TECHNICAL PORTFOLIO · 2026

Connor O'Dea

Technical Portfolio

PREPARED FOR
UC BERKELEY

Selected AI systems, software architecture, and engineering projects. A curated look at how I identify real problems, decompose them, design the systems, and build them into software that runs.

WEB	connorodea.com
GITHUB	github.com/connorodea
EMAIL	connor@odeaworks.com
LINKEDIN	linkedin.com/in/connorpatrickodea
BASED	Denver, Colorado

CONNORODEA.COM

GITHUB.COM/CONNORODEA

Prepared for the University of California, Berkeley. Master of Information and Data Science.

02 · PROFESSIONAL AND TECHNICAL PROFILE

Applied AI engineer and systems architect

Denver, Colorado · builder of intelligent decision systems

I design and build AI-native systems end to end, from architecture through deployment. My work sits where messy real-world information meets software: I take unstructured inputs, warehouse floors, litigation documents, county property records, camera feeds, and turn them into structured representations that support decisions.

I am most drawn to systems rather than isolated models. The interesting engineering is usually in the connective tissue: the event backbone that keeps thirty services consistent, the governed state machine that decides what a system is allowed to believe, the retrieval layer that degrades instead of failing, the audit trail that makes an automated decision defensible. Retrieval, agents, and probabilistic reasoning show up in my products because they make the product work better, not because they are on a roadmap.

I work across domains on purpose. A problem in litigation looks a lot like one in property data, and a grading decision on a warehouse conveyor is the same shape as a routing decision in a document pipeline. Recognizing those shared structures is what lets me move quickly without cutting corners on data models, testing, or deployment.

Technology map

VERIFIED FROM REPOSITORY EVIDENCE

LANGUAGES

Python · TypeScript · JavaScript · SQL · Rust · Swift · Bash

FRONTEND

React · Next.js · Tailwind CSS · Vite · react-three-fiber · shadcn / Radix · Tauri

AI AND MACHINE LEARNING

LLM orchestration · LangGraph agents · RAG · embeddings · knowledge graphs · YOLOv11 · SAM 2.1 · GroundingDINO · OCR / barcode · Monte Carlo / ISMCTS · Bayesian modeling · game theory · Z3 SMT · multi-provider routing

BACKEND AND SERVICES

Node.js · Express · FastAPI · Hono · Celery · WebSocket / ws · REST APIs · API gateway · PM2 workers

DATA AND DATABASES

PostgreSQL · PostGIS · Redis Streams · Neo4j · Qdrant · OpenSearch · Cloudflare R2 / D1 · event sourcing · ETL / adapters

INFRASTRUCTURE AND TOOLING

GitHub Actions CI/CD · Turborepo + pnpm · Modal serverless GPU · Cloudflare Workers · Nginx · systemd · Hetzner VPS · MCP servers · Vitest / Jest / pytest

HOW TO READ THIS PORTFOLIO

Every technical claim is grounded in source code, configuration, or documentation from the actual repositories. Components are labeled by status: **implemented**, **partial**, **prototype**, or **planned**. Where a project advertises a capability its code does not yet support, this document says so. Planned work is never described as if it ships today.

03 · DISTRIBUTED SYSTEMS · EVENT-DRIVEN ARCHITECTURE ● CORE IMPLEMENTED · FRONTENDS PARTIAL

QuickWMS

A warehouse and reverse-logistics platform built as an event-driven service mesh.

PROBLEM

Reverse logistics is a coordination problem. A returned or liquidated item has to be received, identified, graded, placed, and then routed to the right outcome (resale, refurbishment, salvage, auction, or disposal), and every one of those steps is owned by a different team and a different system. When those systems drift out of sync, inventory lies, listings go stale, and items get lost. QuickWMS is my answer to keeping many specialized workflows consistent without collapsing them into one monolith.

SYSTEM RESPONSIBILITIES

- Ingest inbound shipments, purchase orders, and manifests
- Identify and condition-grade items, including a computer-vision path
- Maintain an authoritative stock ledger with reservations
- Route graded items to disposition flows
- Drive marketplace listing lifecycles and multi-channel sync
- Manage outbound orders, picking, packing, and shipping

INTENDED USERS

Warehouse operators at receiving and grading stations, inventory and listings managers, and administrators, each with a dedicated interface rather than one overloaded dashboard.

VERIFIED STACK

MONOREPO	Turborepo + pnpm workspaces, ~1,200 TypeScript source files
FRONTEND	11 Next.js 16 / React 19 apps, shared component library
SERVICES	35+ Express / TypeScript microservices
MESSAGING	Redis Streams event bus, saga orchestrator
DATA	PostgreSQL 16, 55 tables, raw-SQL migrations
VISION	Groq / Claude / OpenAI multi-provider router
GATEWAY	Express reverse proxy, JWT, Swagger / OpenAPI
DEPLOY	PM2 on a single VPS behind Nginx

DATA MODEL

A hand-written PostgreSQL layer (no ORM) of 55 tables across seven migrations. An `items` hub links to manifests, receiving sessions, suppliers, and pallets. A self-referential grading-history chain records condition over time. Separate sub-schemas cover consignment, multi-tenant third-party logistics, salvage and parts, analytics, and the event store itself.

