command, because a surprise multi-gigabyte download is itself a capacity event on the constrained machine the runner exists to respect. Because no external party can revoke its capacity, it is the family's sovereign fallback: it is an opt-in standby for BUILD, and it is the preferred provider for the DESIGN interview, which it can run with no vendor account at all.

D. The transplant thesis

claudeloop's wager was that everything vendor-specific fits in two places: the lexicons that read a vendor's failure text and the transport that speaks to it. Four retargets tested it. The invariants above survived each one; the vocabularies did not stay identical. The runners' capacity unions range from three members (qwenloop) to six (codexloop, which separates throttles, windows, quota, authentication and transient backend errors), and each runner names its states in its own terms. The orchestrator can therefore treat the pool as substitutable executors and

choose among them by policy rather than by state: the delivery semantics live entirely above the vendor line.

E. Budgets and engine selection

Budget caps are per item (turns) and per cycle (dollars), summed from the cost the engines report on each completed turn; engines carry cost rates, not caps. An exhausted budget parks the item before a session starts, never after.

At each rotation point the eligible engines (installed, conformant, authenticated, circuit not open) are ranked by smooth weighted round robin. Each candidate carries an effective weight $w_i = \max(1, \mathrm{round}(b_i h_i f_i c_i a_i))$ for base weight b_i, health h_i from a per-engine circuit breaker, fidelity f_i of the engine's effort projection to the requested effort, a cost factor c_i (fixed at 1 in the current implementation), and a warm-session affinity a_i. The selector adds w_i to each candidate's running total, picks the maximum, and subtracts $\sum_i w_i$ from the winner, so the sequence is deterministic and spreads load in proportion to weight without bursts. Rotation fires only at boundaries (a new item, a capacity rejection, which excludes the rejecting engine, a graceful wind-down, an effort escalation, an engine crash, or a phase transition) and never inside a turn, so every handoff has a well-defined ledger range ρ.

VII. Exact-head evaluation and the release calculus

The orchestrator's output is a pull request, and what happens to it is governed by vibey-gh. Let a pull request's evolution be the finite sequence of heads $H = \langle h_0, h_1, \ldots, h_n \rangle$, each a revision replacing its predecessor. An automation system emits claims (scan results, review verdicts, gate conclusions) and acts on them by merging, releasing, or halting for an operator. The folk assumption is that a claim about h_i speaks for the pull request. It does not: it speaks for h_i alone.

Invariant 6 (Exact-head). Every claim is a pair (c, h_i), and no decision procedure over head h_j may consume a claim (c, h_i) with $i \neq j$.

Evaluation is a pure function over the head, its check results, and a persisted state $\sigma = (a, r, v, k)$: attempts consumed, last-reviewed revision, its verdict, and refills used. For head h and repair budget A,

$$E(h, \sigma) = \begin{cases} \mathsf{pending} & \text{checks incomplete} \\ \mathsf{review} & r \neq h \\ \mathsf{ready} & r = h \wedge v = \top \\ \mathsf{repair} & r = h \wedge v = \bot \wedge a < A \\ \mathsf{blocked} & r = h \wedge v = \bot \wedge a \geq A \end{cases}$$

Invariant 7 (Budget placement). The guard $a \geq A$ is evaluated only where a repair would be spent, strictly after the freshness test $r \neq h$. Reviews are free; only repairs are counted.

Theorem 2 (Bounded convergence). Absent contributor pushes, every lineage reaches a terminal state in at most $A\,(1 + k_{\max})$ repairs, where $k_{\max}$ bounds operator refills.

Proof sketch. Heads advance only via repairs, since a contributor push begins a new lineage by definition. Each repair increments a; a is bounded by A; a resets only via refill, itself bounded by $k_{\max}$. The transition relation admits no cycle that leaves (a, k) unchanged, so the lexicographic measure $\mu = (k_{\max} - k,\ A - a)$ strictly decreases across every non-terminal loop, and ready and blocked absorb. □

A production violation. With the budget guard evaluated before the freshness test, the following trace is reachable: reviews of h_0 to h_2 fail with findings, repairs produce h_3 addressing all of them, $a = A$, and evaluation of h_3 hits the budget guard first and emits blocked. The lineage is escalated as unrepairable on the verdict of h_2, a revision that no longer exists in the pull request. This trace occurred in production: the escalation's own report listed an empty set of remaining failures. Commit 4afafbae reordered the two guards, restoring the budget-placement invariant, and its regression test asserts that the counterexample now yields review.

