# Evidence-Gated Agentic Development: The Conexus Brain

**Author:** Kris Christopher (KrisCodes2) **Date:** 2026-07-23 — **second edition 2026-08-12**
**Home:** conexusbrain.com — companion video: "The Future of AI Was Invented in 1974" **Status disclosure:** Conexus is a pre-release, founder-operated system. Every capability claim in this paper carries one of four labels — **live**, **in flight**, **built-but-gated**, or **planned** — and the labels are maintained in a claims register before any public copy is written. Second-edition receipts were re-verified against the live system and the work board on 2026-08-12; where the board and the narrative disagree, the board wins and the text says so.

## Abstract

AI systems can now generate code, content, plans, and decisions faster than organizations can safely review them. The binding constraint on applied AI is shifting from generation to trusted acceptance: proving what was done, by which actor, under whose authority, and with what evidence. This paper argues that the techniques required are not new. Fifty years of computing history — the internet's narrow-waist protocol design [1], structured and event-sourced data management [6][7], distributed-systems failure handling [8][9][10][11][12], declarative reconciliation [13], observability [14], zero-trust security [4][5], software supply-chain provenance [15][16][17], policy as code [18], and continuous-delivery practice [19] — map directly onto the problems agentic AI creates. Conexus applies those techniques as a working system: a federated, single-writer organizational memory (the Brain), a coordination plane that lets agents from multiple vendors work in one repository without collisions, and a delivery model we call Evidence-Gated Agentic Development, in which agents move quickly precisely because the boundaries around them are explicit.

Since the first edition, the case study has advanced materially: the temporal operational ledger is live (authority classes, evidence references, supersession, as-of-time queries); the context-and-inference efficiency layer is live (a local Context Compiler, disposable physical sessions, and provider inference leasing); and the evidence-gated autonomy controls are implemented and in final review; and multi-tenant operation — every table tenant-scoped, every write identity-checked — is the storage model rather than an enterprise afterthought. The newest result is architectural rather than model-specific: **the model session is disposable; the operating system is durable.** We present the architecture, the operating evidence, the economics (including the composite cost reduction the efficiency layer produces), a risk-tiered model for expanding autonomy, and a roadmap, with limitations stated plainly.

## 1. The verification gap

Modern AI-assisted engineering has an asymmetry problem. A single operator can direct agents that produce hundreds of changes per week, but every change still needs to be reviewed, tested, integrated, and remembered. The organizations adopting agents report a consistent cluster of failures:

- **Amnesiac sessions.** Each new AI session starts without knowledge of prior decisions, fixes, and failures, so the organization re-derives what it already learned.
- **Colliding workers.** Parallel agents duplicate work, overwrite each other's changes, or lose handoffs — classic distributed-systems failures wearing new clothes.
- **Unproven work.** "The agent said it passed" is not evidence. Without deterministic verification and durable provenance, AI-generated changes accumulate risk invisibly.

The economics make the gap structural rather than temporary. Generation cost falls with every model release, while the cost of careful review, integration, and operation falls slowly if at all — so the ratio of produced-to-verified work worsens as adoption succeeds. A team that accepts unverified changes accumulates invisible risk; a team that reviews everything at human speed surrenders the throughput it adopted agents to gain. Worse, the loss is compounding: every unrecorded decision is re-litigated by the next session, every unprovenanced change lengthens the next incident investigation, and every collision between parallel workers burns the exact engineering attention the agents were supposed to free. The way out is not slower generation or braver acceptance. It is infrastructure that makes verification, memory, and coordination cheap enough to apply to everything.

The market's dominant answer — a more capable model — does not address any of these, because none of them are intelligence problems. They are systems problems: identity, state, ownership, timing, failure, authority, observability, and proof. The thesis of this paper: **models create possibilities; systems determine outcomes.**

## 2. Six lessons from fifty years of systems

### 2.1 The narrow waist

The internet scaled because its middle contract stayed small: many applications above IP, many networks below it, one stable interface between [1][23]. The AI ecosystem is re-discovering this shape. The Model Context Protocol standardizes how a model-driven application reaches tools and context [2]; the Agent2Agent protocol addresses communication between independent agents [3]. Connectivity, however, is the easy half. The internet standardized packet delivery without solving identity, fraud, or trust — and agent protocols will repeat that pattern unless identity,

authorization, reputation, and policy are built above them. In Conexus, a registry now serving 326 tools behind one search contract (live; 268 at first edition) is the narrow waist for agent tooling: any model, local or cloud, discovers capability the same way.