WHAT THIS DEMONSTRATES

Domain-driven decomposition, relational data modeling at real scale, distributed-transaction thinking (sagas and compensation), and the discipline to keep dozens of independently deployed services eventually consistent through a typed event contract rather than shared database access.

INTEGRATION POINTS AND CONSTRAINTS

A central API gateway fronts the platform with Helmet, CORS, rate limiting, JWT auth, and generated OpenAPI docs, reverse-proxying path prefixes to ten backend services. Listing services synchronize to external marketplaces (Shopify, eBay, Amazon). The computer-vision service enriches item identity against a third-party UPC and EAN lookup API. The main architectural constraint is deliberate: services never reach into each other's tables. They communicate only through the event bus and the gateway, which is what makes the topology safe to grow.

04 · QUICKWMS SYSTEM ARCHITECTURE

Five layers, one event backbone

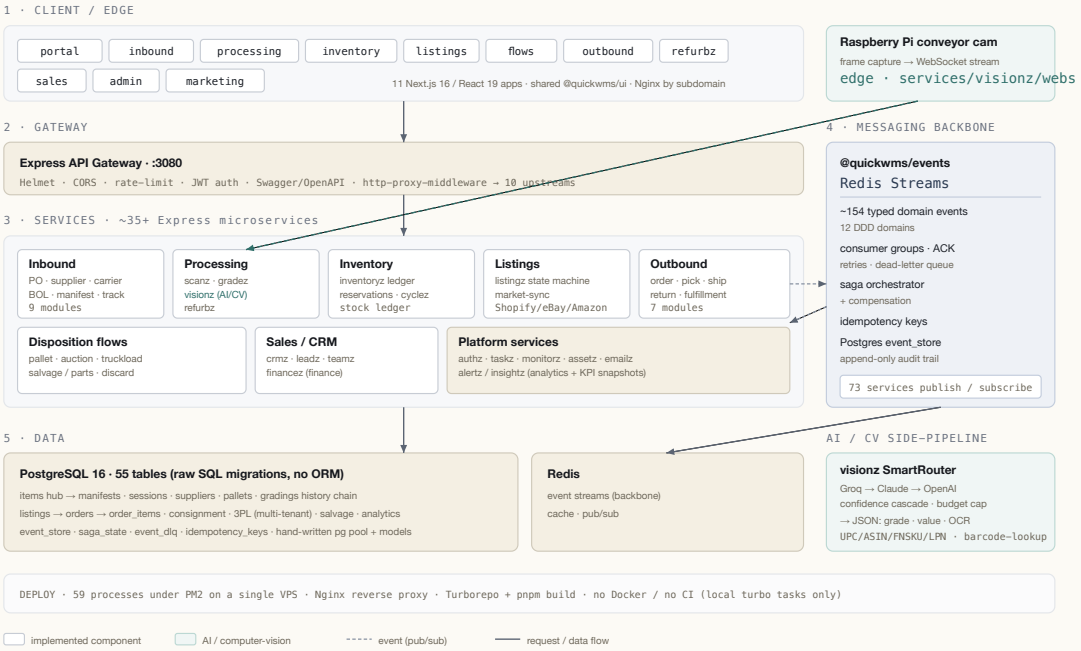


Fig 4.1 · Service topology. 11 client apps and a Raspberry Pi camera feed a gateway that fronts 35+ domain services, all coordinated by the Redis Streams event framework and backed by PostgreSQL and Redis.

KEY DESIGN DECISIONS

- **Events over shared state.** ~154 typed domain events across 12 business domains, published and consumed by 73 service files, with consumer groups, dead-letter queues, and idempotency keys.
- **Sagas for multi-service work.** A saga orchestrator coordinates transactions that span services and compensates on failure, instead of distributed locks.
- **Audit by construction.** Every event is appended to a PostgreSQL event store, so the system's history is queryable, not reconstructed from logs.
- **Cost-aware vision.** The grading service cascades Groq, then Claude, then OpenAI by confidence under a daily budget cap.

ARCHITECTURAL TRADEOFFS

- A hand-written SQL layer trades ORM convenience for explicit control of indexes, upserts, and migration ordering.
- A single-VPS PM2 deployment (59 processes) is operationally simple but is not yet containerized or wired to CI. Both are noted as planned, not claimed as done.
- Frontends are real but partial (four to nine routes each); the backend is the mature half of the system.

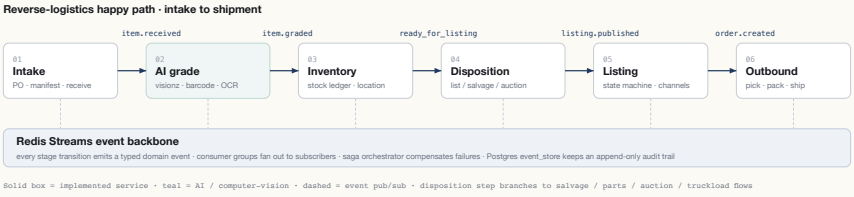


Fig 4.2 · End-to-end reverse-logistics path; each stage transition emits an event.

