AI-Assisted Development: From Crisis to Automated Workflow

How a jail population dashboard migration became a blueprint for AI-powered research workflows

Richie Lin | Data Collaborative for Justice | July 2026

The NYC Jail Population Tracker

6,500+	10+ Years	Daily	5+
Daily Inmates Tracked	Historical Data	Automated Updates	Research Partners

What the Dashboard Does

- Pulls daily inmate custody data from NYC Open Data
- Aggregates by race, age, charge severity, custody level
- Runs SARIMAX forecasting models for 12-month projections
- Serves interactive Streamlit dashboard to researchers & policymakers

Tech Stack (V2)

- AWS Glue (ETL)
- AWS Lambda (forecasting)
- ECS Fargate (Streamlit)
- GitHub Actions (CI/CD)
- S3 + EventBridge (orchestra)

The Breaking Point

March 2026

Schema Failure

NYC Open Data introduced new charge categories. The ETL pipeline's hardcoded column mapping couldn't handle them. The forecasting model crashed because it expected fixed columns that no longer existed.

↳ *Dashboard went blank. Forecasts stopped updating.*

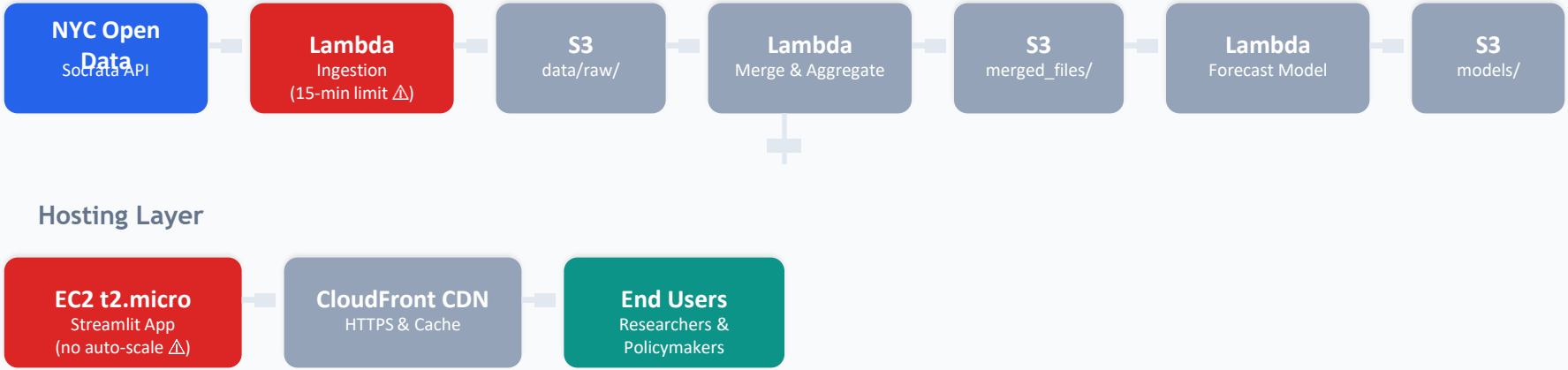
Compute Limits

Data growth pushed the daily ETL past Lambda's 15-minute timeout. Memory limits were exceeded. The legacy EC2 instance couldn't scale to handle peak traffic.

↳ *Pipeline failed silently. Users saw stale data.*

V1 Architecture: Legacy Pipeline

Lambda-based + EC2, all sharing single S3 namespace. Bottlenecks highlighted in red.