### 2.2 Memory must have structure

A transcript is not memory. An operational memory must answer: where did this come from, when was it true, has it been superseded, what evidence supports it, and who created it? Event sourcing keeps the history that produced current state [6]; CQRS separates broad read paths from narrow, audited write paths [7]. Conexus applies both: agents read the Brain freely, but every write crosses one gateway. Retrieval is hybrid — lexical for precision, vector for meaning, graph for relationships — over the same store (live), including a code-intelligence index of 12,853 symbols agents consult before touching source (8,600 at first edition).

What was roadmap in the first edition is now live. The temporal operational ledger (BRN-002, implemented 2026-08-07) adds authority classes — a signed pipeline result outranks an agent's claim — evidence references, supersession links, and as-of-time queries: what did we believe when this decision was made, and what superseded it?

The as-of question sounds academic until the first serious incident review. When a production defect surfaces, the useful question is rarely "what is true now" — it is "what did the system believe at the moment the change was authorized, which evidence was available then, and which later fact superseded it." A memory layer that can answer that converts postmortems from archaeology into queries, and it is the difference between an organization that learns and one that merely accumulates text.

### 2.3 Partial failure is normal

Distributed systems assume components fail independently, and agent systems inherit every one of those modes: a session dies after modifying files but before reporting; a tool call times out after succeeding; two workers believe they own the same resource. The correct stance is to treat every agent as a worker that can be slow, stale, duplicated, interrupted, or wrong — and to use the classical controls: heartbeats for liveness; leases that expire rather than locks that deadlock [8]; fencing tokens so a stale lease-holder cannot overwrite newer work [9] — the scenario is worth spelling out once, because it is the canonical agent-swarm accident. Agent A takes a lease numbered 41 on a work item and stalls. The lease expires; Agent B takes lease 42 and finishes the work correctly. Agent A eventually wakes and attempts to write its stale result. Without fencing, A silently overwrites B; with fencing, the protected resource remembers that 42 is the newest valid token and rejects 41. Stale authority must never become current authority, and no amount of model intelligence substitutes for that check — idempotency keys so retries cannot double-execute consequential actions [10]; transactional outboxes so state changes and their notifications cannot

diverge [11]; and sagas with compensating actions for workflows that cross irreversible boundaries [12]. In Conexus, session registration, cooperative file claims, and fenced work-item leases are live — and they are why agents from three vendors (Claude, Codex, and Grok) committed 227 times to one mainline across four days in July 2026, 104 commits on the busiest day, with zero reverts and zero collisions. (Attribution caveat, added in the second edition: the 227 count is all mainline commits in the window; per-model attribution rides on `Co-Authored-By` trailers, which roughly half the window's commits carry.)

### 2.4 Declare desired state and reconcile

Kubernetes replaced "run this and hope" with a controller loop: declare desired state, observe actual state, act to reduce the difference, repeat [13]. Agentic work should be governed the same way. A requirements document declares outcomes; work items bound obligations; verification profiles declare required checks; evidence schemas declare the proof that must exist. The controller then compares observed state against declared requirements — coverage exists or it does not; the artifact matches its schema or it does not — instead of asking the agent whether it is finished. This converts vague autonomy into measurable convergence, and it makes observability the load-bearing wall: the trace expands from request-level spans [14] to decision provenance — business goal to requirement to agent session to tool calls to commit to pipeline to evidence to approval to outcome.

Reconciliation also supplies the missing regulator for agent throughput: backpressure. When a downstream system slows or fails, controllers reduce work, queue it, or stop — whereas naive automation converts a small outage into a flood of retries and duplicate actions. And once every operation carries a durable identity from intent to outcome, reliability becomes computable by agent, model, task class, repository, and toolchain. Model selection then stops being brand preference and becomes an evidence question: which model succeeds most often on this repository, which produces the fewest repair cycles, which overclaims completion. That is how a system improves itself without ever letting the AI approve itself.

