

Justin Sublette

AI Systems Engineer - Agents, Context, Execution and Evaluation

j.sublette@vmosaic.com · 812-374-4565

Naples, Florida, USA · Willing to relocate

Technical CV · Updated September 23, 2026

Open source: [Mathematical Research](#) · [Protocol Runner](#)

Professional profile

AI systems engineer building practical applications, research environments and the infrastructure that enables AI agents to develop and operate them. Particular depth in context construction, persistent execution, agent tooling and evaluation. Owns problem definition, architecture, integration, experimental direction and acceptance of results, working extensively with AI agents on implementation, investigation and testing. Experience connects product and domain decisions with source access, authentication, data, deployment, recovery and evidence of useful behavior.

Engineering experience

Founder and AI Systems Engineer - Vector Mosaic Inc. · 2025-present

Founded and own Vector Mosaic, a software company developing applications, research environments and agent infrastructure. Responsibilities span product conception, system design, development direction, integration, testing and operating environments. The projects below describe distinct systems within this body of work, with their own outcomes and evidence boundaries.

Began practical software development in June 2025 without prior programming or Git experience. Early automated-trading work introduced broker integration, strategy and risk interfaces, persistence, deployment and behavior tests. Subsequent work expanded into clinical software, personal-context applications, agent development infrastructure and autonomous research environments. The companion professional narrative documents this progression and the extensive role of AI collaboration.

Technical foundation

Technologies and methods used across the projects below:

- **Languages and applications:** TypeScript, Python, PowerShell, Node.js, React and Express.
- **Data and interfaces:** PostgreSQL, SQLite, versioned records, content-addressed storage, HTTP/REST, OpenAPI, MCP, WebMCP and OAuth/OIDC with PKCE.
- **Agent systems and context:** persistent workflows, multi-agent coordination, source and Git inspection, semantic and keyword retrieval, dependency analysis, indexing and context construction.
- **Evaluation and automation:** comparative agent experiments, protected reference answers, held-out tasks, failure diagnosis, adversarial integration tests, Playwright, browser debugging protocols and Windows UI Automation.
- **Operations and deployment:** Windows and Linux, containers, Git worktrees, authenticated worker services, process ownership and recovery, AWS, Railway and Hetzner.

Model-related experience centers on integrating existing models, investigating their behavior and evaluating the systems around them. Hands-on ML model construction and training remain a development area.

Selected technical work

Protocol Runner - Reliable execution for complex agent workflows

Creator and principal designer · 2026-present

Created Protocol Runner to carry complex AI engineering work through long, multi-stage plans while preserving the intended procedure and completed work. Pursued the problem over months of phone/Discord relay and worker-orchestration experiments, then developed a persistent execution system in response to recurring failures in plan adherence, handoffs, and completion reporting.

Led architecture, development, integration, and evaluation in close collaboration with Codex, which contributed extensively to implementation, investigation, and testing. Designed the software to retain plans and execution state outside conversations, deliver each step's prompt, and use reported step status to choose the next action allowed by the plan. Sent context-dependent steps to the same ongoing Codex conversation so the agent could build on the context already available there, and launched separate Codex workers for assignments that could proceed independently. Recorded distinct attempts and their evidence. Kept judgment of work quality and final acceptance with the coordinating agent under my direction. After a run used the wrong checkout, directed a permanent fix binding worker execution to the launching worktree; a fresh run then completed that assignment.

Used the system hundreds of times for repository development and investigation. A retained census identified more than 4,400 Codex worker tasks matching Protocol Runner's prompt signature, corroborating that sustained use.

Applied Runner to a historical update for DRCE, my engineering-evidence system: 25 investigation assignments covering 595 commits produced a draft with 40 accounts of significant engineering developments. Separate AI review identified three accounts needing correction; revised those while preserving the other 37 and the completed investigations. A fresh AI review passed the full corrected draft, after which the coordinator accepted it into DRCE's internal record. Runner carried the investigation, review, and correction workflow; DRCE preserved and released the accepted record.