05 · QUICKWMS END-TO-END PIPELINE

From supplier intake to customer, on one event backbone

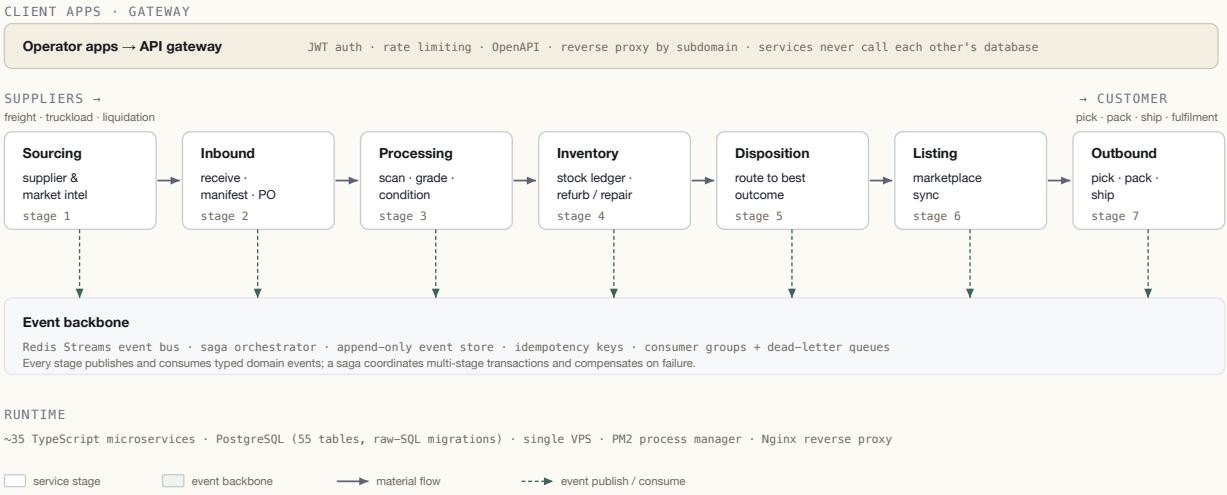


Fig 5.1 · The reverse-logistics pipeline. Seven staged services carry an item from supplier intake to customer fulfilment; coordination happens entirely through a typed event backbone rather than shared database access, so any stage can be deployed and reasoned about on its own.

WHAT THE PIPELINE GUARANTEES

- A single authoritative stock ledger. Inventory is the source of truth; every other stage reads and writes it only through events, never by reaching into its tables.
- Traceability by construction. Because state changes are appended to an event store, an item's full history is queryable rather than reconstructed from logs.
- Failure isolation. A saga orchestrator spans multi-stage transactions and compensates on failure, so one stage stalling does not corrupt the others.

WHY IT IS BUILT THIS WAY

- Each stage is owned by a different team and interface in the real operation, so the system has to keep many specialized workflows consistent without collapsing them into one monolith.
- An event contract between services is safer to grow than shared database access: new stages subscribe to the stream instead of taking new table dependencies.
- The deployment is deliberately modest — one VPS, no container platform — and the document does not claim scale it has not run.

06 · AGENTIC AI · KNOWLEDGE GRAPHS · PROBABILISTIC REASONING ● CORE IMPLEMENTED

Juricratic

Litigation modeled as a dynamic information and decision system, not a single prediction.

PROBLEM FRAMING

Case strategy is usually reasoned from intuition and anecdote. Juricratic treats a matter as what it actually is: a multi-party game played over incomplete, evolving information, where each side has payoffs and moves. That framing turns strategy into something you can represent, reason over, and stress-test, provided the system is honest about what it knows and rigorous about what it is allowed to assert.

MATTER-STATE ARCHITECTURE

State is event-sourced: the next matter state is a pure transform of the current state and an incoming event, which makes every change a traceable delta. On top of that sits a governed **canonical matter state** that is fail-closed. A fact cannot be promoted into canonical belief unless it is sourced and attorney-reviewed. A hard firewall separates the live matter from simulation branches, so a hypothetical can never write itself back into what the system treats as true.

KNOWLEDGE GRAPH AND EVIDENCE

A typed, in-memory knowledge graph models parties, evidence, motions, authorities, and judges with bi-temporal edges that supersede rather than delete, preserving history. An optional Neo4j mirror is code-complete but runs in a degrade-not-fail mode when the store is not provisioned.

VERIFIED STACK

CORE	Python, event-sourced engine and solver
AGENTS	LangGraph StateGraph subgraphs
REASONING	Claude tool-use, constrained to legal action tokens
GRAPH	typed in-memory graph, optional Neo4j mirror
RETRIEVAL	feature-hash embeddings, optional legal 768-d + Qdrant / OpenSearch
VERIFY	Z3 SMT contradiction checking
SURFACE	FastAPI (~60 endpoints), Next.js 15 + react-three-fiber

PROBABILISTIC REASONING AND SIMULATION

The solver runs bounded simulation over settlement dynamics: seeded Monte Carlo rollouts, determinized information-set Monte Carlo tree search, pure-Nash equilibria, and a best-response **exploitability gap**. A transparent Bayesian case model with Kalman belief updates and Dirichlet shrinkage priors carries uncertainty explicitly. The design rule is stated in the code itself: this is simulation, not prediction, and it does not emit a calibrated win probability.