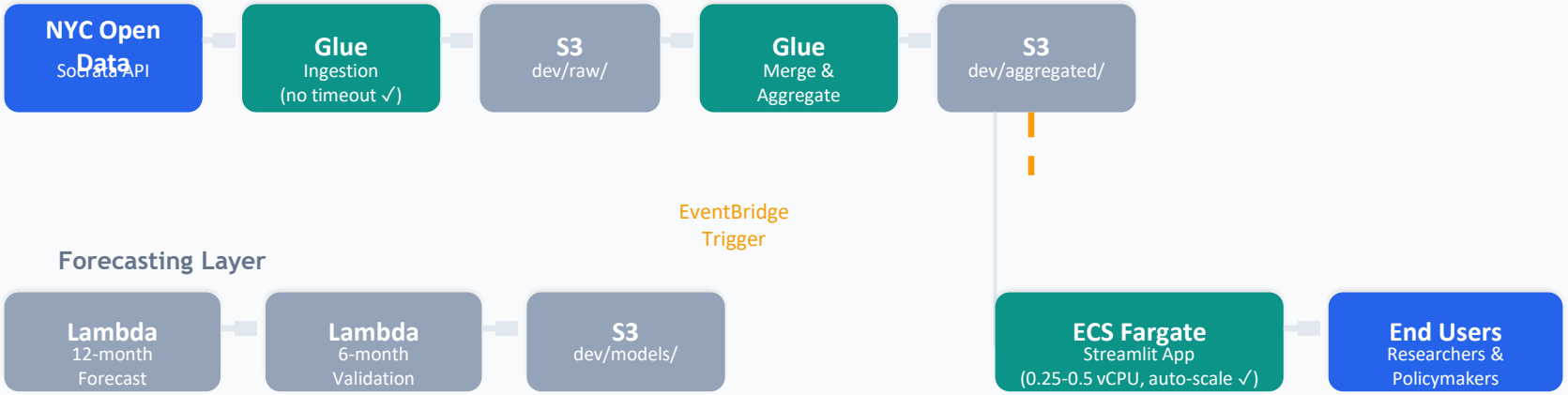


Critical Bottlenecks

Lambda 15-min timeout: ETL jobs hit the wall as data grew beyond 8M+ rows
Hardcoded schema: New charge categories crashed the pivot → forecast returned zeros
EC2 scaling: Single t2.micro instance, no auto-scaling, burst credits depleted under load

V2 Architecture: Glue + ECS Modernization

Serverless ETL + containerized frontend. Isolated dev/prod namespaces. Fully automated CI/CD.



Key Improvements

Glue: No timeout, scales to any data size. Dynamic schema — new charge categories handled gracefully.
Isolated environments: dev/ and prod/ S3 namespaces prevent V2 dev from affecting live dashboard.
CI/CD: Git push → GitHub Actions → auto-deploy. No manual SSH or script execution.
Cost: ~$11/mo dev, ~$18/mo prod. Down from ~$50+/mo V1 EC2 + operational overhead.

What Needed to Be Done (1/2)

Storage Isolation

New S3 namespace (JAIL_POP_DASHBOARD_V2/{dev,prod}/) without breaking the live V1 pipeline. Historical data traversal logic to read V1 data as read-only reference.

ETL Migration: Lambda → Glue

Rewrite ingestion and aggregation scripts for AWS Glue Python Shell. Dynamic pivot logic to handle any future schema changes. Automation script to sync scripts to S3 and provision Glue jobs.

Forecast Containerization

Dockerize SARIMAX models into Lambda containers. Upgrade Python 3.9→3.11. Pin pmdarima, statsmodels, scikit-learn versions. Isolate output paths to V2 dev namespace.

What Needed to Be Done (2/2)

Frontend: EC2 → ECS Fargate

Dockerize Streamlit app, create ECS task definition, provision security groups and ECR repository. Set up standard Fargate for static IP. Stress-test capacity sizing (0.25 vCPU dev, 0.5 vCPU prod).

Orchestration & Alerting

EventBridge rules to chain Glue → Forecast Lambdas. SNS topic for ETL failure alerts. IAM least-privilege: Glue, Lambda, ECS, and GitHub OIDC roles.

Production Cutover

DNS migration (GoDaddy → ALB). SEO injection into Streamlit container. Zero-downtime transition with V1 fallback running for 3-7 days. Decommission legacy EC2 + CloudFront.

The Traditional Path

Find the Right Person

Hire a DevOps engineer or cloud consultant who understands both AWS infrastructure AND the criminal justice data domain. This is a niche intersection — not many candidates.

Months of Coordination

The domain expert (you) and the DevOps consultant must align on architecture, data flows, security constraints, and deployment strategy. Every decision requires back-and-forth.