Released the actual standalone implementation under MIT as [Protocol Runner](#), including its persistent API, executor, dashboard and optional Desktop integration. The [public recovery demonstration](#) uses simulated workers with the real API and store to preserve completed output and retry only the failed item. Separate real Codex CLI checks passed one-worker completion and explicit cancellation on Windows using a dedicated Codex home and workspace-write. The original Desktop/Discord bridge created real threads for serial work before codex exec was fully released for that workflow; direct CLI execution became the cleaner primary path once available. The [engineering account](#) explains the decisions, failure history and contribution.

Technical foundation: TypeScript, Node.js, SQLite, Python, Codex Desktop and CLI, Git worktrees; standalone API/executor and optional Windows Desktop integration.

Companion case study: [The System I Would Trust to Demonstrate Live](#).

CTX Engine - Context infrastructure and evaluation for AI coding agents

Creator and technical lead · 2025-2026

Created CTX Engine to help AI coding assistants find and use the code and documentation needed for a development task. Developed both the context infrastructure and comparative evaluations of whether it helped agents produce better answers. Led architecture, integration, experimental direction, and adoption decisions in collaboration with Codex, which contributed extensively to implementation and testing.

Combined keyword search, semantic search, and dependency analysis behind a command-line tool: an assistant submitted a task question and received a reading guide with relevant code and documentation, related components, missing information, and verification steps. Returned structured recommendations about whether to answer, investigate further, or report conflicting evidence, for the assistant to use while inspecting the sources.

Directed version-aware indexing and synchronization of edited files to keep search results aligned with changing code. When a rebuild unexpectedly returned nothing, recovery preserved the previous searchable data and explicitly identified it as out of date.

In an internal benchmark of 30 code-navigation tasks drawn from the project's own codebase, the combined approach found **69% of the reference set of relevant files, compared with 34% for a keyword-search baseline; 85% of its selections were relevant.**

Built evaluation harnesses that compared agent answers with ordinary repository-search controls and separated file-finding accuracy, answer quality, verification, and time/token measurements. Scoring checked required facts, unsupported claims, and acknowledgment of missing evidence. Protected reference answers from trial agents; for the sealed holdout, froze source, model, prompt, and scoring conditions. Excluded two holdout attempts invalidated by execution or output-contract problems, repaired the run conditions, and completed a clean rerun.

Used discrepancies to improve the experiments themselves. When an experimental context format retrieved more relevant evidence without improving answers, investigated whether agents failed to use that evidence, the extra material was unhelpful, or the scorer obscured meaningful differences. Added diagnostics for evidence supplied by the system, files opened by the agent, and the sequence in which useful evidence was reached. Kept missing timing and token data explicit rather than inferring efficiency gains.

In a separate comparison across 46 tasks from the project's own codebase, the standard configuration recorded nine wins, one loss, and 36 ties or mixed results against repository search under coarse answer-quality scoring. An experimental replacement that had performed well during development then did not demonstrate better answers on 24 held-out tasks from the same repository. Retained the standard configuration, using the evaluation to decide what to adopt and what required further investigation.

Technical foundation: Python, semantic and keyword search, dependency graphs, Git-aware indexing, agent evaluation harnesses, controlled experiments, failure analysis.

Life Links - Private physical-world context for AI agents

Founder and technical lead · 2026-present

Created Life Links, an owner-curated web application that keeps recorded context private within Life Links until its owner explicitly authorizes access. It preserves selected context about physical places, containers, objects, equipment, preferences, and past experience for use by compatible general-purpose AI clients; the deployed checks below cover Codex and ordinary ChatGPT. As founder and technical lead, owned product definition, the domain and authority model, integration, evaluation design, and release acceptance. Set the acceptance criteria, directed and reviewed the work, and decided what shipped; Codex contributed extensively to implementation, investigation, testing, and release execution. Line-level human/agent authorship was not tracked.

Designed each Life Link as a stable record for a physical place, container, or object, organized in a recursive hierarchy and optionally connected to a QR code. Kept purpose-based Collections separate from physical placement, so the same item can support several activities without acquiring conflicting locations. Kept the human UI and separate agent-protocol adapters over the same core, persistence, application queries and commands, and controller behavior so they apply the same ownership, revision, and privacy rules.