WHAT THIS DEMONSTRATES

Turning an ambiguous real-world domain into a governed formal system: event sourcing, a typed knowledge graph, game-theoretic and Bayesian modeling, agent orchestration held apart from a deterministic core, and human-in-the-loop review with an append-only audit trail. It is the clearest example of the kind of consequential decision system I want to keep building.

07 · JURICRATIC SYSTEM ARCHITECTURE

A governed core, an agent layer, a bounded solver

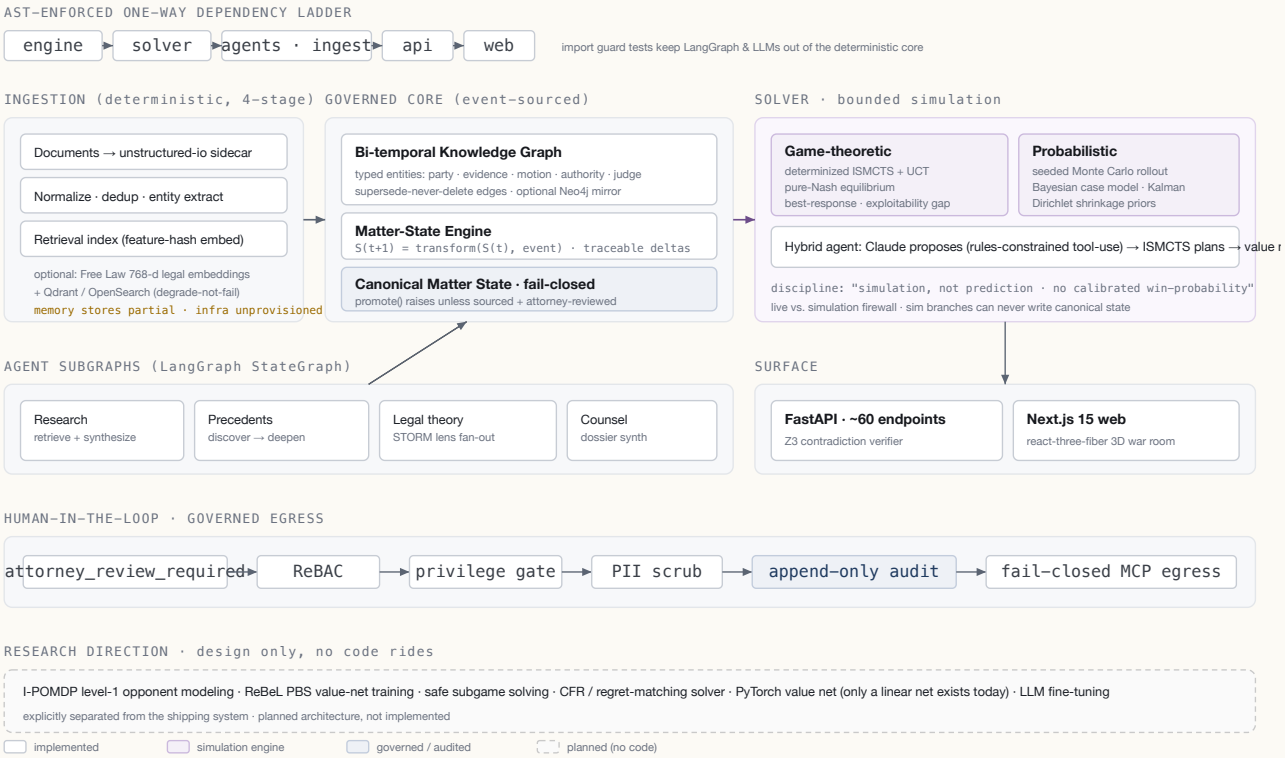


Fig 6.1 · Ingestion feeds a governed, event-sourced core (knowledge graph plus canonical matter state). Agent subgraphs and a game-theoretic solver read from it; nothing reaches a user without passing the human-review and audited-egress path. Planned research is shown separately and carries no running code.

DESIGN DECISIONS WORTH CALLING OUT

- **The core cannot import the agents.** An AST-enforced, one-way dependency ladder (engine, then solver, then agents and ingest, then api, then web) with import-guard tests keeps LangGraph and LLM calls out of the deterministic core.
- **Retrieval degrades, it does not fail.** Deterministic feature-hash embeddings work with no external services; real legal embeddings and vector stores layer on when available.
- **Egress is fail-closed.** Relationship-based access control, a privilege gate, PII scrubbing, and an append-only audit run on every outbound path.

STATUS, STATED HONESTLY

Implemented: the engine, solver, agent subgraphs, FastAPI surface, 3D interface, Z3 verifier, and a large test suite under a strict branch-coverage gate.

Partial: the external memory stores (Neo4j, Qdrant, OpenSearch) are code-complete but not yet provisioned; the app deploy is pending while the public landing site is live.

Research direction: opponent modeling with I-POMDPs, value-net training in the ReBeL style, and a regret-matching solver are documented as design only, with no code riding on them.

08 · COMPUTER VISION · EDGE INFERENCE · SERVERLESS GPU

● EDGE PIPELINE IMPLEMENTED

QuickVisionz

An edge computer-vision system that identifies and grades items on a conveyor, then prints a label.