Trust separation. Three principals with pairwise-disjoint capabilities operate the loop: the per-job platform token publishes claims but cannot act; the reviewing credential proposes revisions on guarded branches but cannot merge; the merging credential consumes verdicts but can never produce them. No single credential can both judge and act, so a compromised judge cannot ship and a compromised actor cannot self-approve. Untrusted third-party revisions are data to all three principals.

Release monotonicity. Versions are derived, not remembered: for mainline M and change set Δ, $\mathrm{ver}(M \cup \Delta) \geq \mathrm{ver}(M)$, with equality exactly when Δ carries no shippable content, and the derivation is idempotent, so re-promoting identical content never compounds a bump. Published versions form a monotone sequence, and the publish step treats an already-published version as a no-op, never an error.

Degraded modes as first-class states. The evaluator's own substrate (API credit, the hosted review lane, the operator's editor) can refuse service while the repositories remain healthy. Each lane is modelled as a probe $p : \mathbb{T} \to \{0, 1\}$, and sovereign operation requires, for every lane, a fallback lattice $L_0 \succ L_1 \succ \cdots \succ L_k$ in which each L_{i+1} is strictly harder to refuse than L_i and control rests at the least i with $p_i(t) = 1$. The operator seat is a two-bit automaton over (paid lane healthy, local seat engaged) that reclaims the seat for the paid lane one probe cycle after funding returns and reports a lattice with no passing seat as an alarm rather than absorbing it. Handoffs between seats stay lossless because pushes use a compare-and-swap on the remote ref whose lease is captured before

any fetch; fetching first would silently re-arm the lease and reduce the swap to an overwrite.

VIII. Deterministic retrieval and fail-closed bootstrap

Two further components apply the ledger's discipline, append before acting and fail closed rather than degrade silently, at other scales.

Retrieval. A skill library cannot be loaded wholesale into a context window, yet fragmentary retrieval of safety- and correctness-critical guidance is worse than none. At revision 559638f4 the library holds 710 skill documents across 135 plugins. vibey-skills indexes it as a pure function of the corpus and retrieves only whole sections.

Invariant 8 (Deterministic, whole, fail-closed retrieval). The index is a pure function of the corpus: identical corpus bytes yield identical manifests, chunk identifiers and scores. The retrieval unit is a heading-bounded section, returned whole with its path, line range and content hash. For a request with mandatory content M and budget B, if tokens(M) $> B$ assembly returns budget_insufficient and never truncates a mandatory section to fit.

Omitted optional content is recorded in the packet's manifest, and a low_confidence flag directs the caller back to native full-skill activation, so degradation is explicit and machine-readable. Full-text queries are parameterised, symlinked sources fail closed, and credential-shaped strings are redacted from packets. Retrieval metrics are diagnostics, not the objective: the deployment criterion is cost per accepted work item, since a packet that halves token spend but raises rework is a regression.

Bootstrap. vibey-bootstrap collapses a cloud workload's first hundred lines (logging, configuration, secrets, telemetry) into one call under one rule: absence of a required precondition halts the start with that precondition named, and an optional capability degrades explicitly, as a recorded decision, never as a silent absence. Above it sit delivery and audit primitives with stated guarantees. A transactional outbox writes intent beside the state change, so every committed intent is delivered at least once and marked delivered exactly once. Audit records form a hash chain, $c_0 = H(r_0)$ and $c_i = H(c_{i-1} \| r_i)$, so any edit, insertion or truncation breaks every later link and verification needs no trust in the writer. Retries are built in one place, with typed retryable errors, exponential bounds, and rate-limit callbacks that can never mask the original error.

The ledger, the outbox and the audit chain are one principle at three scales: the record of intent exists before its effect, and every later state is derived from the record.