Extended the product with document and media attachments whose agent-readable derivatives remain linked to the exact source-attachment bytes and behind authorization checks; Routines that keep editable future plans separate from completed execution Sessions and append later corrections without rewriting the completed Session; and a Calendar where Life Links owns native events, Google and Outlook remain sources of truth for their events, and Routine schedules own read-only projections. For provider writes, used stable command identities, revision guards, provider readback, and guarded recovery after uncertain responses, with explicit per-Calendar agent permission.

Designed and integrated complementary agent interfaces: page-bound WebMCP for an authenticated open workspace and OAuth/Streamable HTTP MCP for later conversations while the site is closed. Both reuse the same application queries and commands. The remote boundary uses PKCE, rotating refresh tokens, revocation, scoped permissions, and stable-command replay and recovery. For destructive changes, it binds confirmation to the exact target, revision, and previewed effects before execution.

During the same candidate-led evaluation, Codex also served as an actual client in bounded release-acceptance workflows against deployed revisions with every Life Links page closed. It created, retrieved, and edited private records across conversations; started and finalized a Routine Run, then revised its future plan while preserving the completed execution Session; and passed destructive confirmation and revocation/reconnect checks. As a separate client, ordinary ChatGPT created, read, and edited across conversations. It canceled one deletion preview without changing the item, confirmed a fresh exact Life Link deletion, replayed that same preview to receive the stored result without another delete, and verified absence through the deployed MCP app. Through deployed Life Links Calendar and agent paths, bounded live round trips against designated Google and Outlook test accounts exercised create, read, update, delete, retry, permission, disconnect, and reconnect flows for exact synthetic events. They exposed defects in host metadata limits, post-write interpretation, and time-zone preservation, which were repaired before the affected live checks passed on corrected deployed revisions.

Redesigned Collection loading around paged records and snapshot-consistent membership context. In a matched browser benchmark on an unchanged 48-member Collection, startup API requests fell from 55 to 7. Across four desktop and same-host mobile-width conditions - desktop cold, desktop repeat, mobile-width cold, and mobile-width repeat - with three observations each, browser-observed median time to the first rendered Collection content fell 43-57%. The sanitized standalone public source contained the deployed application, passed clean-clone build checks, and was released under MIT; monorepo-only development and release

material remained private, and Railway deployed exact checked revisions from that public release line. Submitted the project to the OpenAI WebMCP Challenge.

Technical foundation: TypeScript, React, Node.js, Express, PostgreSQL, REST/OpenAPI, WebMCP, Streamable HTTP MCP, MCP Apps, OAuth/OIDC with PKCE, Playwright, Railway, Google Calendar API, Microsoft Graph, and QR codes.

Pre-Note Delta - Clinical chart review, agent tooling and deployment engineering

Creator and technical lead · 2026

Created an Epic-integrated clinical application around a precise question: what changed in this patient's chart since this clinician last saw them? Led product definition, architecture, implementation direction, integration and evaluation with extensive AI assistance. The work spanned the application, custom tools that made authenticated vendor information accessible to coding agents, browser and desktop test automation, evaluation of generated clinical prose, and an AWS deployment and hospital adoption path.

Built a TypeScript/React application using SMART on FHIR, the authorization and clinical-data standards used to launch applications within an electronic health record. Resolved the clinician's relevant prior encounter, then gathered intervening activity across the patient's chart, including other clinicians' work. Developed twenty deterministic summary sections with source evidence, explicit empty/error states and clinician-readable copy output. Added manual time horizons and a current-admission mode. Kept the evidence-bearing summary separate from optional AI drafting, with source inspection and clinician review; the application did not autonomously write to the chart.

Engineered around real Epic interoperability differences: clinician and practitioner-role identity, encounter timing, advertised versus observed API support, resource-specific query behavior, pagination, bounded retries and replay. Added progressive section delivery and on-demand evidence retrieval. In a recorded sixteen-patient sandbox cohort, all 3,599 evidence references loaded; eighteen of twenty sections had positive examples in at least five patients. Those results established bounded integration and evidence-navigation behavior, with remaining resource-coverage gaps kept explicit.