PROBLEM AND CONTEXT

Identifying returned products by hand is slow and inconsistent. QuickVisionz runs a real-time vision pipeline on a Raspberry Pi at the point of capture, so an item is detected, read, graded, labeled, and registered before it leaves the station. Heavy segmentation work is offloaded to a serverless GPU that scales to zero when idle, which keeps an edge-first system affordable.

VERIFIED STACK

EDGE	Raspberry Pi 5, OpenCV, Python, systemd
DETECTION	YOLOv11-nano + ByteTrack tracking
PREPROCESS	Kornia GPU: CLAHE, denoise, normalize
READ	pyzbar barcode (7 variants), UPC lookup
FALLBACK	GPT-4.1-mini vision when no barcode resolves
SEGMENT	SAM 2.1 + GroundingDINO on Modal H100
OUTPUT	Zebra ZPL over TCP, FastAPI + WebSocket dashboard

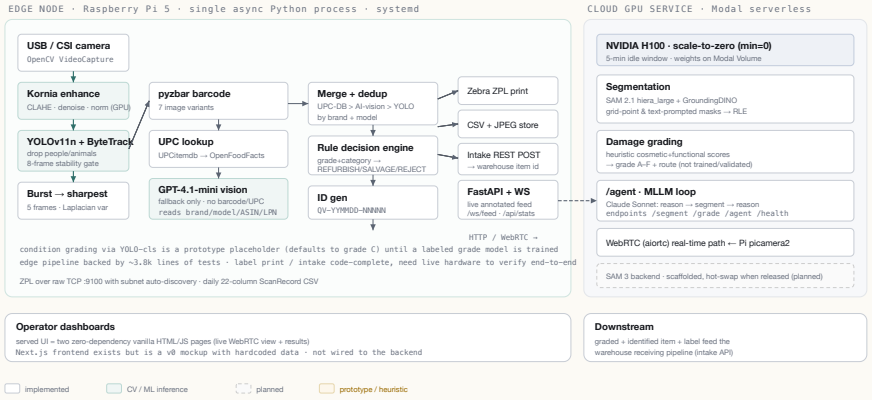


Fig 7.1 · The edge node runs the full detect-read-grade-label loop in one process; a serverless H100 handles segmentation and an MLLM reasoning agent. The live UIs are lightweight HTML dashboards.

WHAT THIS DEMONSTRATES

End-to-end computer vision under real constraints: detection with tracking and stability gating, multi-strategy barcode reading with an LLM vision fallback, a rule-based routing engine, raw-socket thermal printing, and a split between edge and elastic GPU inference. The edge pipeline carries roughly 3,800 lines of tests.

STATED HONESTLY

- Condition grading via the classifier is a prototype placeholder until a labeled grade model is trained; there are no model-accuracy metrics.
- The segmentation service loads SAM 2.1 today; a SAM 3 backend is scaffolded for a future swap, not running.
- Label printing, intake registration, and the WebRTC path are code-complete but unverified without live hardware.

09 · APPLIED DATA SCIENCE · DATA PIPELINES · MULTI-AGENT AI

● PRODUCTION-GRADE · IN SERVICE

AIWholesail

A real-estate data platform: multi-source ingestion, a scoring engine, and a research agent over a property graph.

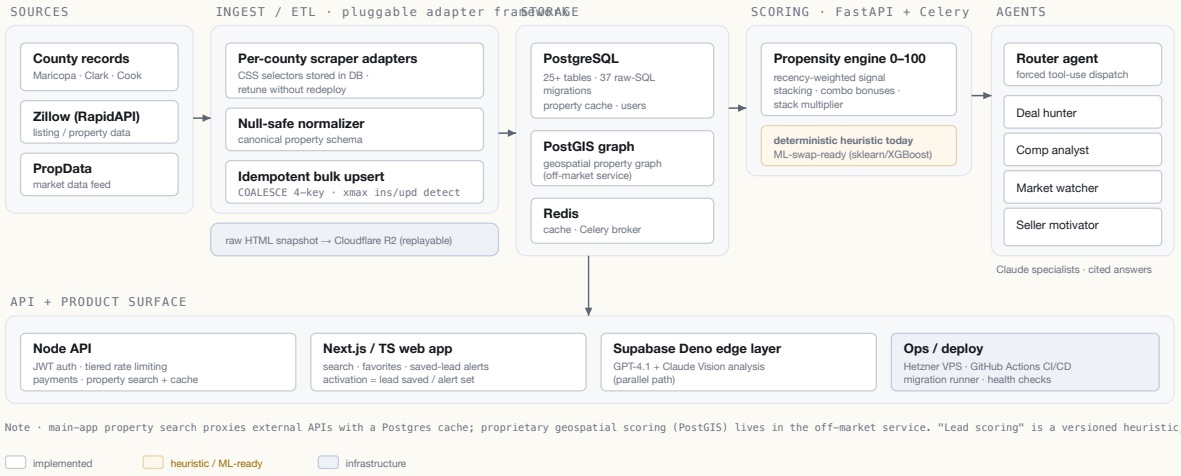


Fig 8.1 · County records, listing APIs, and market feeds flow through a pluggable adapter framework into PostgreSQL and a PostGIS graph, then feed a propensity-scoring service and a multi-agent research assistant.

