Quantum Post-Mortem

How A Quantum Pipeline Fooled Me Twice

The Story So Far

A sharp mind can still be fooled by a confirming result.

Rigor helps. Adversarial review helps. Open data helps. None of them make you immune to the oldest failure mode in science: seeing the thing you hoped was there, then building better and better tools for seeing it.

I know, because I did it twice.

The first time was in my ECDLP work. I thought I had a quantum-assisted signal. The outputs lined up. The pattern looked structured. The result was exciting enough to survive my first layer of skepticism.

I participated in the Project 11 Q Day Challenge, and while Giancarlo Lelli was announced the winner with a fifteen-bit solve, my fourteen-bit solve was privately confirmed, and I was awarded a consolation prize.

After the announcement, I stumbled onto Craig Gidney's falsification test of supplementing randomized data into your classical post-processing, and the result was a fail, falsifying not only my fourteen-bit solve but every methodology of classical post-processing I had been working with over the last twenty months.

In parallel, I was running another long test. I believed I had evidence that synchronized meditation was measurably interacting with IBM quantum hardware.

Not metaphorically. Not as a poetic overlay. As a measurable shift in output distributions.

The strongest sessions appeared to show register-specific reorganization around the meditation window. Control periods looked scattered. Meditation periods looked ordered. In some sessions, the apparent effect scaled with participant count. In others, the register appeared to return toward baseline after the meditation ended. Correlation structures seemed to flip sign during the intervention. Sessions with queue delays appeared to fail, which looked like a negative control: synchronization mattered.

It was exactly the kind of result that should make you nervous, because it was exactly the kind of result I wanted.

Every result in that paragraph is false.

This is not a success story. I am publishing this because a dead result can still become useful infrastructure. Two claims did not survive. The protocol discipline did. If this autopsy saves another researcher, founder, investor, or technical team from mistaking a pipeline artifact for a breakthrough, then the failed experiment still pays rent.

This article is the autopsy of how I proved it.

The Autopsy

Five artifacts produced two years of apparent results. Each one looked like physics until a specific detail gave it away. I'm presenting them in the order they were caught, with the tell that caught each one, because the tells are the transferable part. Every number below is reproducible from the repository; job IDs are included so you can pull the records yourself.

The composite timeline (Session 9-9)

This was the flagship. The charts showed the inner register's output scattering wildly during the control period, "organizing" as the meditation approached, and holding a flat, ordered line through meditation and beyond. Phase-colored plots made the alignment look unmistakable.

The first pass of adversarial review took it apart one layer, and got the mechanism wrong in an instructive way. Decomposing the register metrics into per-bit balances showed the entire effect lived in a single bit: bit 1 carried 97% of the entropy variance (correlation 0.997 between total register entropy and that one bit's contribution). That bit appeared to flip between two states; biased (fraction of ones ≈ 0.21-0.40) and balanced (≈ 0.52), before locking into the balanced state thirteen minutes *before* the meditation began. The reviewer's diagnosis: a two-level-system defect on one qubit, a documented hardware phenomenon, settling on hardware time rather than meditation time. Plausible, mundane, and wrong.

The correct mechanism surfaced when I went back to the IBM dashboard to reconcile job counts. **Session 9-9 was never one timeline. It was two backends.** I had started submitting to ibm_torino that evening; when the queue stalled, I switched to ibm_brisbane and kept going. The analysis file merged both streams and sorted them by *submission* timestamp. The decomposition is total:

- Every "biased" data point is a Torino-submitted job (bit 1 = 0.212 - 0.402, mean 0.32; IDs `d309ne...`, `d30aju...`, `d30bd7...`, 19 jobs).
- Every "balanced" point is Brisbane (bit 1 = 0.490–0.552, mean 0.52; 55 jobs).
- Two populations, **zero overlap**, sorting perfectly by backend.

The "telegraph switching" in the transition window (22:50–23:22 on the session clock) was the two submission streams interleaving. The "organization event" was the mixture ratio changing as Torino submissions tapered and Brisbane took over. There was no transition on any chip. Two machines with different inner-register baselines were glued into one plot.