Built the information-access tooling my coding agents needed when they could not directly use my authenticated Epic developer browser. Progressed from DOM extraction of a difficult API-selection interface to authenticated browser capture and a searchable local documentation library. Preserved raw responses alongside readable material and tracked selected APIs, rendered guidance and unresolved captures. An April library collection contained 191 of 192 selected API documents, six guidance pages and 59 of 60 broader reference pages, giving agents concrete vendor contracts to implement against.

Developed test harnesses for Epic's browser-based LaunchPad and Hyperspace desktop environment. Combined scripted navigation with an optional AI pilot using screenshots, UI state and bounded actions, then checked the resulting application behavior separately. Made deterministic navigation the bulk default and reserved model-driven control for diagnosis and recovery. After shared replay recordings made patient provenance unreliable in an initial test corpus, introduced isolated patient-evidence collection, stable identities, content hashes and observed browser requests. The strict recollection ledger recorded 1,733 successful collections and eleven unresolved failures. Collection success remained distinct from eligibility for controlled scenarios comparing expected baseline selection with actual launches and AI drafts.

Evaluated AI inside the application as well as using AI to build and operate tests. A 96-case matrix crossed scenario, baseline and drafting mode, distinguishing working streams from adequate inputs and faithful prose: 53 passed the quality audit, 42 were input-limited and one contained corrupted output. Source-level diagnostics then exposed a digest-and-merge pipeline that timed out or failed to use available detail. Replaced it with bounded direct source packets; a twelve-case rerun completed every output capture: four drafts

incorporated additional source detail, six remained effectively summary-only and two lacked sufficient source material. Later browser-faithful reruns passed eight behavior scenarios, while the broader quality audit retained its separate scope.

Implemented dedicated AWS customer-environment templates, private application and data services, container rollout, controlled migrations, signed sessions and customer-specific authorization for optional AI. Separated operational logs from encrypted audit and deployment evidence with six-year S3 Object Lock retention. Recorded vendor demonstration deployment proofs showed two web and two API tasks stable, healthy public responses and a known audit event delivered to durable storage. Prepared hospital/Epic review materials, support procedures and a bounded pilot package. This was demonstrated vendor infrastructure and commercial preparation, without an established hospital rollout or commercial success.

Developed the concept without knowing Epic had a comparable native offering coming. I later regarded the close fit of Epic's offering to the workflow I had targeted and its physician reception as strong validation of the product insight. I chose not to pursue the remaining competitive opportunity once convinced that fighting the incumbent was not the best use of my effort. The system was retired into historical reference in July 2026.

Technical foundation: TypeScript, React/Vite, Node.js, SMART on FHIR, OAuth/PKCE, Playwright/CDP, Python, AI APIs, Redis, PostgreSQL, Docker, Terraform, AWS ECS/RDS/S3/KMS/WAF and Railway.

Mathematical Research - Durable AI research toward the Riemann Hypothesis

Creator and technical lead · 2026-present

Created a research system for pursuing a checkable proof or disproof of the Riemann Hypothesis (RH) with autonomous AI. The engineering problem is sustaining creative mathematical work across many conversations and processes while retaining exact assumptions, useful partial results and failed approaches. RH remains open. Led system definition, architecture, integration, evaluation and acceptance with extensive Codex contributions to implementation, investigation and testing. The research executive is itself an AI agent responsible for choosing and revising mathematical strategies; my role is not routine human proof adjudication. Line-level human/agent authorship was not tracked.

Designed a durable research objective that survives replacement of its executive agent. Separated the objective, current strategy, exploratory branches, accumulated scientific understanding, evidence and proposed results into explicit revisioned records. A successor reconstructs the current research position from those records and a checkpoint instead of depending on an immortal conversation. The host controls process lifecycle and containment; it does not choose mathematics. A single SQLite writer, immutable revision identities and content-addressed source material preserve the relationship between authored work, execution and recovery.