IX. Production rate and governance

The components above make engines substitutable and human decisions explicit. This section asks what, given that, bounds the rate of delivery. It uses two tracked sources and nothing else: the sovereignty stress record (src/vibey_tools/gh/docs/sovereignty-stress-2026-08-30.md), a controlled escalation of the local review lane, and this repository's git history, which is field data. scripts/paper_evidence.py recomputes every figure in this section from those two sources; history figures are stated at revision 559638f4, which the script's –rev 559638f4 reproduces.

A. The stress record

A harness fired N simultaneous generations at qwen2.5-coder:14b, served by ollama on one machine with 24 GB of memory and 10 cores, for N from 1 to 128, with a 900 s deadline per generation. Each generation was a real unit of work, an issue triaged or a pull-request diff reviewed, drawn from a pool of seven artifacts of 2 to 22 KB. Throughput is successful generations per minute of rung wall clock.

N	Succeeded	p50 (s)	Per minute
1	1/1	166	0.36
2	2/2	118	0.99
3	3/3	132	0.99
4	4/4	154	1.17
6	6/6	59	1.72
8	8/8	191	1.27
12	12/12	174	1.54
16	16/16	440	1.37
24	21/24	701	1.40
32	30/32	292	2.00
48	24/48	900	1.60
64	40/64	555	2.66
96	53/96	821	3.52
128	23/128	901	1.53

In all, 243 of 444 generations succeeded over 2.18 hours. Every failure was a clean timeout; not one response was malformed or corrupt.

B. The measured regularity

Observed regularity. On a fixed substrate, with hardware, model, serving stack, deadline and admission rule held constant, once offered load saturates the substrate the rate of successful output lies in a bounded band that depends on the substrate and only weakly on the quantity of work offered. In the record, across the nine rungs from $N = 2$ to $N = 32$ (107 generations, 102 succeeded, 95.3%, and every rung at or above 87.5%), throughput stayed between 0.99 and 2.00 per minute, with mean 1.38 and sample standard deviation 0.33, while offered concurrency rose sixteen-fold. The least-squares slope of log throughput on $\log N$ over those rungs is 0.20; output that scaled with demand would have slope 1.

The band is a band, not a constant. Its spread is a quarter of its mean, and it drifts upward with N, because the server batches concurrent sequences into shared forward passes. It holds only inside the region. Below it, the serial rung ran at 0.36 per minute: one job cannot saturate the substrate. Above it, throughput

peaked at 3.52 per minute at $N = 96$, but only 55.2% of generations succeeded, and at $N = 128$ success collapsed to 18.0% while throughput fell back to 1.53.

A held-out check. A band fitted on the rungs with $N \leq 16$ is $[0.99, 1.72]$. Of the two stable rungs held out, $N = 24$ (1.40) fell inside and $N = 32$ (2.00) fell 16% above. The floor held; the ceiling was exceeded once, in the direction that continuous batching predicts. We therefore claim the floor and the scale of the ceiling, not a tight window.

A correction. The vibey-gh tenant paper reported this experiment as 61 generations at a success rate of 1.00, with throughput held to 1.4 ± 0.25 per minute and latency growing linearly at 22 s per added job through $N = 12$, then superlinearly at 59 s per job by $N = 16$. None of these survives comparison with the record. No contiguous run of rungs totals 61 generations, whether counted as attempted or as succeeded; four of the nine stable rungs lie outside 1.4 ± 0.25; and the record itself fitted both latency models during the run and falsified both. The tenant paper now carries the figures above, with a dated correction note.

C. Commodity production and scarce governance

Thesis. For a bounded task class, on a fixed substrate, under a fixed governance regime, production behaves as a commodity: its executors are substitutable, its units are priced, and its rate is set by the substrate rather than by demand. What the same records do not show as a commodity is governance, meaning the authority to decide, fund, accept and ratify, and correct judgment about what was produced.

The first half rests on three observations. Substitutability is a property of the design: five engines implement one contract, the scheduler chooses among them by policy, and the live runbook forces a rotation between two of them in the middle of a project. Price is explicit: every engine descriptor carries per-token cost rates, and budgets are sums of reported cost. The rate is the regularity above.