Knowledge Transfer Risk

The consultant builds the system, but their knowledge leaves when the contract ends. Documentation is often rushed or skipped. The next person to maintain it starts from scratch.

Cost: $15K - $50K+

Consulting rates for cloud migration range from $100-$250/hr. A project of this scope typically takes 2-4 months. Plus ongoing maintenance and handoff costs.

The AI-Assisted Path

Domain Expert Drives

You describe the problem in plain English. AI translates it into technical plans, code, and infrastructure. No intermediary, no lost context.

Weeks, Not Months

14 phases from initial setup to production cutover. Each phase: plan → implement → verify → document in a single AI session.

Auto-Documented Forever

34KB development memory + 12 implementation plans + 10 walkthroughs. Every decision and bug fix is permanently recorded in the repository.

Key insight: The human never wrote a single line of production code. The human's job was to review, verify, and make architectural decisions.

overhead. No consulting fees.

How AI Helped: Research & Analysis (1/2)

Codebase Understanding

AI analyzed the entire legacy codebase — Lambda functions, EC2 scripts, S3 structures, Jupyter notebooks — and mapped all active dependencies. It identified which scripts were in production, which were archived experiments, and which were dead code. This alone would take a human days.

Architecture Research

AI researched AWS Glue vs Lambda tradeoffs for this specific workload, ECS Fargate capacity sizing (0.25 vs 0.5 vCPU), EventBridge trigger patterns, GitHub OIDC authentication, and Docker multi-stage builds — all tailored to the existing AWS account's constraints.

How AI Helped: Research & Analysis (2/2)

Problem Diagnosis

When the treemap broke after stakeholder review, AI traced the bug through 4 layers: Streamlit rendering → CSV data → Glue ETL script → pd.get_dummies column naming. It designed a controlled experiment on S3 to confirm the root cause, then implemented both a fix and a self-healing detection mechanism.

External Knowledge Integration

AI incorporated best practices from AWS documentation, Streamlit deployment guides, Docker optimization patterns, and Python package compatibility matrices — without the human needing to search, read, and synthesize dozens of articles manually. It surfaced relevant solutions proactively.

How AI Helped: Planning & Orchestration

Implementation Plans (12 files)

Each phase started with a detailed plan: exact files, step-by-step commands, verification criteria. The plan served as both blueprint and contract — the AI executed against it, and the human reviewed against it.

Development Memory (34 KB)

A single master document tracked every decision, bug fix, deployment result, and architectural choice. When the AI lost context between sessions, this file restored it instantly — the AI's external brain.

Walkthroughs (10 files)

After each phase: a verification document confirming what was built, tested, and deferred. Stakeholders could audit progress without reading code or accessing AWS console.

How AI Helped: Coding & Auto-Deployment

What the AI Wrote

- Python ETL scripts for AWS Glue
- Forecast model Dockerfiles
- Infrastructure-as-Code (ECS, ECR, ALB)
- GitHub Actions CI/CD workflows
- Streamlit dashboard bug fixes
- SEO injection + DNS cutover scripts
- Deployment automation scripts

GitOps Auto-Deploy Flow

git push

→ GitHub Actions

→ OIDC auth (AWS)

→ Build Docker image

→ Push to ECR

→ Force Fargate redeploy

→ Live in minutes

Two branches → two environments:
v2_ecs_dev → dev | v2_ecs_prd → prod

Zero manual steps. Push code → everything updates automatically. Rollback is just git revert.

How AI Helped: The Memory System

AI coding agents have no long-term memory. Each session starts fresh. The solution: structured docs as external brain.

development_memory.md (34 KB)

The master record. Every phase completed, every bug fixed, every architectural decision, every S3 path and IAM role. When a new AI session started, reading this file gave it the full project history — like handing a new teammate a complete briefing.

Implementation Plans (12 files)

Blueprint for each phase: exact files to modify, step-by-step instructions, expected outcomes. The AI wrote these before executing — serving as both plan and contract. The human reviewed and approved before any code was written.