### 2.5 Sign what happened

Supply-chain security reframed builds around provenance: verifiable statements about where, when, and how an artifact was produced (SLSA [15]), attestation formats for supply-chain claims (in-toto [16]), and accessible signing and verification (Sigstore and Cosign [17]). Agentic development needs the same discipline extended backward into source provenance: which requirement initiated a change, which agent wrote it, which context influenced it, which policies constrained it, which repairs modified it, and which human approved it. Git records what content changed; the system around it must record why it changed, under whose authority, and with what proof. A future evidence bundle for an agent-produced change should therefore bind the source

revision, the agent session and model, the work authorization, the tool versions, the checks executed with their canonical states, the policy profile that interpreted them, the approvals, and the final artifact digest — one chain from intent to deployment. Equally important are canonical result states: a pipeline that only knows green and red will mistake a missing scanner for a clean scan. Conexus's verification model distinguishes PASS, FAIL, SKIPPED-NOT-APPLICABLE, INFRASTRUCTURE ERROR, TOOL MISSING, and EVIDENCE INVALID (live), and its full suite — 11,271 tests across 1,190 files at second edition (8,347 / 908 at first) — runs in about six minutes, because fast feedback is what makes small changes cheap [19].

### 2.6 Protect the objective

Optimizing systems learn to satisfy the measurement rather than the goal — Goodhart's law [20]. In agentic engineering the canonical failure is the repair agent that deletes a failing test to make the pipeline green. Conexus's local repair loop (Ralph, live) is therefore bounded by construction: it may fix code, configuration, routing, or infrastructure, and it may not lower thresholds, remove meaningful tests, disable scanners, or convert required checks into advisory ones. The rule generalizes: **the optimizer must not control the definition of success.** Model improvement follows the same logic. In a champion-challenger arrangement, the current best model for a task class stays in service while alternatives run in shadow mode or on selected low-risk work, judged by the same independent criteria — verified success rate, cost per accepted change, repair cycles, human-intervention rate, regression rate after deployment — so promotion comes from measured evidence rather than marketing. And high-impact behavior earns authority through simulation first: a publishing agent validates actions without sending them, a deployment agent produces the plan without applying it, a repair agent patches an isolated workspace and passes verification before any shared state changes. Aerospace, networking, and finance all learned the same sequence — test under controlled conditions before connecting the controller to an irreversible surface.

## 3. The Conexus Brain architecture

The Brain is the memory-and-coordination substrate of Conexus.

**Federated store (live).** One DuckDB database — 375 tables as of 2026-08-12 (343 at first edition) — carries a `project_id` on every table, so every product, agent, pipeline, and dashboard shares one source of operational truth. A full identity-key migration (47 tables, 110,811 rows) was executed live with zero corruptions using full-table rebuilds — never in-place key updates — a working example of the risk-tiered ceremony this paper advocates.

**Single-writer gateway (live).** All external writes cross one HTTP gateway: token-free reads, bearer-gated writes and transactions through a single writer mutex, work-item leases with fencing

tokens, and hybrid search and graph traversal. The narrow write path is a scar-derived design: an earlier in-place primary-key update corrupted the write-ahead log, and recovery worked only because snapshots and log replay were designed before the failure.

**Coordination plane (live).** Sessions register and heartbeat; file claims detect collisions cooperatively; work items carry leases and fencing tokens; durable messages carry acknowledgment requirements; checkpoints and handoffs let a successor session — any vendor — resume without chat history. This is the infrastructure behind the multi-vendor receipt in section 2.3.

**Memory with provenance (live).** A decaying fact store captures decisions, fixes, and failures automatically from CI, repair, and session events, with ranked recall injected at session start. Since the first edition the specified next epic has shipped: the temporal ledger (BRN-002) adds authority classes, verification-evidence links, supersession, and as-of-time queries — all live as of 2026-08-07. Alongside the fact store sit the retrieval surfaces agents actually work through: hybrid lexical-plus-vector search, graph traversal over entities and relationships, the 12,853-symbol code-intelligence index, and a curated research corpus of 3,454 documents that agents are required to consult before the open web — so marketing and strategy answers stay traceable to sources the company chose.