Built the research process around falsification and exact scope. Tracked claims and dependencies to prevent an unsupported prerequisite from silently becoming support for a later theorem. Preserved distinctions between a counterexample to a specific method, an inconclusive attempt and a useful conditional lemma. Early research campaigns separated derivation, dependency review and adversarial falsification before exchanging answers. One campaign's audit rejected an overbroad obstruction claim because its proposed class had not been declared before the search, while retaining the narrower local lemma. Designed separate paths for preserving a purported complete result, reviewing its exact statement and relied-on evidence, deciding whether to admit it, and applying an authorized change to the mathematical record. Model agreement and passing software tests do not establish theorem truth.

Redesigned research context after an expanding history reconstruction became an unsuitable default input. Current strategy selects attention; a separate cumulative scientific document preserves broader understanding, qualifications and reusable connections. The executive receives selected treatments with their restrictions and can retrieve exact history when needed. A material discovery or scoped failure prompts consideration of

consequences elsewhere in the research, even before a missing historical connection is suspected. Integrated authenticated indexed retrieval and an optional atomic publication of one evidence result with its already-authored scientific continuation, so an interruption cannot preserve the result while losing that paired continuation. Exact-request replay recovers committed effects without rerunning completed research.

Qualified reproducible execution of one frozen finite mathematical-software suite across three fresh sequential remote replicas, each approximately 104 minutes, with matching semantic results and unchanged mathematical proof state. Separately, an isolated Linux staging rehearsal using retained research data reconstructed the workspace, made a write, committed and read a checkpoint, and reached a stopped successor boundary with zero model-provider calls. Built a shared immutable artifact format and verified three selected Host/bridge tests against the same exported build bytes without rebuilding or selecting a release. These are bounded software and continuity results, not a scientific success claim.

Used actual retained data to challenge migration and retrieval beyond synthetic fixtures. The later storage-migration attempts exposed failures in historical provenance and scope handling; separate diagnosis identified defects in historical readers and redundant indexing work. Repaired those boundaries and passed focused native regressions. Preparation of the retained-data test workspace still failed before checks intended to read stored evidence and compare its linked source bytes with the authenticated originals. At the September 16 cutoff, full qualification of the newer software build and restoration of autonomous scientific behavior remained unfinished. This distinction keeps implemented mechanisms, successful bounded checks and unresolved research capability visible together.

On September 23, released the actual research core, Mission Host and execution adapters under MIT as [Mathematical Research](#). An isolated Linux installation with fresh public state passed initialization, real research-tool interaction and orderly stop/restart with retained state. Published [four reproducible research examples](#) covering a missing-assumption correction, a scoped matrix obstruction, a wider construction after a restricted approach failed, and a bounded Jensen error-budget improvement. In the [live finite-free example](#), a distinct successor retrieved exact retained Evidence and Candidate revisions, preserved the corrected assumptions and tightened a localization constant from 10 to less than 9. This was reference-assisted continuation; the other three examples use checkers and authored store walkthroughs. These public results establish their stated behaviors, not completion of the separate private retained-history restoration, a general research success rate or an RH solution. The [engineering narrative](#) connects the design decisions, tradeoffs and failure repair to the implementation.

Technical foundation: Python, TypeScript, Node.js, SQLite, content-addressed storage, cryptographic commitments, Git, MCP, Codex App Server, Linux/systemd, SSH, and Hetzner.

Workstation Control - Operating environments and tools for AI agents

Creator and technical lead · 2026-present

Created Workstation Control to give AI coding agents practical ways to operate across my Windows workstations, worker machines, browsers and local services. The problem extended beyond generating code: an agent needed to discover available capabilities, install configuration, reach the correct application session and inspect the result of its actions. Led system design, integration, evaluation and development with extensive Codex contributions to implementation and testing. Separated machine-level execution from the systems deciding which development or research tasks to pursue.