Walkthroughs (10 files)

Post-execution validation: what was built, what was tested, what was deferred. Stakeholders could verify progress without reading code or accessing AWS. These documents made the invisible visible.

The Human-AI Communication Pattern (1/2)

1

Grill Me

Before writing any code, the AI asks clarifying questions. What are the constraints? What can't change? What's the priority? Which AWS account and region? This prevents wasted work and catches assumptions early.

2

Plan

The AI writes a detailed implementation plan: exact files to change, step-by-step commands, expected outcomes. The plan becomes the contract — both sides agree on what success looks like before execution begins.

3

Execute

The AI implements the plan. For code: write tests first, verify they fail, implement, verify they pass. Every step is logged. The human watches and intervenes only when architectural decisions are needed.

The Human-AI Communication Pattern (2/2)

4

Verify

Human reviews the output. Is the treemap rendering? Do the forecasts produce real numbers? Are the S3 paths correct? The AI runs automated checks (syntax validation, file existence, API responses) — but the human does the final judgment.

5

Document

Update development_memory.md with everything that happened. Write a walkthrough of the phase. Update architecture docs. The next AI session (or the next human teammate) can now pick up exactly where this one left off — no context lost.

This cycle works for EVERYTHING — infrastructure, data analysis, literature reviews, report generation. The pattern is universal.

Where the Human Stays Essential (1/2)

Security & Permissions

AWS IAM roles, S3 bucket policies, OIDC trust relationships, and API key scoping. The AI can suggest policies and write the JSON — but only the human understands the broader organizational security context, knows which services should have which access, and can approve changes that affect the entire AWS account.

Architecture Decisions

Glue vs Lambda? Fargate Spot vs Standard? ALB or direct IP? The AI provides tradeoff analysis with pros/cons for each option — but the human decides based on budget constraints, organizational timeline, stakeholder expectations, and risk tolerance that the AI cannot fully understand.

Where the Human Stays Essential (2/2)

Stakeholder Communication

When a stakeholder reports that the forecast model is 'wrong,' the AI can investigate the code, trace the logic, and identify discrepancies. But only the human can navigate the organizational context: Is this the approved model? Who decided the methodology? Should we revert, adapt, or run both models concurrently? These are human judgments.

Data Integrity Judgment

The AI found 2 broken rows out of 3,300 dates in the aggregated dataset. It diagnosed the root cause and suggested a fix. But the human had to decide: repair in-place or trigger a full historical rebuild? What's the downstream impact on published numbers? Is a public correction needed? The AI handles the technical — the human handles the consequences.

Beyond DevOps: AI for Data Analysis

Give the AI your dataset and research question.

It writes R or Python scripts to:

- Clean and transform raw data
- Run statistical models (regression, time series, clustering)
- Generate publication-quality tables and figures
- Write interpretation of results in plain English
- Produce a complete replication package

Real example: Brookings researchers built a fully documented R package + 20-page analysis report with visualizations "*in just over a day*" using AI coding agents (Messing & Tucker, 2026).

For you: The same R skills you already have. The AI handles boilerplate, debugging, and documentation. You focus on the research design.

Beyond DevOps: AI for Literature Review & Synthesis

Describe your research domain to the AI.

The AI can:

- Search academic databases and extract key findings
- Identify methodological patterns and gaps across studies
- Draft structured literature reviews with proper citations
- Update automatically as new papers are published
- Cross-reference findings across disciplines

Survey data (Anthropic, May 2026): 81% of social scientists have used AI in research. 97% of coding agent users use it for code generation. 64% use it for editing prose. But only 20% have adopted full coding agents — the productivity gap is real and growing.

For you: AI doesn't replace your domain expertise — it amplifies your ability to synthesize and connect findings at scale.

Beyond DevOps: AI for Automated Reporting

Build automated pipelines: data pull → analysis → report → delivery

- Schedule daily/weekly/monthly data refreshes
- AI runs the analysis and generates the report automatically
- Same version-controlled, documented workflow as the dashboard project
- No more: manual Excel export → copy to Word → format → email
- Instead: push to GitHub → report renders → stakeholders notified

The same 5-step cycle applies: Grill (what's the audience? what format?) → Plan (design the pipeline) → Execute (build + test) → Verify (does the output make sense?) → Document (so next quarter's report is one click).