The second half rests on what stopped work. The stress run's collapse was the substrate's bound being exceeded, not a malfunction. At the break, the harness's heal probe recorded lane responsive; no heal needed: the lane was healthy and simply could not do that much work before the deadline. Every remedy (less concurrency, a longer deadline, more hardware) is a decision the operator makes, and the deadline is itself a governance parameter: moving it moves the band. The record's lasting correction was a governance rule, too: admission should gate on payload size, not on concurrency count. ADR-0035 records the same shape inside the orchestrator. A review-independence rule, made absolute in two places, deadlocked BUILD on a one-engine pool, deferring the same job forever with nothing in the ledger. Capacity to produce was present; a rule stopped it; the repair was a decision about the rule.

Judgment is the other scarce input, and we do not claim that governance is the only one. The design already treats judgment as scarce: ADR-0035 calls an engine grading its own work the weakest possible check, the release calculus separates the credential that judges from the one that acts, and every human gate requires an explicit verdict rather than an absence of objections. The records show why. The exact-head violation was automation acting correctly on a stale verdict. The figures corrected above were a confident, wrong claim that stood unchallenged in a tracked paper from the day its package was imported until this revision checked it against the record. Artifacts are commoditised; correct judgment about artifacts is not, and that gap, more than throughput, is where the remaining engineering lies.

D. Six materials and the modulators of the rate

vibey-gh postulates that every dilemma in the practice of software engineering decomposes into six materials, each with three properties, plus the pairwise couplings between them.

Material	Available	Stable	Reliable
Network	reachable	steady	delivers
Hardware	capacity	no drift	computes
Software	installed	pinned	to spec
Agent	present	rested	correct
Information	at hand	fresh	true
Agency	permitted	unrevoked	honoured

Write the state as $x \in \mathbb{R}^{18}$, one coordinate per material and property, and let x^* be the state of long-term stable peak performance. The dilemma vector is $d = x^* - x$. Agency is not agent: an agent may be present, rested, correct and informed and still lack the power to act, which the release calculus's trust separation imposes by design on the credential that reviews. Coordination is a coupling rather than a seventh material, since it consumes information transfer, agent waiting and authority resolution. For an operation o with requirement vector r_o,

$$\text{feasible}(o) \iff x \succeq r_o$$

$$T(o) = T_0(o) \prod_i \phi_i(d_i)$$

$$C(o) = \int_0^{T(o)} c(x(t))\, dt$$

where ϕ_i is the dilation that a shortfall in coordinate i imposes on duration: unity at $d_i = 0$, and divergent as the coordinate approaches its floor. The gradient $-\nabla_d T$ ranks which shortfall to repair first. The postulate is falsified by exhibiting a real dilemma that does not decompose into these coordinates and their couplings; it is a postulate, not a law.

The regularity holds its substrate fixed. Relaxing each modulator moves the band, and each maps to a coordinate of d:

- Hardware maps to the hardware material's availability and stability, and is measured. Memory was the eventual constraint: paging space reached 99.1%

during the rung at $N = 128$, where the collapse and the crashes coincide, and the serving process died under context pressure at rungs 64 and 128 and was restarted automatically each time.

- Network stability maps to the network material's stability, and is not measured, because the stress run was local by construction. For the paid engines it is designed for rather than measured: a window is re-probed under a deadline, and a handoff moves the work to another engine.
- Operator availability, including rest maps to the agent material's availability and stability, and is not measured. Authorship in the history does not record whether an agent or the operator did the work (automated repair commits appear under both the bot's name and the operator's), so it cannot separate the two; what it shows is production at every hour, described below.
- Governance and agency maps to the agency material's availability, and is set rather than measured: the deadline and admission rule of the stress run, and the human gates and merge authority of the delivery process.
- Information freshness maps to the information material's stability and reliability, and has one measured instance, corrected by this revision: the stale figures above.

E. Time to completion

The regularity's use is prediction. Let $r_{\min}$ and $r_{\max}$ bound the measured band. For W units of the task class admitted in the stable region, with every coordinate at its target ($d = 0$, so each $\phi_i = 1$),

$$T_0 \in \left[\frac{W}{r_{\max}}, \frac{W}{r_{\min}} \right]$$

and in any state $T = T_0 \prod_i \phi_i(d_i) \geq T_0$, since every $\phi_i \geq 1$. T_0 is the zero-shortfall time-to-completion: the ideal time, against which every shortfall is a dilation. For the record's substrate and $W = 100$ units, the band $[0.99, 2.00]$ gives T_0 between 50 and 101 minutes, where the serial rate alone would give 278. The interval is wide because the band is, and replication would narrow it. Because the held-out check exceeded the ceiling, the firm half of the prediction is the upper end: with every coordinate at its target, the work should take no longer than $W/r_{\min}$.