THREE THINGS THAT STAND OUT

- **Multi-source ETL with a pluggable adapter framework.** Per-county foreclosure scrapers use CSS selectors stored in the database, so a county can be retuned without a redeploy. A null-safe normalizer and idempotent, conflict-preserving bulk upserts (four-part key, insert-versus-update detection) keep records clean; raw HTML is snapshotted to object storage so any run is replayable.
- **A signal-based propensity engine.** A 0 to 100 score built from recency-weighted signal stacking, combo bonuses, and a stack multiplier over a PostGIS property graph, explicitly versioned to be swapped for a trained model once enough labeled outcomes accrue.
- **A multi-agent research assistant.** A router agent uses forced tool-use to dispatch specialist subagents (deal hunter, comp analyst, market watcher, seller motivator) that return cited answers.

VERIFIED STACK

DATA	PostgreSQL, 25+ tables, 37 raw-SQL migrations; PostGIS
SERVICES	Node API + FastAPI / Celery scoring service
AI	Claude agents; GPT-4.1 and Claude Vision analysis layer
WEB	Next.js / TypeScript: search, favorites, alerts
OPS	Hetzner VPS, GitHub Actions CI/CD, JWT, rate limiting

STATED HONESTLY

The lead score is a deterministic weighted heuristic by design, not a trained model. Main-app property search proxies external APIs (Zillow, PropData) with a Postgres cache; the proprietary geospatial scoring lives in the off-market service.

WHAT THIS DEMONSTRATES

Applied data engineering across the full path: resilient ingestion of messy public data, careful schema and upsert design, transparent scoring that leaves a clean seam for a future ML model, and agentic AI wired into a product with citations and cost controls.

10 · ADDITIONAL SELECTED SYSTEMS

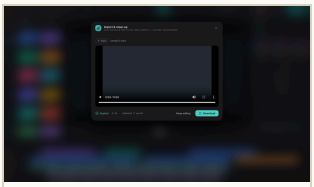
Three more systems worth a look

Cutroom

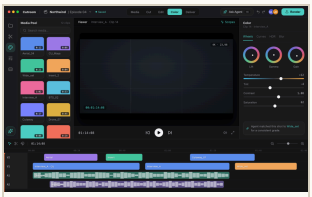
BUILDING · DISCIPLINED TDD

An AI-native video editor that cuts from the transcript.

Problem and contribution. Editors spend most of a session scrubbing for a moment they already remember. Cutroom transcribes first and makes the transcript the timeline: strike a sentence and the cut happens. It is built as a real non-linear editor, not a wrapper, with a media pool, multi-track timeline, color, and delivery. The engine is a worker of roughly forty FFmpeg operations (reframe, chromakey, transcription, text-to-speech, captions, color), each with its own test, driven by a Claude agent server that plans multi-step edits with forced tool-use and prompt caching, exposed over web, MCP, and CLI.



Transcript-driven timeline



Editor surface

STACK	TypeScript, React, Vite, Hono, FFmpeg, Claude · web / MCP / CLI
LINK	github.com/connorodea/cutroom

Sightline

BUILDING

Measures when AI assistants recommend a merchant's store.

Problem and contribution. Search is moving inside assistants, where there is no rank to check and no analytics to read. Sightline fans a set of tracked buyer queries across multiple AI answer engines on a schedule, with bounded concurrency and cost caps, detects whether and how a store surfaces, and computes a visibility score and share-of-voice from a clean rubric. It reads naturally as data science: the interesting work is metric design and scoring over noisy generative output.

STACK	Next.js, TypeScript, PostgreSQL · answer-engine query fan-out + scoring
STATUS	private repository; listed at connorodea.com/work

GeoStamp

LIVE

Batch-writes correct EXIF GPS to photos from a map pin.

Problem and contribution. Local businesses are told to geotag their photos and given no usable tool to do it. GeoStamp writes correct EXIF GPS across a batch on both desktop and web. It is a compact, shipped systems piece: a Rust core wrapped in a Tauri desktop app, chosen for correct low-level metadata handling and a small install footprint.

STACK	Rust, Tauri, React · desktop + web
LINK	github.com/connorodea/geostamp · geostamp.app

11 · DATA, ML, AND AI SYSTEMS CAPABILITY MAP

Capabilities, tied to the systems that prove them

Each capability points to the projects where it is actually implemented.

Data ingestion and ETL

Multi-source scrapers, adapter frameworks, manifest and record ingestion.

`AIWholesail · QuickWMS`

Data validation and modeling

Null-safe normalizers, idempotent upserts, large relational schemas.

`AIWholesail · QuickWMS`

Event-driven pipelines

Redis Streams bus, consumer groups, sagas, event sourcing.

`QuickWMS · Juricratic`

Knowledge graphs

Typed, bi-temporal entity and relation graphs with optional Neo4j.

`Juricratic`

Retrieval and embeddings

Feature-hash and legal embeddings, Qdrant / OpenSearch, degrade-not-fail.

`Juricratic · AIWholesail`

Agent orchestration

LangGraph subgraphs; router plus specialist agents with forced tool-use.

`Juricratic · AIWholesail · Cutroom`

Computer vision

YOLOv11 detection and tracking, SAM 2.1 segmentation, OCR and barcode.

`QuickVisionz · QuickWMS`

Model inference and routing

Multi-provider cost-aware cascades; serverless GPU with scale-to-zero.