And it gets worse, because the Torino jobs didn't even execute when the timeline says. The dashboard shows all of them completing in a single burst at 7:19 PM Central, roughly 1 hour 50 minutes after submission, squarely *inside* the meditation window. So Torino's control-labeled jobs and meditation-labeled jobs all sampled the machine at the same instant; the labels describe intent, not execution. Session 9-9 had three independent timeline defects at once: two backends merged, submission times used instead of execution times, and one backend's entire contribution compressed into a moment.

One inversion is worth spelling out because it shaped everything downstream. The "ordered" state I was celebrating is the register at near-maximal entropy, 4.99 bits out of a possible 5.0, i.e., near-perfect uniform randomness, which is exactly what this circuit should produce on healthy hardware. The "chaos" was one backend's biased qubit dragging the entropy down. I had the labels backwards from day one: the effect I attributed to consciousness was, physically, a plot of *which computer the job ran on*.

Within each backend separately, every phase is statistically flat.

The tell: per-bit decomposition plus a bimodal distribution with no overlap. A real register-level effect doesn't live in one bit, and a real temporal transition doesn't partition perfectly by backend ID. When something in your data forms two clean populations, check the metadata before the metaphysics.

The decoder that analyzed compressed bytes

When I re-extracted the sessions in 2026 with a newer "quantum metrics" script, three sessions showed exactly what my model predicted: effects starting near the meditation, scaling with

participant count, and a correlation sign flip that appeared only during the meditation window (7-11: coherence–entanglement correlation +0.911 in control, −0.602 during meditation).

None of those numbers came from a quantum computer.

The extraction function decoded IBM's result payloads like this: base64-decode, then `np.unpackbits`, directly on the payload. The payload is zlib-compressed. The correct path (present in the *same file*, in a function the batch pipeline didn't call) is base64-decode → `zlib.decompress` → `np.load`. The deployed path skipped decompression entirely, so every "bitstring" it produced was a 5-bit slice of a compressed byte stream, which is, by the design of compression, statistically indistinguishable from uniform random noise regardless of what the hardware did.

Three fingerprints proved it:

1. **Impossible shot counts.** Every real job had exactly 8,192 shots. The buggy output reported 8,726-8,761 "shots" for 5-bit registers and 11,768-12,016 for the 1-bit register, varying job to job, because the number is actually the bit-length of the compressed payload divided by the register width, wobbling with each job's compression ratio.
2. **It erases real signals.** Simulation with known ground truth: a register with one heavily biased qubit (the 9-9 Torino regime) reads coherence 0.073 through the correct decoder and 0.001 through the buggy one, identical to pure noise (0.0005). Even 8,192 shots of a *perfectly deterministic* all-zeros state come out looking like static.
3. **It's insensitive to everything.** A metric pipeline that produces the same output for a deterministic state and a random one is free to "confirm" any hypothesis. Its correlations across sixteen jobs are correlations between noise series; sign flips between small subsamples of noise aren't just possible, they're guaranteed.

The fix was two lines plus one assertion (`num_shots == 8192` on every job, an assertion that would have caught the bug the day it shipped). Validation: the corrected pipeline reproduces the original 2025 extraction job-for-job at correlation +1.0000, maximum difference 0.00000, across all 55 shared jobs. Both decoders were then correct; only the 2026 batch had ever been garbage.

The tell: an impossible invariant. Shot counts are set by the experimenter and cannot vary. Any pipeline output that violates a known invariant of the experiment is describing the pipeline, not the experiment.

The session with no time axis (Session 5-25)

Session 5-25 was classified as a "failed synchronization", part of what I treated as a natural negative-control class: delayed sessions show no effect, therefore the effect requires

synchronization. The session's own queue report, however, said "Delay from intended start: 0.0 hours," which contradicts the exclusion.

The dashboard resolved it: the report computed delay from *submission* times. The seventeen jobs were submitted on schedule (5:55-6:43 PM) and executed roughly four hours late, in a back-to-back burst between 9:39 and 9:41 PM. Every job sampled the backend within the same two minutes.

That doesn't just invalidate 5-25 as a synchronized session; it invalidates the entire negative-control argument. A burst-executed session doesn't show "no temporal structure because no one was meditating in sync." It shows no temporal structure because *it has no time axis*, seventeen measurements of one instant. "The effect vanishes when synchronization fails" was fully confounded with "the time series vanishes when the queue collapses." My strongest evidence class was uninformative by construction.