Governance has a price in time, and the price can be lowered without lowering the bar. The workstream that parallelised the orchestrator's test suite measured the four per-layer coverage gates falling from about 1,530 s (four sequential suite runs) to 136 s, an 11.3-fold reduction, by computing all four floors from one instrumented run, and the bare suite falling from 383 s to 135 s, without removing a gate (docs/runbooks/expansion/evidence/13-front1-validation.md). That is a governance dilation made smaller while the requirement stayed the same.

F. Field data

The git history is field data: nothing in it was held fixed. At revision 559638f4, 1,103 commits are reachable across nine root histories, the absorbed histories of the family's packages. Since 2026-08-09, when the family's own development begins, 1,091 commits landed on 28 active days, between 1 and 130 per day (median 31, mean 39.0, sample standard deviation 31.7). Commits landed in all 24 hours of the day in US Eastern time, with the fewest (15) in the 09:00 hour and the most (82) in the 02:00 hour. The longest pause was nine days with no commit, from 2026-08-31 to 2026-09-08, and nothing in the repository records its cause. Eleven vibey release tags point at commits dated between 2026-08-16 and 2026-09-18, and 503 commit subjects across the absorbed histories end in a pull-request reference.

The daily rate's spread is 81% of its mean, against 24% in the controlled region. That is what the model predicts when the coordinates of d vary freely, but it is also what almost any model would predict of an uncontrolled process, so the history does not test the regularity. We report it so that the controlled band is never mistaken for a field rate.

G. Falsification

The regularity is falsified, for its substrate, by any of the following:

- a replication on the same substrate and task pool in which a stable-region rung (success at least 87.5%) runs below 0.99 per minute, or in which the band's mean moves by more than its measured spread of 0.33 per minute;
- a log-log slope of throughput on N near 1 in the stable region, which would make the rate demand-set rather than substrate-set;
- a set of W units, admitted in the stable region with every coordinate at its target, that takes longer than $W/r_{\min}$ to complete.

The commodity thesis is falsified by a stall, in the tracked record, that was caused by a shortage of production capacity while every governance coordinate stood at its target: a funded, permitted, rested and informed system that simply could not produce.

H. Scope

This is evidence, not proof, and its scope is narrow: one machine, one model served one way, one deadline, one artifact pool whose payload mix was not balanced across rungs (the record names payload size, not concurrency, as what decided survival), one session, and one operator. The field data is one project's history. Neither source measures network state or operator availability, so two of the five modulators are named, mapped and unmeasured. We do not claim a natural law, and we do not claim that the rate is constant. We claim a band, on a substrate, together with the conditions under which the claim would fail.

X. Validation

The model is validated at three levels. At the property level, the gate, the phase guards and the selector are pure functions under a 100% branch-coverage floor per architectural layer, with property tests on the selector and the credits type. At the chaos level, concurrent workers process a job set against a real PostgreSQL while each randomly abandons claimed jobs mid-flight and a concurrent reaper reclaims the expired leases; the verified property is no double execution, no lost job, and every job terminal. At the live level, a runbook drives one project from design through build and review to local completion on two paid engines, claudeloop and agyloop, including a forced rotation between them. Live runs over the other engines rest today on a scripted-binary conformance suite that asserts each runner's flags, run-directory shape, event vocabulary, capacity mapping and completion marker against the installed binary; they are not yet reported here. The production-rate claims are validated separately, by the stress record and the evidence script above.