**Context and inference efficiency (live, new in this edition).** The AOE-001 layer (implemented 2026-08-10) treats context and frontier-model inference as schedulable resources. A **Context Compiler** assembles a bounded, provenance-bearing Context Packet for each physical model session from canonical state: the approved requirement, current Git state and diff, authoritative Brain facts, code symbols and precise source ranges, current ownership, the latest checkpoint, current verification evidence, applicable policy, and the model/context budget. A logical work item may span many short physical sessions — `session A -> checkpoint -> session B -> checkpoint -> verified completion` — and no physical session owns the truth. Provider inference is leased: a logical worker holds no upstream slot while it runs local tests or CI, so logical swarm size stays independent of a provider's concurrency envelope. Verification, not the generator's narrative, is the stop condition.

**Durability posture (live).** Knowledge admission is queue-backed: a durable ingest queue with retry and an outage spool means nothing is lost when the writer is busy or down. External writes take an automatic snapshot first; the write-ahead log self-heals on restart; and every known corruption class has a documented, tested recovery path. Institutional memory that cannot survive a crash is not institutional memory.

**What the Brain is not.** It is not a SaaS: the deployment model is customer-controlled infrastructure, with cloud capacity as an explicit, bounded exception. It is not multi-writer: that limit is stated rather than hidden. And it does not replace Git or CI — it binds them to intent, authority, and evidence.

## 4. Evidence-Gated Agentic Development

> *Evidence-Gated Agentic Development (EGAD) is a software-delivery model in which AI agents produce and integrate small mainline changes, while commit, push, promotion, and release authority remain constrained by deterministic verification, signed evidence, policy gates, and human-controlled exceptions.*

The operating loop has ten stages, and each exists to kill a specific failure mode: **observe** current system and business state (no work from stale assumptions); **recall** prior decisions, fixes, and failures from the Brain (no re-deriving what the organization already paid to learn); **plan** bounded work with declared verification requirements; **claim** ownership through leases and file claims (no silent collisions); **compile context** — build the bounded Context Packet from canonical state (new in this edition; no session inherits a transcript as its memory); **execute** inside the claimed boundary; **verify** with deterministic tools emitting canonical result states; **approve** consequential transitions per policy, with humans holding the exceptions; **ship and observe** the deployed outcome, not just the merge; and **learn** — write the verified outcome back into memory where the next session's recall will find it. Three properties distinguish the loop from "autonomous coding":

1. **Authority derives from state, not confidence.** An agent's report that work is complete changes nothing; the evidence bundle attached to the current commit does.
2. **Capabilities are rungs, not a switch.** Generate, stage, commit, push, merge, promote, and deploy are separate delegable capabilities, expanded per task class from verified history — trunk-based small batches [19] make each rung cheap to check.
3. **Verification is independent of generation.** Deterministic tools, separate models, or humans evaluate; the generator never grades its own work (section 2.6), and policy — encoded as versioned, testable rules rather than prompt text [18] — decides whether the evidence satisfies the profile [4][5].

## 5. Risk-tiered autonomy

Autonomy should expand by proving which classes of action can be delegated under which controls — never by declaring a system autonomous.

- **Tier 1 — reversible, low-impact** (documentation, generated indexes, formatting, research): broad automation once evidence is complete.
- **Tier 2 — bounded engineering changes** (small code changes, dependency updates, internal refactors, test generation): deterministic gates, human review at selected boundaries.

- **Tier 3 — consequential external actions** (production deployment, data deletion, paid media, customer communication, financial transactions): limited credentials, explicit approvals, idempotent execution, tested rollback.
- **Tier 4 — irreversible or safety-critical actions:** human-authorized, potentially with independent review, simulation, or formal analysis.