The dashboard project took this exact approach: daily automated pipeline → model → dashboard → zero manual steps. The same architecture works for any recurring report.

Getting Started: Your First AI Workflow

Start small, then scale. Don't migrate a dashboard. Start with one R script. Ask the AI to clean a dataset, generate summary statistics, or create a visualization. Build confidence before taking on bigger projects.

Write a README as your AI's memory. Create a markdown file describing your project, data sources, and goals. Feed this to the AI at the start of every session. Update it at the end. This becomes your external brain.

Use the 5-step cycle. Grill → Plan → Execute → Verify → Document. It works for DevOps AND for data analysis. The pattern is the same regardless of what you're building.

Keep a session log. After each AI session, write 3-5 bullet points: what was done, what worked, what broke. Start the next session by feeding this log to the AI. Context persists across sessions.

Let AI write the documentation. Always end with: 'Now write a walkthrough of what we did.' The AI excels at this. Having docs means you or a colleague can pick up exactly where you left off.

Key Takeaways

AI is not replacing you — it's amplifying you.

The human owns the decisions. AI handles the execution.

Documentation is the secret weapon.

Without it, AI has no memory. With it, AI works across days, weeks, and team members.

The 5-step cycle works everywhere.

Research, analysis, reporting, infrastructure — Grill → Plan → Execute → Verify → Document.

Start today, not tomorrow.

Only 20% of social scientists use coding agents (Anthropic, 2026). The productivity gap is real and

Thank you! • Questions? • nyc-jail-population-tracker.datacollaborativeforjustice.org

Easter Egg: My prompt to generate these slides:

✦ Grill-Me in Action: A Real Prompt Exchange

Richie:
In my GitHub repo, the jail_pop_dashboard project has been our flagship. It was primarily designed by a previous data scientist and implemented + deployed by me. It's run into some tech issues recently, so I've been fixing it with the help of AI (antigravity). I kept the entire process documented in the doc/ folder — since vibe coding agents aren't great with long-term project memory, the docs became valuable for procedure analysis.
My team is interested in the process and wants me to do a presentation about it: how to harness AI and do vibe coding productively, using this project as an example. Can you go through the project repo, especially the doc/ folder, and prepare me a few slides?
The audience: social scientists focused on criminal justice. They know statistics and CJ deeply, but not software development or DevOps. So think about how to help *them* use AI in their workflow.
And of course — *grill me if necessary.*

Dave (AI) — The Grill:
Before I dive in, a few clarifying questions:
1. Duration & slide count? How many minutes? How many slides?
2. Audience positioning? Are we teaching them to use AI themselves, or are we showing them how you used AI to maintain the system? Different emphasis.
3. Sensitive details? S3 bucket names, GoDaddy domains, IP addresses, team member emails — keep or anonymize in slides?
4. Core storyline check. From reading the docs, the story I see is:
> Lambda timed out → schema broke the forecast → 14 AI-assisted phases to migrate from EC2 to Glue+ECS → dev to production cutover → decommission V1 to save cost.
Does this storyline match what you want, or should I emphasize something else (tech debt, stakeholder communication, etc.)?

Richie — The Answers:
1. About 30 minutes
2. It's a showcase from the DevOps angle, but expand it to automated workflows. Nobody here is a developer — they write R for analysis. Don't go into technical weeds.
3. Keep it simple — the team has clearance, but they don't need those details and won't understand them without CS background.
4. The storyline should be more like:
> What was the problem? → What needed to be done? → What's the traditional way to fix it? → What can AI do differently? *(research, analysis, orchestration, planning, coding, auto-deployment, documentation)*

The key should be: how to communicate with AI, and how to harness this workflow.

Result: 23 slides, built from real project docs. Architecture diagrams added. Dense slides split. Research from Anthropic (2026) and Brookings woven in to broaden the narrative beyond DevOps. All because the grill-me step caught assumptions before any work was done.