Cleo Portfolio

BioLinked — May 2021 – May 2022

BioLinked built a registry that connects blood institutes and medical researchers with people willing to participate in clinical research. I was the founding engineer and built the product from nothing.

Participant Portal

Started alone, grew to a team of 3

- The core of the product, the app that participants actually used. Someone could sign up, link their account to their blood institute, and answer research surveys, and their answers were stored so researchers could use them later. I built it from scratch as both a website and a phone app from a single codebase, and handled everything through to release, including shipping it to the app store and pushing updates.
- React Native Web on the front end, a .NET backend on Azure, deployed through Firebase.

Partner Portal

Started alone, grew to a team of 5

- The other side of the marketplace. Blood institute admins and researchers needed to actually find the participants who had signed up for their organization, so I built them a web app to search and explore that data. It let them build custom queries without writing any code, browse dashboards, and see a map of which counties their participants lived in.
- Built in Angular on the same backend as the Participant Portal.

Question Editor

Solo project

- Originally, changing the survey questions participants saw meant asking an engineer. I built a tool that let admins edit their organization's questions themselves, which opened up running targeted campaigns for specific kinds of research. Started as an Electron desktop app and became part of the Partner Portal.

Press Website

Solo project

- The public landing page for the company, built in React.

Amazon — June 2022 – April 2026

I worked on catalog quality, the systems that keep the information on Amazon's product listings accurate. Some context that makes the rest readable, every individual product on Amazon has an ID called an ASIN. Listings are classified by product type and by browse node, which is roughly the category the item shows up under. When one of those classifications is wrong or protected from edits, teams file appeals to get it fixed. Over my four years there, this work shifted from pipelines of human reviewers to LLMs making the decisions, and my projects trace that shift.

Task Tag Management System (TTMS)

Team of 3

- TTMS was the assembly line for appeals. It took in appeals about product classifications, bundled them into batches for human reviewers, then took the reviewers' answers and pushed the corrections back into the Amazon catalog. I maintained the system and shipped many features onto it over time, and it became the foundation several of my later projects plugged into.
- Java, built on DynamoDB, Athena over S3, and AWS state machines.

TTMS In Flight Cancellation

Solo, with a senior supervisor

- TTMS had no way to cancel an appeal once it was in motion. A cancellation system had been planned originally but was cut during the initial build. I brought it back, redesigned around everything we had learned from running TTMS in production.
- The hard part is that an appeal can be at any stage when someone cancels it, just received, mid batch, or already sitting in front of a human reviewer. I handled every case, including pulling back and reconciling work reviewers had already finished, and solved a timeout problem in the slowest stage by moving that step onto longer running infrastructure.

Annalog

Team of 2

- The website where the human review actually happened. Reviewers opened batches of appeals and worked through them one by one. My favorite design problem here was concurrency, many people needed to work the same batch at once without two people ever grabbing the same appeal, which we guaranteed with database level locking.
- React front end, Java backend, DynamoDB.

Guardian Appeals Dedupe *Solo, pitched and built end to end*

- This one started with me noticing a pattern nobody was acting on. On Amazon, every size and color of a product is its own listing, but they all belong to one family with a single parent, and the whole family inherits its category from that parent. Appeals resolve at the parent level. So when ten appeals came in for ten shirts from the same family, we were paying human reviewers to answer the same question ten times.
- I pitched the fix and built it, when an appeal arrives, look up its parent, remember that parent for a couple of weeks, and block any new appeal that shares it, while signaling back to the original appeal so the person who filed it knows why. The first version saved about $1 million per month in redundant review work.
- After demoing it to my director, I convinced leadership to give me one more week to turn it into a package any appeals team could drop into their own system. I also made adoption nearly effortless by handling the data access permissions centrally instead of making every team set that up themselves.
- It spread across multiple appeals teams, cut appeals to the Guardian system by 60%, and now saves Amazon $20 to $30 million per month. It was my first big demo to Amazon leadership, and the project that taught me how far a well packaged idea can travel inside a big company.

PT Appeals *Solo, with a senior supervisor*

- A new front door for product type appeals. It picked up the products under appeal from the internal ticketing system, gathered all the information a reviewer would need, sent everything through TTMS for correction, then wrote the results back onto the original ticket so the person who filed it could see the outcome.

BN Appeals *Team of 2*

- The same idea for category appeals, but supporting bulk submissions. People needed to appeal thousands of products at once, so we built Excel upload support, which meant both an ingestion and conversion pipeline on our side and working with external teams to allow spreadsheet uploads in the ticketing system at all.

AMS Migration *Solo*

- Amazon built a successor to our human review system, and I moved TTMS over to it, coordinating with the new team until we hit 100% and could retire Annalog.

Starfish Translation Orchestration *Solo*

- Starfish was the Ontology organizations move to have LLMs make catalog decisions instead of human reviewers. I found a place where it was wasting money. When Starfish had already generated text for a listing in one language, it would generate again from scratch for other languages. I added a bypass that translates the existing output instead, skipping the regeneration entirely. A small change on paper, but it touched pipelines owned by many teams, and I coordinated and implemented all of it myself.