The tell: an internally contradictory report ("delay: 0.0 hours" on a session excluded for delay), and completion timestamps clustered within minutes of each other. Reconcile every report against the primary source before it becomes an evidence class.

The statistics that lie on drifting hardware

Even after the artifacts above fell, standard statistics kept offering hope. T-tests and ANOVA across phases returned a long list of $p < 0.05$ results, some at $p = 0.0002$.

Every one of them is an artifact of applying independence-assuming tests to autocorrelated time series. QPU proxy metrics sampled every 1-3 minutes drift; consecutive jobs are correlated. My strongest "significant" series, 9-9 central-register coherence, has lag-1 autocorrelation of **+0.65**. Simulating effect-free AR(1) series at that autocorrelation with my exact phase block sizes: the t-test fires "$p < 0.05$" **40%** of the time and ANOVA **56%** of the time, against a nominal 5%. At the battery size I ran (~108 uncorrected comparisons), my ~19 "significant" hits represent an 18% hit rate, at or *below* what pure drift generates. The tests weren't detecting an effect. They were measuring how much the hardware wanders.

The flagship hit dissolves under one extra plot: "central coherence, Meditation vs Post, $p = 0.0002$" is a monotone climb *within* the post phase alone ($r = +0.70$, $p = 0.008$ within-phase), a continuous slope passing through the phase boundary, which a comparison of block means misreads as a level shift. Simpler heuristics fared no better: a ">10% change between phases" criterion fires on two random halves of the *same* phase 69% of the time. A detector that triggers on identical conditions seven times out of ten is a random-number generator wearing a lab coat.

The statistically valid test, exact circular-rotation permutation, which compares the real phase labels against every time-shifted version of the same drifting series and therefore cannot be fooled by drift, combined with false-discovery-rate correction across all 36 session × register ×

metric endpoints, returns: **zero significant results.** One raw p < 0.05, against ~1.7 expected by chance.

The tell: significance that scales with autocorrelation and evaporates under a block-respecting null. If your effect survives a t-test but not a rotation test, you have measured drift.

The metrics that weren't what their names said

The last artifact wasn't a bug. It was the vocabulary.

Reverse-engineering the pipeline's outputs against the raw data (the 1-bit central register acts as a Rosetta stone, with one bit, candidate formulas either match exactly or fail exactly) showed what the named metrics actually compute:

Name on the chart	**What it actually is**
entropy	Shannon entropy of the measured bitstring distribution
entanglement	entropy ÷ number of bits, *identically*, to machine precision (10^{-15}), on all 296 rows
coherence	probability of the single most frequent bitstring
interference	1 − (max outcome probability − min outcome probability)

"Entanglement" was never a second observable; it is the entropy chart with a rescaled axis, and I counted it as corroboration for two years. None of the four can detect the physics it's named after, they are descriptive statistics of a classical output distribution, and the "entanglement" score is *maximized* by uniform randomness: a fully decohered, completely broken quantum computer outputting coin flips scores a perfect 1.0. The later "quantum metrics" pipeline was no better: its "purity-based coherence" is a quadratic function of the same per-bit biases; its "von Neumann entanglement" is classical mutual information whose correctly-decoded values (~0.0005-0.0008) sit exactly at the analytic small-sample bias floor of the estimator at 8,192 shots, the noise floor, measured and mistaken for a signal; its "FFT interference" fabricates amplitudes as √p with every phase set to zero (measurement destroys the phases where quantum interference actually lives) and then, through an indexing bug (`fft_result[::dim+1]` on a 1-D array), reduces to yet another non-uniformity statistic.

The metrics are also mechanically intertwined, pooled across the data, "entanglement" and "interference" correlate at +0.97 by construction, so the cross-metric relationships I treated as

independent evidence were largely the same quantity in different costumes, correlated with itself.

The tell: compute your metric on a known state before trusting it on an unknown one. A maximally mixed state should not score maximal "entanglement." If the name and the formula disagree, the formula is telling the truth.
Everything above compresses into a set of rules. None of them require a PhD; all of them would have prevented a specific artifact in this project, and each rule below names the one it kills. If you run experiments on NISQ hardware, for any hypothesis, mundane or wild.