`QuickWMS · QuickVisionz`

Probabilistic modeling

Monte Carlo, ISMCTS, Bayesian and Kalman updates, Dirichlet priors.

`Juricratic`

Simulation and game theory

Nash equilibria, best-response and exploitability over bounded branches.

`Juricratic`

Scoring and metric design

Signal-weighted propensity scores; answer-engine visibility scoring.

`AIWholesail · Sightline`

Human-in-the-loop and audit

Fail-closed review gates, append-only audit, governed egress.

`Juricratic · QuickWMS`

READING THE MAP

Breadth here is not a list of buzzwords: every capability is anchored to source in a specific repository. The concentration around ingestion, event-driven design, retrieval, agents, and probabilistic reasoning is the through-line of my work, and it is squarely the ground the MIDS program covers.

12 · SOFTWARE ARCHITECTURE AND INFRASTRUCTURE

How these systems are actually built and run

AREA	WHAT IS IMPLEMENTED	WHERE
API design	REST services behind a reverse-proxy gateway with JWT auth, rate limiting, and generated OpenAPI docs; ~60-endpoint FastAPI surface.	QuickWMS · Juricratic · AIWholesail
Service boundaries	Domain-driven services that communicate only through events and the gateway; an AST-enforced dependency ladder in the Python core.	QuickWMS · Juricratic
Data architecture	Hand-written PostgreSQL layers, 55 and 25+ table schemas, PostGIS geospatial graph, event store, bi-temporal graph.	QuickWMS · AIWholesail · Juricratic
Messaging and jobs	Redis Streams event bus with consumer groups, DLQ, and sagas; Celery workers; PM2-managed background processes.	QuickWMS · AIWholesail
Caching	Redis for cache and pub/sub; Postgres-backed API response caching for external data feeds.	QuickWMS · AIWholesail
Auth and access	JWT auth, tiered rate limiting; relationship-based access control with a privilege gate and PII scrubbing on egress.	AIWholesail · Juricratic
Deployment	GitHub Actions CI/CD to a Hetzner VPS; Turborepo + pnpm builds; systemd services on edge devices; Modal for serverless GPU.	AIWholesail · QuickWMS · QuickVisionz
Testing	Vitest, Jest, and pytest suites; per-operation tests in the video engine; a strict branch-coverage gate in the Python core.	Juricratic · Cutroom · QuickVisionz · QuickWMS
Observability and audit	Append-only event and audit stores; health-check and rollback scripts; monitoring services.	QuickWMS · Juricratic
Security posture	Fail-closed governed egress, secrets kept out of source, Helmet and CORS hardening at the gateway.	Juricratic · QuickWMS

AN HONEST NOTE ON SCALE

These systems run on modest, self-managed infrastructure (single VPS deployments, serverless GPU on demand). They are engineered for correctness, consistency, and auditability rather than for enterprise traffic, and this document does not claim production scale it cannot show. Containerization and CI for QuickWMS specifically are planned, not yet in place.

13 · CURATED PROJECT INDEX

Selected repositories

A curated subset chosen for technical depth and relevance, not a full repository dump. Public repositories are linked.

PROJECT	PURPOSE	LANGUAGES	STATUS	REPOSITORY
QuickWMS	Event-driven warehouse and reverse-logistics platform	TypeScript, SQL	Core built	private
Juricratic	Litigation modeled as a governed decision and simulation system	Python, TypeScript	Core built	private
QuickVisionz	Edge CV item identification and grading with GPU offload	Python	Edge built	private
AIWholesail	Real-estate data platform, ETL, scoring, research agents	TypeScript, Python, SQL	In service	`github.com/connorodea/aiwholesail`
Cutroom	Transcript-driven AI video editor with an agent engine	TypeScript	Building	`github.com/connorodea/cutroom`
Sightline	AI answer-engine visibility scoring for merchants	TypeScript	Building	private
GeoStamp	Batch EXIF GPS geotagging, desktop and web	Rust, TypeScript	Live	`github.com/connorodea/geostamp`
APIDistributed	Agent-native, edge-deployed API marketplace (MCP surface)	TypeScript	Building	private

ON ATTRIBUTION

This index lists only original work. Repositories that are unmodified forks or clones of third-party frameworks are deliberately excluded, so nothing here credits me with code I did not write.

Full public shipping log at connorodea.com. Private repositories are described in this document but not linked; they can be shared directly on request.

14 · ENGINEERING PHILOSOPHY

What I try to build, and why

I am drawn to systems where machine learning, probabilistic reasoning, distributed software, and human judgment meet. The work I care about ingests messy real-world information, turns it into a structured representation, and improves the quality of a decision through data, inference, and feedback.

I design systems, not isolated features. A model is rarely the hard part. The hard part is the pipeline that feeds it, the state it updates, the events that keep everything consistent, and the interface a person actually trusts. I spend my effort on that connective tissue.

I turn unstructured reality into structured representations. Warehouse items, legal documents, county records, and camera frames all start as noise. Giving them a schema, a graph, or an event stream is what makes downstream reasoning possible.

I connect machine learning to operational workflows. Inference is only useful when it moves through to a routed item, a published listing, a reviewed claim. I build the path from a prediction to an action.

I design for auditability and human review. Consequential systems should be able to explain what they believed and why. Append-only histories, fail-closed review gates, and governed state are defaults in my work, not afterthoughts.