Tier boundaries are only credible when crossing them costs ceremony. A working example from this system: renaming the identity key stamped on every row of the Brain — part of the primary key in many tables — was executed not as one UPDATE statement but as forty-seven sequential full-table rebuilds (build, bulk-insert, atomic rename, checkpoint, verify), 110,811 rows re-keyed live with zero corruptions, precisely because an earlier in-place key update had corrupted the database. The ceremony is not overhead; it is why the system is still running. Promotion between tiers should likewise be earned with measured criteria — collision rates, stale-worker rates, verified success rates, human-intervention frequency, rollback effectiveness — recorded per activation, never granted because the system has been quiet lately.

Conexus's Autopilot path — the machinery that takes an approved requirements document through implementation to a verified green pipeline — is **built-but-gated**: it has run supervised live drills end to end, and its ruling-locked posture is fail-closed, with re-arming an explicit operator action. We consider that posture a feature. Systems should earn authority from operating history, and the earning process should be boring.

## 6. Business value I — cost advantage: local-first by architecture

The first cost advantage is structural, not negotiated: Conexus runs on infrastructure the operator already owns.

**Local-first execution.** The Brain is a local DuckDB file behind a local gateway. The verification engine runs on a local k3s cluster with a local Zot image registry — more than 69 deterministic jobs (approximately 125 job definitions at second edition) execute without purchasing cloud CI minutes. Local models (Ollama-hosted, e.g. a 27B instruct/coding model on the operator's GPU) perform bounded work — context compilation, drafting, log reduction, failure classification — at zero marginal provider cost. Cloud models are leased capacity for work that justifies frontier intelligence, not the default path for everything.

**Local compute does the context management.** The AOE-001 design rule is that the cloud model has working memory while Conexus has project memory. Deterministic reducers and local inference assemble, compress, and format context before a frontier model ever sees it. The operator's measured 30-day workload (2026-07-11 through 2026-08-09) shows why this matters: of 18,284,233,434 total tokens, 17,088,709,895 — 93.46% — were cache reads: history being re-

carried, not new work being done. List-priced, that window estimates at $13,910.58. Every token of context assembled or reduced locally is a token the frontier provider does not bill.

**SQL replaces grep.** Agents do not explore the codebase with `grep`, `sed`, and hopeful file reads. The code-intelligence index (Doxygen-ingested, 12,853 symbols for this project alone) answers "which file, which lines, what exact signature, which files depend on it" as one DuckDB query — milliseconds, with a result small enough to paste into a context packet. The same is true for institutional knowledge: PRDs, requirements, policies, decisions, and provenance records are tables and documents in the Brain, retrievable by SQL and hybrid search, instead of being re-derived from chat history or scattered across a wiki. The standing rule for agents is retrieval-first: symbol index and LSP for code, Brain query for knowledge, grep only as the declared fallback.

## 7. Business value II — the 70× cost reduction, decomposed

The headline figure is a **composite across the stack**, not a single benchmark. We present it as a product of independently anchored factors, each labeled by provenance — measured on the live system, specified in an implementation record, or modeled. Where a factor is modeled, the model is stated so it can be falsified against future telemetry.

Worked example: one bounded code-change task on this repository (~1M lines across the fleet) executed the naive way versus the Conexus way.

| # | Factor | Naive session | Conexus | Factor | Provenance |
|---|---|---|---|---|---|
| 1 | Find the code to change | grep/sed sweeps + reading candidate files into context: minutes to hours of session time; tens of thousands of tokens per exploration round | One DuckDB symbol query + LSP references: milliseconds; a few hundred tokens | **10–50×** on the navigation portion | Measured latency (ms vs min); token factor instrumented per lookup in the symbol-index skill |
| 2 | Assemble working context | Re-carry the growing transcript; major providers chop or compact the conversation mid-task, losing decisions | Local Context Compiler rebuilds a bounded packet from canonical state; local model drafts/distills it at zero marginal cost | **~2–3×** verified work per dollar | AOE-001 targets (G1 ≥2×, G2 3× stretch), set against the measured 93.46% cache-read share above |
| 3 | Knowledge access | Internal docs, PRDs, requirements, policies, and provenance re-explained or re-uploaded per session | Brain recall at session start: PRDs, decisions, policies, and evidence are queryable state with provenance controls | subsumed in factors 1–2 | Live (BRN-002 / BME) |

