Recovering the Bridge Paths Data Trail

By: Jenna Rashkovsky

Date: June 2026

When the Dashboard Works but the Pipeline Does Not

The Bridge Paths dashboards were already useful. Staff could see hourly pedestrian and bicycle volumes on major Bay Area bridges, compare weekdays to weekends, and track how non-motorized use changed over time. The gap was not the visualization itself. The gap was that nobody outside the original workflow could reliably explain how a number on the screen got there.

When I started this practicum project, there was no single documented path from the Eco Counter API to the dashboard-ready tables. Much of the cleaning logic lived in notebooks, tribal knowledge, and side-by-side comparisons with existing charts. That is a common situation in public agencies: the reporting product exists before the data product is fully owned.

Bridge Paths is a small domain compared with regional freeway incident systems, but it has its own complications. Counts come from physical sensors that can fail quietly. The San Francisco–Oakland Bay Bridge combines two sites with a different aggregation rule than the Richmond–San Rafael Bridge. Outliers are removed with a rolling lookback window, not a fixed cutoff. Pacific time matters because an hourly bucket at midnight UTC is not the same hour planners care about.

My job was to turn that implicit workflow into something MTC could rerun, audit, and hand off.

Reverse-Engineering Before Automating

I did not begin by writing Lambda code. I began by asking a simpler question: if I pull raw hourly data from Eco Counter and apply my best guess at the cleaning steps, do I get the same curves the dashboard shows?

That process took time. I mapped site IDs to bridge names, separated Bike and Ped modes, rebuilt RSR as a sum of Marin and Richmond, and rebuilt SFOBB as a mean of OTD and YBI after nulling whole-site zero days. I compared outputs week by week until the shapes lined up. Formal 05b documentation and stakeholder review came later in the project and confirmed most of what I had already coded in `visualization_transforms.py`.

This order mattered. Automating the wrong rules would have produced a very efficient pipeline that published the wrong history every morning. For Bridge Paths, trust in the transforms had to come before trust in AWS.

What We Actually Built

The repository now separates concerns the way a maintainable pipeline should.

Ingest lives in `build_traffic_cache.py`. It pulls site metadata and hourly traffic from the Eco Counter US API, merges new rows into a growing cache, and writes CSV, Parquet, and site metadata locally or under `/tmp` in Lambda.

Raw history lands in S3 under an `hourly/` prefix: `traffic_cache.csv`, `traffic_cache.parquet`, and `counter_sites.json`.

Transforms live in `visualization_transforms.py`. They apply the confirmed 05b rules and write `traffic_for_visualizations.parquet`.

Daily updates run through `eco_lambda_handler.py`, triggered on a schedule. Each run uses a short rolling date window, downloads the prior cache from S3 when `S3_DOWNLOAD_CACHE=1`, merges fresh API data, uploads raw files, runs transforms, and publishes the viz Parquet.

Redshift loads that Parquet on its own schedule via SQL in `sql/redshift/`. In a reference run, the visualization table held **717,551 rows** after the initial backfill and COPY.

The full history load is intentionally not a Lambda job. Eco Counter rate limits and runtime limits mean the first multi-year backfill belongs on a laptop or controlled compute environment using `scripts/run_full_backfill.sh`. Daily automation and one-time backfill are two different operating modes, and the documentation now says so explicitly.

Production Lessons That Only Show Up After Backfill

A few failures taught more than the happy-path diagram.

Cache integrity is the whole game. Lambda starts with an empty `/tmp` directory. If a run cannot read the existing cache from S3, it may merge a few days of API data into an empty file and upload that as the new truth. The fix is not smarter Python. The fix is IAM that allows `GetObject` on the cache keys and a habit of checking row counts after each deploy.

Rate limits define your calendar. The API client spaces requests and retries HTTP 429 responses with a cap. That is fine for a nightly overlap window. It is not fine for pretending a full history refresh is just another scheduled invocation.

Packaging is part of the product. The Lambda zip bundles pandas, pyarrow, and pipeline code into a ~60–70 MB artifact that usually has to be uploaded through S3 before `update-function-code`. Code in git does not change what Lambda runs until someone rebuilds and deploys the zip.

Cleaning rules are business rules. Nulling an entire SFOBB day when one site reads zero all day is not a statistical nicety. It is a decision about when staff should not interpret a flat line as real demand. Those choices belong in version-controlled code with comments pointing to the 05b spec, not in a one-off notebook cell.

What Becomes Possible Once the Table Is Real

Machine learning was not the center of this project, and it should not be the center of the story. The immediate win is simpler: MTC gets a warehouse table with stable hourly Bike and Ped counts by bridge, with the same nulling and rollup logic every time.

That alone unlocks work the dashboard was never designed to carry.

Planners can compare the same hour across bridges without rebuilding exports by hand. Analysts can measure how a bridge recovered after a closure, a fare change, or a season with unusually heavy rain—provided they join external context themselves. Staff can write SQL or connect BI tools to `bridge_path.eco_counter_bridge_hourly_viz` instead of asking someone to rerun a notebook when the question changes slightly.

There is also a monitoring story that does not require a neural network. A weekly job that plots each bridge's missing-hour rate, flags when OTD and YBI diverge before averaging, or compares this month's weekday morning bike totals to the same month in prior years would already be an upgrade over waiting for visual surprises in a dashboard.

If MTC wants to go further later, the most natural analytics extensions are practical rather than flashy: seasonal baseline bands by bridge and hour, simple forecasts to support maintenance windows, or event overlays once a shared calendar exists. Those are worth exploring only after staff agree the curated table matches the dashboards they already trust.

Handoff Is Still the Hard Part

The code is in GitHub. The architecture is documented in `docs/pipelines/`. The remaining work is organizational: production AWS credentials, a named owner for the API key and site list, CloudWatch alarms when Lambda fails or row counts collapse, and a short runbook for rerunning backfill after a rule change.

Bridge Paths will stay healthy when people treat it as infrastructure, not as an intern project artifact. The sensors keep counting. The API keeps updating. The question is whether the agency has a repeatable way to absorb those updates into the numbers decision-makers use.

That is the outcome this pipeline was built for, not a model that predicts tomorrow's bike count, but a daily path that makes today's count defensible.