I model uncertainty honestly. When a system cannot know something, it should say so. I would rather ship a calibrated simulation that refuses to fake a probability than a confident number with nothing behind it.

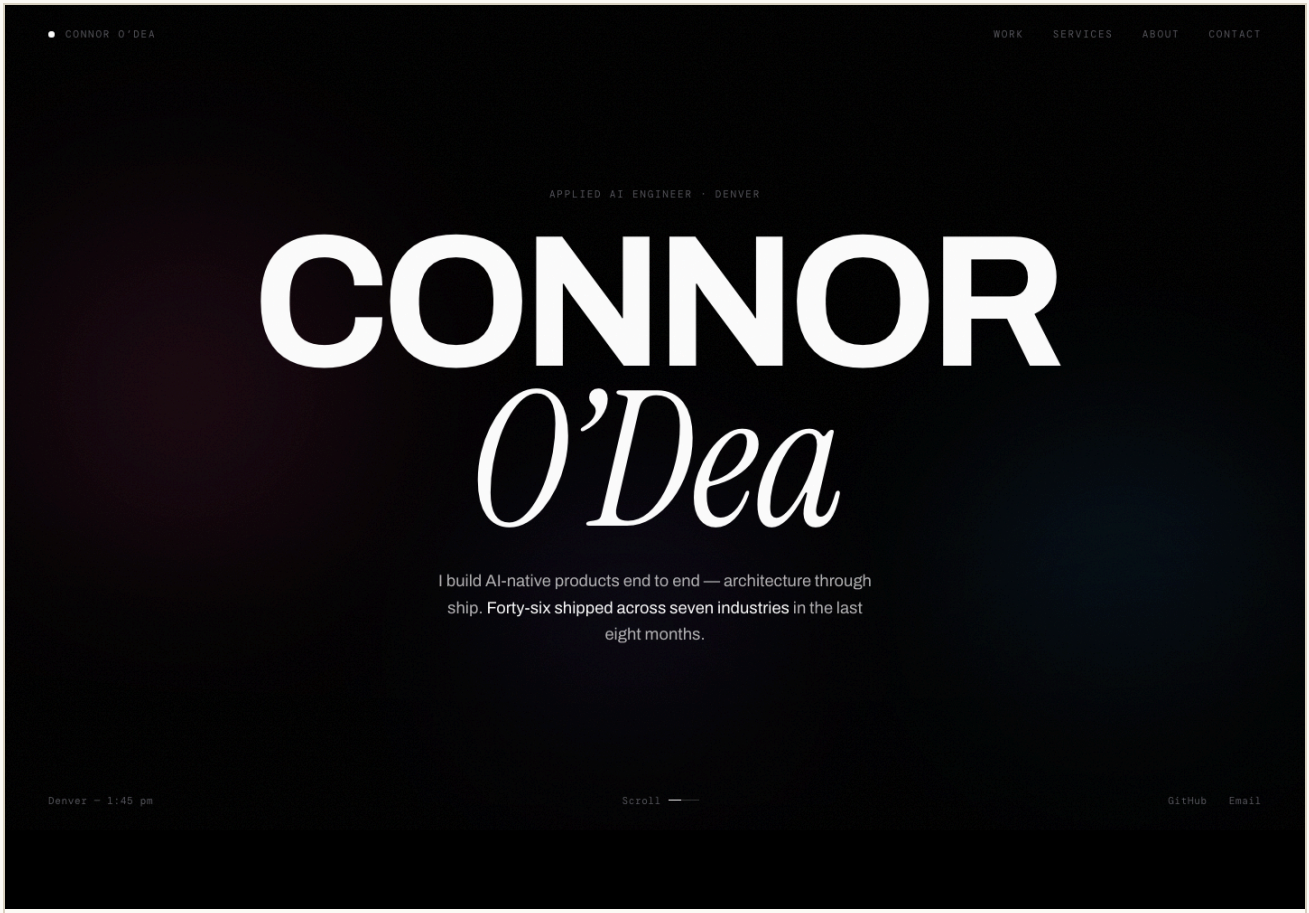
I balance experimentation with discipline. Tests, migrations, and deployment are how a prototype becomes software that runs. I move fast, but I leave a clean seam for the next model, the next service, and the next engineer.

WHY BERKELEY MIDS

I have learned this by building, across many domains, on real infrastructure. What I want next is the formal grounding to match the systems intuition: rigorous statistics and machine learning, causal and experimental methods, and the shared language of a field. That is the gap the program fills, and it is the direction the rest of my work is already pointed.

15 · WEBSITE AND CONTACT

Where to find the rest



connorodea.com · full public shipping log, case studies, and live project links

WEBSITE	connorodea.com
GITHUB	github.com/connorodea
EMAIL	connor@odeaworks.com
LINKEDIN	linkedin.com/in/connorpatrickodea
BASED	Denver, Colorado

CONNORODEA.COM

GITHUB.COM/CONNORODEA

Private repositories described here (QuickWMS, Juricratic, QuickVisionz, Sightline) can be shared directly with the admissions committee on request.

Thank you for reading. Everything in this portfolio is something I designed, built, and can walk through in depth. I would welcome the chance to do exactly that.