The composite: a navigation-heavy task at the 50× retrieval anchor, executed under context discipline worth ~1.4× on the remainder, lands at **≈70× cost reduction for that task class** (50 × 1.4 ≈ 70). The claim is deliberately narrow: it is the navigation-and-context portion of agentic work, which the operator's telemetry identifies as the dominant cost center (~92% of cost attributable to high-context sessions; operator baseline, being validated against AOE-001's per-work-item cost attribution). Tasks dominated by novel generation rather than navigation and context will see a smaller multiple. The controlling KPI is not the multiple — it is **verified work per dollar**, which the system now attributes per work item so this section can be audited against its own telemetry rather than believed.

## 8. Business value III — better and faster

Cost is the floor; control is the multiplier.

**Provenance and governance.** Every consequential action carries a chain from intent to outcome: requirement, work item, principal, agent session, model, files, commit, pipeline, evidence, policy decision, approval, deployment, operational result (sections 2.4–2.5). The temporal ledger answers "what did we believe when this was authorized" as a query. Governance is not a slide deck; it is a table you can audit.

**IAM-type policy, zero-trust internally.** Identity is separated the way mature IAM systems separate it: principal identity (who is operating), tenant/project identity (whose resources are in scope), agent/session identity (which worker is acting), work authority (which bounded task it owns), capability authority (which action classes it may perform), provider identity, and evidence identity. Policy is versioned code with persisted decisions [18], not prompt text. Network presence is not authority — a LAN connection grants no implicit trust, per zero-trust doctrine [5]. A worker may be able to reach a resource while lacking the policy authority to use it, and both facts are checked.

**Multi-tenant by design.** Tenancy is not a feature bolted on for the enterprise edition; it is the storage model. Every one of the Brain's 375 tables carries a `project_id`, every write crosses the single-writer gateway where identity and scope are enforced, and every retrieval is tenant-scoped by default — cross-tenant recall is a deliberate, policy-mediated act, never an accident of a shared database. Products, agents, pipelines, and dashboards for multiple businesses already share this one control plane today (the federation is live and operator-verified); the TEN-001 epic that hardens multi-tenant access for external principals is **in flight** — research complete, early phases operator-executable, 78% on the board at this edition, and this paper says so rather than rounding up. The consequence for buyers: tenant isolation is verified the same way code is — as evidence, per the six-state result model — not as a contractual promise.

**Production readiness, including disconnected operation.** The system remains fully operational if a model provider changes terms, the internet is unavailable, or the work must run in a closed environment: the Brain, gateway, coordination plane, verification engine, image registry, and CI runners are all local; local models continue bounded work; provider identity and limits are timestamped configuration, so substituting or removing a provider changes configuration rather than architecture. Frontier-model lanes degrade to local models; the control plane does not degrade at all. This is the same vendor-neutrality property that makes the system churn-resistant commercially — and it is demonstrated, not aspirational: the multi-vendor receipts in this paper were produced on exactly this local-first topology.

**Productivity, stated conservatively.** The operator's internal tracking (reports from January 2026 onward) measured an initial ~4,200× productivity gain for the best developer workflow versus the pre-AI baseline — a figure that held within 10% across four months — and, after the Conexus application itself was built and the workflow disciplined, now states the figure conservatively at ~2,000×. Provenance matters here: these are **operator-reported internal metrics**, not an independent audit, and this paper's own claims register treats them as such. The mechanism behind them is not mysterious: durable memory removes re-derivation, retrieval-first removes exploration waste, parallel workers multiply throughput, and evidence-gating removes the review bottleneck that otherwise eats the gains.

**Parallel model execution without interference.** Logical swarm size is independent of any provider's concurrency envelope: Kimi, Claude, Grok, Codex, and custom local models work the same board simultaneously, separated by file claims, work leases, and fencing tokens rather than by courtesy. The July receipt stands: 227 mainline commits across four days, 104 on the busiest day, zero reverts, zero collisions, three vendors in one repository. A second, fresher receipt: on 2026-08-12 a single orchestrating session (a) scanned the 266-card work board, (b) ran two explorer agents in parallel to map two code seams, (c) had a local 27B model draft the requirements document from their reports, (d) ran two coder agents in parallel on disjoint workstreams implementing four stories, and (e) independently re-ran the full verification suite — 11,271 tests green — before the work was accepted. No session's context held the project; the Brain did.