The Checklist I Wish I Had Before I Started

Timeline integrity

1. **Use QPU execution timestamps, never submission times or file times.** Every timeline artifact in this project, the 9-9 composite, the 5-25 collapse, a re-extraction that stamped every job with the current date, came from this one substitution. The execution timestamp is in the job's result metadata. Use nothing else.
2. **One backend per timeline, tagged and asserted.** Never merge backends into a single temporal series; their baselines differ by 5-20× on distribution metrics, at $p < 0.001$, independent of anything you did. Record the backend on every row and assert uniqueness before any temporal analysis.
3. **Check execution spacing before treating data as a time series.** If jobs are executed in a burst, you have a snapshot, not a series, and a snapshot cannot serve as a temporal control or a temporal effect. Compute inter-job execution gaps as a standard preflight.

Pipeline integrity

4. **Assert every invariant you control.** Shot count, register width, unique-state ceiling. One line, `assert num_shots == 8192`, would have caught the decompression bug the day it was written instead of a year later.
5. **Freeze one pipeline and validate it against ground truth.** Run your extractor on synthetic data with a known answer (a deterministic state, a planted bias) before running it on the experiment. Cross-validate any pipeline change against the old one job-by-job; the agreement should be exact, and if it isn't, stop.
6. **Write down what each metric actually computes, next to its formula.** Then test the formula on states where the right answer is known. If your "entanglement" metric scores a coin-flipping brick at 1.0, rename it before it renames your conclusions.

Design integrity

7. **Pre-register the primary endpoint and the exclusion criteria, in timestamps and thresholds, before seeing results.** "Synchronized" must mean something like "all jobs executed within N minutes of intended time," decided in advance. Otherwise excluded sessions quietly become whichever sessions didn't show the pattern, and your negative-control class becomes post-selection.
8. **Controls must bracket the intervention, not just precede it.** A control block that is always the session's opening minutes absorbs all early-session settling and calibration drift, contaminating every comparison. Interleave or bookend.
9. **Run the sham condition.** Same circuit, same backend, same cadence, same duration, nobody doing anything. It is the only way to learn what your pipeline produces from hardware alone, and if you find yourself arguing the sham is unnecessary because you already know its result, that argument is the alarm.

Statistical integrity

10. **On drifting hardware, permutation nulls beat parametric tests, and the null must respect the time structure.** Circular rotation of phase labels preserves autocorrelation; naive shuffling and t-tests do not. At the autocorrelation levels typical of session-length QPU series ($r_1 \approx 0.25$-0.65), independence-assuming tests false-positive at 26–76%. Also respect granularity: a 16-job session has only 16 rotations, so its floor is $p = 0.0625$, some designs cannot reach significance no matter what happened, and you should know that before running them.
11. **Correct for every look.** Registers × metrics × phase-pairs × sessions multiplies fast; ~108 uncorrected comparisons *guarantee* a page of "findings." Benjamini–Hochberg is four lines of code.
12. **Demand sign replication before belief.** A real register-specific effect points the same direction twice. Mine went −61%, +29%, −8% across three sessions on the same endpoint, the signature of noise, visible the moment I looked for it.
13. **Decompose composite metrics to their components as a standard diagnostic.** Per-bit balances would have exposed both the single-bit dominance and the two-backend structure on day one. Any register-level claim that survives should also be visible, and distributed, at the bit level.

Epistemic integrity

14. **Specify, in advance, what result would falsify the hypothesis, and let it.** My model accommodated effects during the meditation, before it, increasingly before it across sessions, with a return to baseline and without one. A model that no timing can falsify is a model that no timing can confirm. This was the deepest bug in the project, and no

assertion statement catches it. Write the kill condition down before the first job is submitted, and when the data meets it, stop.

15. **When a result dies, log it and re-derive from the primary source.** Keep a falsification log in the repository. When numbers don't reconcile, go back to the dashboard, the raw payload, the original record, twice in this project the primary source overturned an explanation everyone involved, believer and skeptic alike, had already accepted.

The list is long because there are many ways to fool yourself. The theme is one sentence: **an experiment is trustworthy exactly to the degree that it is built to survive your own preference for its outcome.** Mine wasn't, for two years. It is now, and it took losing the result to build it.