WHERE THE MONEY IS

You've tuned the infrastructure. *The bigger number is still hiding in the code*

Every enterprise BigQuery estate carries recoverable spend, and infrastructure work is where most teams start — it's visible, it's finite, and in most estates it's already close to tuned. That's only one part of the game. The larger pool sits below it, in the code itself: the pool that native tooling shows you least. And it isn't only a cost problem — the same inefficient code is what makes pipelines slow and unpredictable.

01 Where the inefficiency actually lives

A PATTERN ACROSS HUNDREDS OF ESTATES

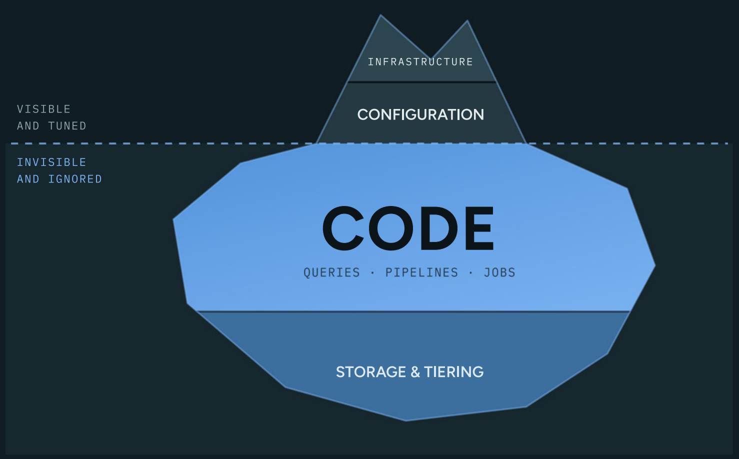
WHAT HUNDREDS OF ESTATES HAVE IN COMMON

100s of customer BigQuery estates Unravel has studied, across industries and estate sizes.

40–60% of the recoverable spend in a typical estate sits in the code itself — not infrastructure, not slot sizing.

That makes code the single largest — and least visible — category of recoverable spend in most estates, ahead of infrastructure, configuration, and storage & tiering. They are not remotely equal in size — and they are not equally visible.

Work the biggest, least-visible pool first — *not the one your dashboard happens to show you.*



The four places inefficiency hides. Proportions are illustrative, but the shape is consistent across estates: the two you can see are the two that are already close to tuned.

It shows up as cost

You are billed for every wasted scan, every shuffle-heavy stage, and every failed run that retried three times before it gave up.

It shows up as slowness

The same inefficiencies are the reason a job that should take minutes takes hours — and the reason adding compute stops helping.

It shows up as unreliability

Inefficient code is fragile code. It misses windows, breaks under volume growth, and fails in ways that look green on a dashboard.

02 What code inefficiency *actually looks like*

None of these require deep engine internals to recognize. Each one is a normal engineering decision that was reasonable when it was made and expensive at today's volume. The technical names are there for your platform team — the description is the part that matters to everyone else.

1 Scanning way more data than needed

OVER-SCANNING · POOR PRUNING · EXPENSIVE JOINS

COST · SPEED · RELIABILITY

Small question. Whole table gets scanned. Nothing looks broken, so nothing gets flagged.

IN PRACTICE

Two 20M-row tables joined for two months of data. Every run still scans all 20 million rows.

2 Work that can't split evenly

SKEW · SHUFFLE · POOR PARALLELISM

COST · SPEED · RELIABILITY

One stage holds up the whole query. The rest of the slots sit idle. Too much data through one worker, and the shuffle backs up.

IN PRACTICE

Execution details show compute time per stage — a wide gap between average and max is skew. Heavy shuffle bytes on one stage is the other tell.

3 Running more than anyone needs

REFRESH CHURN · RUNAWAY JOBS

COST · SPEED · RELIABILITY

Jobs and views refresh on a schedule nobody downstream asked for. No one compares the two, so no one notices.

IN PRACTICE

Runs 100×/day. Read twice. 98 runs pay for nothing.

4 Fails, retries, fails again

FAILED & RETRIED WORK · SILENT SKIPS

COST · SPEED · RELIABILITY

You pay for the compute either way. A job can even go green while skipping everything downstream.

IN PRACTICE

BigQuery won't retry a failed query for you — that's on whatever's orchestrating it. Get it wrong, and a broken step downstream still reports success.

5 Code nobody owns

UNATTRIBUTABLE SPEND

COST · SPEED · RELIABILITY

No name on it, so no one fixes it. Usually the biggest waste class of all.

IN PRACTICE

Pooled reservation slots don't come with a name on them — without per-job attribution, cost can't be traced back to a team.

WHY THE BACKLOG NEVER CLEARS

Finding bad code is the easy part.
Fixing it is why the backlog never clears.

Everyone can find the waste. Almost no one can fix it fast enough — not a human working alone, and not an LLM.

03 The five steps — and where four of them stall

01 Detect
Something costs or runs more than it should

02 Root-cause
Which of the five patterns, and where exactly

03 Rewrite
Change the code and the config

04 Validate
Prove it is better and still correct

05 Scale
Do it again across every job you run

WHERE THE WORK ACTUALLY IS — AND WHERE FEWER THAN 1 IN 10 FINDINGS EVER ARRIVE

Two kinds of expertise

Business logic and engine internals — rarely the same person.

Two bars to clear

Correct and faster. Both proven against real runs.

Doesn't multiply

One engineer, one job at a time. The estate keeps growing.

THE LOOP THAT MAKES TEAMS — AND COPILOTS — GIVE UP

Rewrite the query or job → **Run it** against real data → **Identical?** byte-for-byte output → **Better?** cost and runtime → **Ship** with rollback ready

NO — AND MOST ATTEMPTS COME BACK HERE MORE THAN ONCE

04 Where native tools — and LLMs — *stop*

BigQuery ships genuinely good detection — that's not a knock on the platform. What happens after detection is where both native tooling and a general-purpose LLM run out of road.

WHERE IT STOPS	ARVIX BY UNRAVEL
Native tools Give you the metrics. Never the why — or how to fix it.	Root-causes every finding automatically, from what actually ran.
A general-purpose LLM Can suggest something to try. You still have to run it, check it — and often try again.	Writes the fix and validates it against real runs — before it ships.
One suggestion at a time Doesn't cover millions of jobs a day. Or tens of thousands.	Applies the fix across every job in the estate, automatically.

Your next step

FOUR QUESTIONS FOR YOUR PLATFORM TEAM THIS WEEK

1. Can we name our ten most expensive jobs last week, by team, in under five minutes?
2. For those ten jobs, can anyone tell you why each one costs what it does — not just what it costs?
3. How many optimization findings are open right now — and how many actually shipped last quarter?
4. If we rewrote one of them today, how would we prove it's faster and still correct before it ships?

NEXT STEP · NO INSTRUMENTATION REQUIRED

Find your number *before you decide what it's worth.*

A free BigQuery HealthCheck returns your actual code-caused spend, the five patterns ranked by dollar impact, and what the first 100 rewrites would return — in about two weeks, on your own estate.

Request a HealthCheck