**Who this serves.** The same architecture prices in two directions. For a **small startup or solo operator**, the value is leverage without payroll or cloud burn: one person directs a multi-vendor agent workforce that remembers everything, verifies its own work against deterministic gates, and runs on owned hardware — the local-first topology means the marginal cost of another worker is electricity, and the offline/closed-environment posture means the company is never one vendor-policy change away from a stalled roadmap. For a **large organization**, the value is the control plane itself: multi-tenant isolation with per-tenant authority and audit, IAM-grade identity separation, policy as versioned code, provenance chains from intent to deployment, and evidence bundles that map directly onto SSDF-style assurance expectations [21][22] — the properties that

let a platform team raise autonomy tiers for hundreds of developers without losing the ability to answer, months later, exactly what was done, by which worker, under whose authority, and with what proof. The small operator buys throughput; the enterprise buys governed throughput; both are the same system, because the controls were built for the hard case first.

## 9. Roadmap

Ten directions, labeled honestly. Each carries binary acceptance criteria in the product PRD, because a roadmap item without a falsifiable exit condition is marketing. Status updated 2026-08-12 against the live work board:

1. Unified identity contracts across tools, agents, runners, work, and evidence (planned).
2. Temporal operational ledger: authority classes, evidence links, as-of-time queries (**live** — BRN-002 shipped 2026-08-07; planned at first edition).
3. Verified agent capability profiles: route work by demonstrated, verified capability (implemented with EGA-001; in final review — 83% on the board, not yet closed).
4. Language-agnostic verification: one canonical evidence envelope per ecosystem — Node/TypeScript strongest today, Java and C++ components built, Python/Go/Rust specified but not built (in flight / planned).
5. Autopilot supervised staging with measurable entry and exit criteria (built-but-gated; continuation phases in flight at 74%).
6. Layered Git authority earned per task class (planned).
7. Repair-loop maturation: taxonomy, calibrated confidence, independent re-verification (live loop; maturation planned).
8. Closed business-action loops with idempotent execution and outcome learning (in flight).
9. Enterprise trust surfaces: RBAC, tenancy, recovery, audit export (in flight — multi-tenant Brain access, TEN-001, is 78% on the board: research complete, early phases operator-executable today).
10. Agent-network governance above MCP/A2A: membership, delegation, revocation, replay (planned).

**Nearing completion at this edition** (review column of the work board, 2026-08-12): the per-file complexity ratchet (CPX-001, 93%), the content-calendar/social-posts merge (88%), evidence-gated autonomy closeout (EGA-001, 83%), and operator-managed art types (ART-001 E03d, 82%). Shipped today: the ASP-001 closeout — narration overlay on the composer video stitcher and lane-aware OAuth token storage — which took the parent epic from 96% to closed, executed end-to-end by the parallel-session pattern described in section 8.

## 10. Limitations and status

Conexus is pre-release and founder-operated; it has not been independently audited, and no claim here should be read as certification or compliance — frameworks are guidance, a control is implemented behavior, evidence proves the behavior occurred [21][22], and certification is a separate determination. The Brain is single-writer by design; multi-writer operation is unsolved here. Verification depth is uneven across languages (section 9, item 4). Autopilot remains fail-closed by ruling. The 70× figure in section 7 is a composite model anchored to measured factors, not a single benchmark; the productivity multipliers in section 8 are operator-reported internal metrics. The remaining receipts quoted in this paper (commit counts, test counts, index sizes, table counts, token telemetry) were verified against the live system, git history, and the work board on 2026-08-12 and will drift; the claims register, not this paper, is their source of truth. Where the board and this narrative disagreed during re-verification, the board won (section 9, items 3 and 9). Finally, the multi-vendor coordination result is one system's operating evidence, not a controlled study — we publish it as a receipt, not a benchmark.