Pearl *Started alone, grew into a team of 12*

- Pearl, the Prompt Experimentation and Research Lab, is my proudest work. It was my idea, I pitched it to leadership, I built it from zero, and it grew into its own team at Amazon. It moved me into a science org, and it was the first time my designs were shaping who the company hired.
- Starfish needs two prompts for every product type and attribute pair, one to fill in missing data and one to verify data is correct. Amazon has about 25,000 product types with roughly 10 attributes each. That is half a million prompts. No amount of careful hand tuning covers half a million prompts. It is a problem you can only solve with infrastructure.
- Pearl gave prompt writers a place to actually do science. It ranked all those prompts by how much room they had to improve, so writers always worked on what mattered most. When someone submitted a new prompt, Pearl ran it head to head against the current one on a live sample of 200 products, then 500, and only promoted it if it won both tests with statistical significance. The staged samples were deliberate, borrowed from classic ML benchmarking, to stop people from overfitting a prompt to one lucky batch.
- It also had a live panel where writers could test a prompt interactively against Starfish's production model and the newest Claude model available, so experimenting felt immediate instead of bureaucratic.
- Scaling it was its own project. Our original scoring provider could not keep up with demand, so we migrated scoring onto our own distributed fleet of machines. Vue.js front end, Python backend, DynamoDB throughout.

Supervisor Agent *Team of 4 engineers and a principal scientist*

- Once LLMs were making decisions and other LLMs were reviewing them, a new question appeared, what happens when they disagree and neither backs down? We built a third agent to settle it, a larger reasoning model with access to every tool it needed to investigate the dispute and make a final call.
- I designed the tools it could call. Years of loading and surfacing product data for appeals meant I knew exactly what information an arbiter would need and how to get it.

Clear Sight *Team of 2 engineers and a senior scientist, built in 3 days*

- A hackathon project that won and ended up demoed to Doug Herrington, the CEO of Amazon's stores business. Search for a backpack on Amazon and the title, images, and description all repeat the same keyword stuffed text. I think it is one of the biggest UX sins on the site. Clear Sight used LLMs to fix it.
- You entered any product and the app recreated its live listing. One button launched an agent that researched comparable listings and competitors, figured out what actually made this product unique, put that in the title, and stripped the repeated text from everything else. I owned the backend, working alongside a front end engineer and a scientist. Python, TypeScript, and React.

Bulk Prompt Evaluator *Solo*

- Pearl could originally only evaluate one prompt at a time, and I discovered users were already writing scripts to hammer the endpoint in loops. Rather than fight that, I built what they were telling us they needed, a bulk API that could take large volumes of prompts at once, built on queues and batching with a throughput dial we could configure per customer. I also wrote the onboarding process for teams that wanted access. This turned out to matter beyond convenience, because it made Pearl usable by other software, not just people.

Auto Prompt Generator *Solo*

- The natural next step after the bulk API, if machines can submit prompts, machines can write them. Every run, it pulled the 250 prompts in Starfish most in need of improvement, spun up a swarm of small models to draft candidate replacements, had a separate model critique the drafts, checked the survivors against trusted reference data, and pushed the winners through the Bulk Prompt Evaluator, where they faced the same statistical bar as any human's prompt.
- A multi agent, multi model orchestration system in Python, running on a schedule in Docker on EC2.

BYO ASIN Pearl *Team of 2*

- Pearl graded prompts against a curated list of products with known correct answers, but for some product types that list was small or low quality, which made scientists distrust their results. Bring Your Own ASINs let them supply their own test products and correct answers instead. Getting there meant coordinating with three other teams that owned the data pipelines involved.

Aurelian — March 2026 – April 2026

Aurelian builds AI for 911 call centers. AVA, an AI agent that handles non emergency calls on its own, and CORA, a heads up display that assists human call takers during live calls.

AVA Address Validation *Solo*

- Callers rarely give clean addresses. They say things like "the corner of 5th and Main" or "the Denny's by the highway," and AVA's address validation could not handle any of that, which meant those calls could not get a non emergency response dispatched. I added a new endpoint to the validation system and tested it against Google's Gemini and street data APIs, turning those fuzzy descriptions into exact locations in our system.

Route Ingestion Automation *Team of 2*

- A route is the script of questions an operator asks for a given scenario, like a noise complaint, and every police station has its own set, buried in PDF guideline documents. Onboarding a station meant transcribing those by hand. I built a system that uses an LLM to read the PDFs and produce structured routes ready to import into AVA or CORA, turning a manual onboarding chore into an automated step. Python, running in Docker, using Google Gemini.

CORA *Started with 1 engineer and a PM, grew to 2 engineers*

- A heads up display for 911 call takers, a transparent screen that listens to the live call and does the paperwork in real time. It fills in details as the caller says them, shows which scenario the call has become, and surfaces the next question the call taker should ask. When AVA hands a call over to a human, CORA arrives pre filled with everything the AI already learned, so the caller never has to repeat themselves.
- Python backend in Docker using Langfuse, TypeScript front end in an Electron wrapper.

CORA Live Translation *Solo*

- Made CORA work when the caller does not speak English. It detects the language, switches to a transcription model suited to it, and shows the call taker a live English translation. The call taker can type replies in English and CORA speaks them back to the caller in their own language, letting one operator handle a call they otherwise could not.

CORA Live Maps *Solo*

- The moment a location is identified on a call, a map of it pops up on the display, so the call taker sees where the incident is without leaving the conversation.