Built authenticated worker services for configuration delivery, credential installation and remote file inspection, with explicit profiles describing each host's role and capabilities. File delivery used atomic replacement, recognized repeated identical content and required explicit overwrite for conflicting content. Credential installation targeted registered destinations and keys, preserved unrelated settings and returned fingerprints

instead of secret values. Automated tests inspected resulting files and checked replay, overwrite conflicts, authentication and secret disclosure in responses and receipts.

Developed browser-session management that binds a workflow to a browser, debugging port and profile. Refused conflicting ownership and required exactly one matching managed browser process before reusing a responding endpoint. This addressed a practical automation failure: finding an open browser or responsive port does not establish that it belongs to the intended task. Added tests for mismatched processes and competing sessions.

Implemented Windows startup and maintenance around the difference between background services and interactive applications. Reconstructed the intended service-user environment for boot-time services while keeping browser and desktop automation dependent on an actual logged-in desktop. Coordinated reboot preparation so health inspection remained available while services refused new ordinary work. Moved stateful agent smoke tests into separate workspaces after early tests created synthetic conversations in normal working state. Repeated reboot recovery across machines still required live qualification.

Built a bridge to the Codex command-line app server and investigated its relationship to the open desktop application. Retained experiments showed that a prompt could persist in shared conversation state without becoming visible in the desktop. Developed a separate delivery path through Windows UI Automation, with explicit reports when focus was required or an action was unsupported. Later live tests using Protocol Runner, my workflow orchestrator, delivered prompts through exact visible conversation selection and received worker start and completion reports. Seamless synchronization of an already-open desktop conversation remained unqualified.

Extended the work through related repository-wide tool-interface engineering. An audit exposed inconsistent commands, rejected flag placement and tools whose first invocation did not explain safe use. Repaired native tool entrypoints and introduced a non-mutating discovery command with machine-readable purpose, commands, selectors, examples, effects and error meanings. Standardized structured results and made shared options work before or after subcommands. Introduced validation in report-only mode, then enforced the contract as tools were repaired. The closing June audit covered **81 stable tools with zero interface findings or probe failures**, establishing a checked interface baseline for agent invocation. This measured command discovery and interface conformance; it did not establish every operation's correctness or a general productivity gain.

Technical foundation: TypeScript, Node.js, Python, PowerShell, Windows runtime management and scheduled tasks, HTTP/JSON-RPC, Codex App Server, Windows UI Automation, browser debugging protocols, authenticated local APIs and private networking.

Repo Reader MCP - Authenticated source and history access for AI agents

Creator and technical lead · 2026-present

Created Repo Reader so approved AI clients could inspect current files and Git history in my development environment through the Model Context Protocol (MCP), which exposes operations an assistant can call. Built direct, on-demand source access around a single-owner authentication model and explicit read-only boundaries. Led architecture, development, integration and evaluation with extensive Codex contributions to implementation and testing. Established the service independently from my context-retrieval system, keeping repository inspection a distinct capability.

Implemented directory browsing, file and text search, ranged reads and content hashes, then added Git log, commit inspection, scoped diffs, file history, historical file reads, line provenance and history statistics. Current-file reads reflect the permitted working tree; historical references resolve to immutable commits before use. Restricted Git execution to fixed read operations with path checks, output limits, deadlines and

cancellation. When Git rejected the differently owned managed checkout, repaired the shared adapter with exact-repository trust applied to each command, preserving global Git configuration and distinguishing unavailable history from a missing reference.

Engineered durable OAuth authorization after the initial in-memory provider lost registrations and tokens on restart. Persisted client registrations, provider bindings and token-family relationships through serialized atomic updates, storing credential hashes rather than raw tokens or client secrets. Rejected corrupt or mismatched state and stopped authorization after persistence failure. Rotating refresh tokens detected reuse and revoked the affected family; wrong-client requests could not revoke another client's family. Added exclusive writer ownership and process-creation identity so a reused process ID could not be mistaken for the original state owner.

Built automated tests for server restart, interrupted authorization, token rotation and replay, concurrent refresh, competing writers and failed persistence. This test coverage targets the local authorization implementation; end-to-end continuity through an authenticated ChatGPT client remained a separate qualification gap in the retained record.