XI. Related work

Queue-based job schedulers built on SKIP LOCKED provide claims, leases and retries but no delivery semantics; the ledger here is event sourcing applied above the queue, with the write-ahead discipline of ARIES and the lease bound of Gray and Cheriton. Agent frameworks such as SWE-agent and AutoGen provide sessions and tool loops but bind state to one vendor's context window; here the session is disposable and the ledger is not. Selection reuses nginx's smooth weighted round robin and the circuit breaker pattern. Platform-native automation (merge queues and required checks) enforces revision-bound checks but leaves verdict freshness to its consumers; the exact-head calculus closes that gap. Yanking semantics for published artifacts (PEP 592) informed the report-only supersession of releases, and keyless publishing through PyPI's Trusted Publishers removes stored release credentials. Degraded-mode design follows the fail-operational tradition, with the difference that our refusals include commercial ones, which is why fallback lattices are ordered by how hard a lane is to refuse rather than by mean time between failures. The retrieval engine is lexical over SQLite FTS5; dense retrieval would reintroduce the nondeterminism the reviewability invariant forbids. The upward drift of the throughput band is the continuous batching of Orca, and the coordination cost that the six-material postulate treats as a coupling is the one Brooks described for human teams.

XII. Conclusion

Putting the ledger, not the session, at the center makes autonomous delivery survivable and auditable: engines become fungible, crashes become replays, and human authority is a structural property of the state machine rather than a prompt-engineering hope. The same discipline, binding every claim to the state it describes, governs the runners beneath the orchestrator and the release calculus above it. In the project's controlled record, production kept to a band set by its substrate, and the inputs that stopped work were decisions and judgments. The engineering that remains is less about producing faster than about deciding well and cheaply.

References

[1] PostgreSQL Global Development Group, SELECT — The Locking Clause (FOR UPDATE SKIP LOCKED, available since PostgreSQL 9.5). PostgreSQL documentation, https://www.postgresql.org/docs/current/sql-select.html.

[2] M. Fowler, Event Sourcing, 2005. https://martinfowler.com/eaaDev/EventSourcing.html.

[3] P. Helland, "Immutability Changes Everything," Communications of the ACM 59(1):64–70, 2016. doi:10.1145/2844112.

[4] C. Mohan, D. Haderle, B. Lindsay, H. Pirahesh, and P. Schwarz, "ARIES: A Transaction Recovery Method Supporting Fine-Granularity Locking and Partial Rollbacks Using Write-Ahead Logging," ACM Transactions on Database Systems 17(1):94–162, 1992. doi:10.1145/128765.128770.

[5] C. Gray and D. Cheriton, "Leases: An Efficient Fault-Tolerant Mechanism for Distributed File Cache Consistency," Proceedings of the 12th ACM Symposium on Operating Systems Principles, 1989, pp. 202–210. doi:10.1145/74850.74870.

[6] nginx, smooth weighted round-robin upstream balancing, introduced in nginx 1.3.1 (2012), ngx_http_upstream_round_robin.c.

[7] M. T. Nygard, Release It! Design and Deploy Production-Ready Software, 2nd ed., Pragmatic Bookshelf, 2018 (the circuit breaker pattern).

[8] J. Yang, C. E. Jimenez, A. Wettig, K. Lieret, S. Yao, K. Narasimhan, and O. Press, "SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering," NeurIPS, 2024. arXiv:2405.15793.

[9] Q. Wu et al., "AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation," 2023. arXiv:2308.08155.

[10] G.-I. Yu, J. S. Jeong, G.-W. Kim, S. Kim, and B.-G. Chun, "Orca: A Distributed Serving System for Transformer-Based Generative Models," 16th USENIX Symposium on Operating Systems Design and Implementation (OSDI), 2022.

[11] F. P. Brooks, Jr., The Mythical Man-Month: Essays on Software Engineering, Addison-Wesley, 1975.

[12] PEP 592, Adding "Yank" Support to the Simple API, Python Packaging Authority, 2019.

[13] PyPI, Trusted Publishers, https://docs.pypi.org/trusted-publishers/, 2023.

[14] GitHub, About protected branches and rulesets, GitHub Docs, 2024.

[15] SQLite, FTS5 Extension, https://www.sqlite.org/fts5.html.

[16] The vibey repository: the sovereignty stress record, src/vibey_tools/gh/docs/sovereignty-stress-2026-08-30.md; the evidence script, scripts/paper_evidence.py; the architecture decision records, docs/architecture/decisions/; https://github.com/the-vibey-project/vibey, 2026.