Research to Runtime Correctness Under Adversarial Numerics

MorphIQ Labs Research

Authored by Stephen Romano for MorphIQ Labs

July 2026

Version 0.1 — Draft for review

A research result and a production system are two different artifacts, and the distance between them is where correctness quietly fails. A pricing model is validated in a notebook and re-implemented in a fast language; a signal is proven on clean historical data and then run on a live feed; a strategy is backtested against one ordering of events and deployed into another. Each translation is a place where the mathematics that was checked stops being the mathematics that runs. In quantitative trading, that gap is a dominant source of model risk, and it is largely unmeasured.

This paper describes how MorphIQ Labs closes that gap. The thesis is deliberately narrow and falsifiable: correctness and runtime performance are properties of the *same* artifact, not a trade-off between two of them, and the disciplines that keep them together—reference oracles, proof gates on stable laws, property-based and mutation testing, deterministic replay, and bounded arithmetic—are engineering contracts we can show, not adjectives we assert. We describe the standard, point to the evidence that the standard is real across the Ferro engine suite, and—as important—state precisely what we do *not* claim. The companion whitepapers cited throughout are the receipts; this paper is the map to them.

Contents

1 The Seam **3**

2 The Claim **3**

3 The Standard **4**

4 The Evidence **5**

4.1 Reference oracles: FerroWave and FerroRisk 5
4.2 Proof gates on stable laws: FerroReplay, FerroWave, FerroRisk 5
4.3 Mutation-tested transforms: FerroWave 6
4.4 Determinism as a contract: FerroReplay 6
4.5 Release gates: FerroRisk 6

5 What This Buys a User **6**

6 What We Do Not Claim **7**

7 Closing **7**

1 The Seam

Most software-correctness effort is spent inside a single artifact: this function, given this input, returns this output. Quantitative systems fail somewhere else—at the *seam* between the research that established a result and the runtime that is supposed to preserve it.

The seam has a characteristic shape. A model is derived and checked in a research environment optimized for expressiveness: arbitrary precision, rich libraries, no latency budget, a fixed and friendly dataset. It is then re-expressed in a runtime environment optimized for the opposite: fixed-width arithmetic, a hard latency budget, adversarial and incomplete inputs, and event orderings no one chose. Nothing forces the second artifact to agree with the first. The tests that pass are usually the tests someone thought to write, over the inputs that occur most often—which is exactly the region where two implementations are *most* likely to agree and least likely to reveal a divergence.

The failures that result are not loud. They are a Greek that is correct at the money and wrong in the deep wings; an implied-volatility solver that converges to the wrong root in an admissible-but-rare region; a transform that is exact on the sample and biased on the tail; a backtest that is reproducible only because yesterday a thread won a race it will lose tomorrow. None of these throw. They produce a number, and the number is used to rank, hedge, size, or explain a decision. This is what *model risk* means operationally: not that the model is wrong on paper, but that the running system silently stopped being the model.

The industry's usual answer is process *around* the seam—model-validation groups, sign-off gates, reconciliation reports. Those are real controls, but they treat the seam as a handoff to be inspected. Our position is that the seam should not be a handoff at all.

2 The Claim

The claim is deliberately small enough to be wrong:

> *Correctness and runtime performance are properties of the same artifact. The rigor that validates the research is carried—provably, where the law is stable enough to specify—into the artifact that runs at speed.*

Two things this claim is careful *not* to say. It does not say the artifact is "verified" in the sense that its outputs are guaranteed correct; only specific, stable primitive laws carry maintained proofs, and we are precise about which (§6). And it does not say correctness was achieved *by* sacrificing speed, or speed by sacrificing correctness. The load-bearing engineering observation is that the two do not trade against each other when the disciplines are in place from the start rather than retrofitted: a reference oracle is a test-time dependency, not a runtime one; a proof gate runs in CI, not in the hot path; determinism is a structural property, not an overhead.

The remainder of this paper is the evidence for that claim: the standard (§3), the receipts (§4), what it buys a user (§5), and its limits (§6).

3 The Standard

We hold a layered standard. No single discipline is sufficient; each is chosen to catch a failure mode the others structurally cannot. This section describes the *shape* of the practice—what each layer is and why it exists. The mechanics live in the per-engine companion papers cited in §4.

Reference oracles. A test that compares an implementation to itself proves only that it is self-consistent. Numerical kernels are instead checked against an *independent* source of truth computed at far higher precision—an arbitrary-precision oracle, or a canonical reference implementation—whose answer is trusted because it is derived differently and to more digits than the runtime path. This is the discipline that catches the deep-wing Greek and the wrong-root solver, because it evaluates agreement *where the fast path is under stress*, not where it is comfortable. The oracle is a test dependency, never the runtime path; the runtime stays fast.