Expanded access to sibling worktrees and large development artifacts by separating what an agent may read from what ordinary searches enumerate. Configured directories could contain generated files that remained available through exact reads or explicitly targeted searches without entering the global search catalog. Credential paths, raw Git metadata files and unsafe paths remained denied. Verified Git repositories and linked worktrees independently from directory scopes. Streamed large files in bounded ranges and computed whole-file hashes separately without loading complete files into memory.

Corrected capped searches that repeated expensive authorization checks across an entire cached catalog. Selected candidates first, then applied mandatory live path checks in bounded batches. Built regression tests that counted the work performed, rather than treating elapsed time as the correctness test. Retained large-tree evidence supported structural isolation, while failed timing envelopes left latency qualification incomplete.

Extended the authenticated service with a read-only observation gateway for research-system evidence. Kept three stable operations for discovering available queries, issuing a query and reading its content, with a runtime catalog restricted to implemented source adapters. Bound paged content access to the authorized client, grant family, exact source identity and originally observed extent; expired or invalidated access could not silently return newer content. Source systems retained their data meaning and control authority. Real-client acceptance of this newer gateway remained separate from implementation and focused verification.

Technical foundation: TypeScript, Node.js, MCP SDK and Streamable HTTP, OAuth/PKCE, Git, filesystem security, atomic persistence, process ownership, bounded asynchronous I/O and adversarial integration tests.

DRCE - Traceable engineering evidence for people and AI agents

Creator and technical lead · 2026-present

Created the Development Record and Candidate Evidence system (DRCE) to preserve an evidence-grounded account of my engineering work for private reflection and analysis, and to let people and their AI agents evaluate selected material at the depth I choose. Led design, integration, and qualification in collaboration with Codex, connecting historical investigation, evidence management, secure delivery, and practical agent access.

Built a pipeline from one Git-managed record into reproducible, versioned releases containing readable documents, structured records, and links from claims to supporting evidence. Chose one source for both representations to prevent separately edited human and machine accounts from drifting apart. Built website, read-only HTTP, and Model Context Protocol (MCP) interfaces, with a complete resource inventory and reading guidance so agents could find and interpret the available evidence.

Separated private authoring from independently deployed public and restricted-access services. Hosted services receive only selected, frozen releases, with no live access to the development repository. Kept login identity separate from application permissions, binding restricted access to the organization, authorized application client, and permitted release, with download permission controlled separately.

Qualified hosted agent access using synthetic material through ChatGPT and a Codex test harness. Exercised all six retrieval tools, including search, document retrieval, and evidence tracing, and verified that removing permission blocked subsequent access.

Used DRCE with Protocol Runner to add 40 evidence-backed accounts of significant engineering developments, including decisions, corrections, and results. Runner coordinated AI investigation and review; DRCE retained the earlier accepted history by preserving its source files byte-for-byte while compiling the expanded record into a reproducible release, with verified cloud archive readback and local-backup recovery. Real career-material publication remains pending.

Technical foundation: TypeScript, Node.js, React, Zod, Git, REST, MCP, OAuth/WorkOS, Railway, AWS S3/KMS.

Earlier experience

Manufacturing and pipeline construction · 2012-2018

Worked in forging-plant production and mainline pipeline and station construction, including shift-team leadership and field-coordination responsibilities.

Independent market research and trading · 2018-2025

Studied market structure, risk and execution before beginning software development in 2025.

Education and credentials

Cisco Certified Network Associate (CCNA) - certified April 7, 2025; valid through April 7, 2028.

Indiana University, SPEA - policy analysis studies, 2013-Spring 2015; left without completing a degree.

Columbus North High School, Columbus, Indiana - Academic Honors diploma.

Supporting narratives

- Professional narrative - Building with AI, and building the systems around it
- Protocol Runner - The System I Would Trust to Demonstrate Live
- CTX Engine - Building context for coding agents and testing whether it helps
- Pre-Note Delta - Building the clinical product and the tools needed to build it
- Mathematical Research - Sustaining autonomous research across conversations