## References

All URLs below were fetched and verified live on 2026-07-23; second-edition additions were verified 2026-08-10 through 2026-08-12. The full list with the specific claim each source supports is published at conexusbrain.com/references.html.

[1] Cerf, V., and Kahn, R. "A Protocol for Packet Network Intercommunication." IEEE Transactions on Communications 22(5), 1974. https://doi.org/10.1109/TCOM.1974.1092259 [2] Model Context Protocol — specification (revision 2025-11-25). https://modelcontextprotocol.io/specification/latest [3] Agent2Agent (A2A) Protocol — Linux Foundation project, contributed by Google. https://a2a-protocol.org/latest/ [4] Saltzer, J., and Schroeder, M. "The Protection of Information in Computer Systems." Proceedings of the IEEE 63(9), 1975. https://web.mit.edu/Saltzer/www/publications/protection/ [5] NIST SP 800-207, "Zero Trust Architecture," 2020. https://csrc.nist.gov/pubs/sp/800/207/final [6] Fowler, M. "Event Sourcing." https://martinfowler.com/eaaDev/EventSourcing.html [7] Fowler, M. "CQRS." https://martinfowler.com/bliki/CQRS.html [8] Gray, C., and Cheriton, D. "Leases: An Efficient Fault-Tolerant Mechanism for Distributed File Cache Consistency." SOSP, 1989. https://doi.org/10.1145/74850.74870 [9] Kleppmann, M. "How to do distributed locking." 2016. https://martin.kleppmann.com/2016/02/08/how-to-do-distributed-locking.html [10] Stripe API documentation: "Idempotent requests." https://docs.stripe.com/api/idempotent_requests [11] Richardson, C. "Pattern: Transactional outbox." https://microservices.io/patterns/data/transactional-outbox.html [12] Garcia-Molina, H., and Salem, K. "Sagas." SIGMOD, 1987. https://doi.org/10.1145/38713.38742 [13] Kubernetes

documentation: "Controllers." https://kubernetes.io/docs/concepts/architecture/controller/ [14] OpenTelemetry documentation. https://opentelemetry.io/docs/ [15] SLSA — Supply-chain Levels for Software Artifacts, specification v1.2. https://slsa.dev/spec/v1.2/ [16] in-toto attestation framework (CNCF). https://in-toto.io/ [17] Sigstore and Cosign. https://www.sigstore.dev/ [18] Open Policy Agent documentation. https://www.openpolicyagent.org/docs/latest/ [19] DORA research: "Trunk-based development." https://dora.dev/capabilities/trunk-based-development/ [20] Strathern, M. "'Improving ratings': audit in the British University system." European Review 5(3), 1997. https://doi.org/10.1017/S1062798700002660 [21] NIST AI 100-1, "AI Risk Management Framework (AI RMF 1.0)," and NIST AI 600-1, "Generative AI Profile." https://www.nist.gov/itl/ai-risk-management-framework [22] NIST SP 800-218, "Secure Software Development Framework (SSDF) v1.1" (https://csrc.nist.gov/pubs/sp/800/218/final), and SP 800-218A, the generative-AI SSDF Community Profile, final 2024 (https://csrc.nist.gov/pubs/sp/800/218/a/final) [23] Beck, M. "On the Hourglass Model." Communications of the ACM 62(7), 2019. https://doi.org/10.1145/3274770 [24] Moonshot AI / Kimi. "Kimi Code membership guide" and "Model Configuration." https://www.kimi.com/help/kimi-code/membership-guide — provider quota/concurrency facts cited as timestamped configuration, second edition.

Internal implementation records cited by this edition (case-study evidence, not external research): BRN-002 temporal operational ledger (implemented 2026-08-07); EGA-001 evidence-gated autonomy (implemented 2026-08-07, in final review); AOE-001 agent context and inference efficiency (implemented 2026-08-10); TEN-001 multi-tenant Brain access (research complete, in flight); internal usage telemetry 2026-07-11 through 2026-08-09; operator productivity tracking reports, January 2026 onward (operator-reported).