Proof gates on stable laws. Where a property is a stable mathematical law—monotonicity, put–call parity, no-arbitrage bounds, the clock and interval laws of a deterministic scheduler—it is stated as a law and machine-checked in a proof assistant; where it is a required check, it gates the build. Proof gates do not replace tests; they cover the cases tests can only sample. They apply only where the law is stable enough to specify, which is a minority of any system and precisely the minority whose violation is catastrophic.

Property-based testing. For behavior that is lawful but not provable in closed form, we assert the *invariant* and let a generator attack it across a large, randomized input space—including the pathological corners a hand-written suite omits. Properties test the region between "specific example" and "formal proof."

Mutation testing. A passing test suite says the tests pass; it does not say the tests would *notice* a defect. Mutation testing deliberately injects faults and measures how many the suite kills. It is the discipline that tests the tests, and it is why a high test count is not, by itself, a correctness claim we make.

Deterministic replay. Wall-clock time, iteration order, and thread races are untested inputs in most systems. We remove them as a class: the runtime has no direct access to time or nondeterministic ordering, a constraint enforced mechanically rather than by convention, so that a backtest and a production run can be the *same program* over the same event tape. Determinism is what makes every other guarantee reproducible; without it, a result you cannot re-run is a result you cannot trust.

Bounded arithmetic. Where correctness depends on it, we do not use floating point for value-bearing quantities; kernels that must use floating point carry known error envelopes rather than hope. The point is not purity—it is that rounding behavior is a specified property, not an accident of the hardware.

Benchmark gates. Performance is a contract too. Perf-sensitive paths carry benchmarks whose regressions gate the build, because an artifact that is correct but misses its latency budget has failed the *same* claim as one that is fast but wrong.

The layers compose into defense in depth: the oracle catches numerical divergence, proofs cover the stable laws, properties attack the lawful-but-unproven middle, mutation testing validates that the whole suite has teeth, replay makes all of it reproducible, and the benchmark gate keeps correctness from being bought with speed.

4 The Evidence

A standard is only a claim until it produces artifacts. The disciplines above are documented, per engine, in a family of companion whitepapers; the following are representative receipts, each checkable against a repository or a published paper.

4.1 Reference oracles: FerroWave and FerroRisk

FerroWave anchors every numerical value in the library to a citable external authority and treats a failing reference test as a finding, not a tolerance to relax. Its methodology paper documents this discipline catching, among others, a wavelet-normalization error that had been latent since the project's first commit, and a Paul-wavelet admissibility-constant error inflated by three orders of magnitude—each surfaced by comparison against an analytical or canonical reference, not by a self-consistency test that a buggy implementation can satisfy.[1] FerroRisk applies the same posture with an arbitrary-precision oracle: its Greeks (all ten, for both the Black–Scholes and Black-76 models), the normalized-Black price and vega, and the underlying Gaussian numerics are validated against a 512-bit MPFR reference, gated behind a test-only feature and run in CI.[2] Its implied-volatility solver is validated against a different independent authority—Peter Jäckel's published "Let's Be Rational" reference—rather than against MPFR, precisely because the right oracle for a solver is another trusted solver.[3]

4.2 Proof gates on stable laws: FerroReplay, FerroWave, FerroRisk

The proofs are written in Lean 4 and carry no `sorry` and no custom axioms; the trusted base is audited. FerroReplay carries machine-checked clock, wake, interval, and realignment laws under a *blocking* CI gate—though, stated precisely, these prove a specified model that mirrors the runtime surface, not the Rust code directly.[4] FerroWave proves thirteen structural primitive-contract laws (index, length, and boundary logic) on the extracted Rust via Aeneas under a blocking gate, and separately carries 111 theorems over $\mathbb{R}$ in Lean/mathlib—perfect reconstruction $\mathrm{IDWT}(\mathrm{DWT}(x)) = x$, Parseval energy preservation, filter orthonormality—plus Gappa floating-point bounds (Haar DWT step ≤ 1 ulp, db2 ≤ 1.6 ulp).[5] FerroRisk states put–call parity, no-arbitrage price bounds, monotonicity in spot, convexity, and the Black–Scholes PDE

[1] *Reference-Based Numerical Validation in FerroWave*, MorphIQ Labs Research.
[2] *Pricing Models* and companion FerroRisk notes, MorphIQ Labs Research.
[3] *Implied-Volatility Solver*, MorphIQ Labs Research.
[4] *FerroReplay*, *Sequence-not-Timestamp*, and *Crash-Tolerant Replay*, MorphIQ Labs Research.
[5] *Reference-Based Numerical Validation in FerroWave*, MorphIQ Labs Research.

as machine-checked Lean theorems over $\mathbb{R}$.[6] The honest scope is stated in §6: the extraction proofs cover structural logic (Aeneas has no floating-point model), the $\mathbb{R}$ proofs cover the idealized mathematics, and the floating-point gap is bounded separately by Gappa or measured in conformance tests.

4.3 Mutation-tested transforms: FerroWave

FerroWave sets a minimum mutation-kill floor of 70% and measures above 90% in practice—evidence the suite detects injected faults, not merely that it is large. This is the discipline behind the claim that its more than 1,800 tests have teeth, rather than the count being the claim itself.[7]

4.4 Determinism as a contract: FerroReplay

The deterministic-replay constraint is enforced mechanically—not by reviewer discipline—through two independent, blocking gates: a Clippy `disallowed-methods` lint that bans direct wall-clock and timer APIs (`SystemTime::now`, `Instant::now`, the `tokio::time` family), and a separate textual CI check that greps the source for the same tokens ("belt and suspenders"). Exactly one sanctioned clock module is exempt, so a backtest and a production run are the same program over the same event tape.[8]

4.5 Release gates: FerroRisk

FerroRisk's CI enforces formatting, Clippy warnings-as-errors, the full regression suite (which runs the MPFR oracle tests), documentation warnings-as-errors, a fuzz build check plus fuzz smoke runs, SIMD correctness under aarch64 emulation, and a required Lean model-proof gate; performance thresholds are enforced as a release gate. A release cannot silently drop a guarantee because each of these is a gate, not a guideline.

Each item above is a claim a reader can check: an artifact exists, and it says what we say it says. That is the standard this paper is itself held to.

5 What This Buys a User

Rigor is not held for its own sake; it changes what a user can *rely on*.

- **Numbers you can stand behind.** A risk surface whose contracts are proven or oracle-checked can be *explained*—parity holds, monotonicity holds, the wings are specified—rather than merely produced.
- **Backtests that mean something.** When production is the same program as the backtest, a result that reproduces is evidence of determinism, not of profit, and not coincidence.
- **Failure where it is visible.** Bounded arithmetic and stated error envelopes turn silent numerical drift into a specified, inspectable property.

[6] *Put–Call Parity and the Forward* and *Pricing Models*, MorphIQ Labs Research.
[7] FerroWave mathematical-validation documentation, MorphIQ Labs; per-run figures and scope are recorded there.
[8] *Determinism as a Correctness Contract*, MorphIQ Labs Research.

- **A standard that does not decay.** Gates run on every commit. The correctness is a maintained contract, not a launch-day snapshot.

6 What We Do Not Claim

A paper about rigor earns trust in proportion to what it refuses to overstate. The limits are therefore part of the standard, not a disclaimer appended to it.

- We do **not** call any product, platform, application, or trading strategy "formally verified." The claim is specific, stable *primitive laws* carrying maintained proofs—never investment outcomes, model certainty, execution quality, or risk-free trading.
- A proof boundary covers the law it specifies and nothing outside it. Where the underlying law is not stable enough to specify, we say so, and rely on oracles, properties, and mutation testing instead.
- Machine-checked does not mean blocking everywhere. The structural primitive contracts and the deterministic-clock model laws gate CI today; the deeper mathematical-law proofs over $\mathbb{R}$ (reconstruction, Parseval, put–call parity, the Black–Scholes PDE) are machine-checked but currently run as non-blocking proof surfaces. And the Rust-extraction proofs (Aeneas) cover structural and index logic, not floating-point values, which are bounded separately. We do not claim the compiled floating-point binary is end-to-end proven.
- Evidence is verifiable but bounded: a caught-bug narrative is one data point, a mutation-kill rate is a property of a suite at a point in time, a benchmark is a measurement under stated conditions. We publish the conditions.
- Rigor reduces model risk; it does not eliminate it. The residual is the part we are honest about, because a rigor practice that oversells a single claim spends more credibility than any one result can return.

7 Closing

The seam between research and runtime is where quantitative systems actually fail, and it is usually managed by inspecting a handoff. We do not treat it as a handoff. The same rigor that validates the research is carried into the artifact that runs—oracle-checked, proof-gated where the law is stable, mutation-tested, deterministic, and benchmarked—so that the mathematics that ships is the mathematics that was proven. The receipts are in the repositories and in the companion whitepapers, and they are the standard this paper holds itself to.

Companion Whitepapers

The following are the primary evidentiary sources for this thesis. All are MorphIQ Labs Research whitepapers.

- *Reference-Based Numerical Validation in FerroWave* — the reference oracle and test-triangle methodology, with case studies.

- *Implied-Volatility Solver*; *Pricing Models*; *Volatility Surface*; *Put–Call Parity and the Forward* — FerroRisk numerical contracts and oracle validation.
- *FerroReplay*; *Sequence-not-Timestamp*; *Crash-Tolerant Replay*; *Determinism as a Correctness Contract* — deterministic replay primitives and their proven laws.
- *Portable SIMD Numerical Kernels* — performance under the same correctness contract (note: NDA-tier; cite existence, not internals).

Colophon

Version 0.1 — draft for review. July 2026.

This paper describes engineering correctness practices. It is not investment advice, not an offer or solicitation, and makes no representation of trading performance or outcomes.