# Quantum Harmonic Manifolds: A Unified Framework for Retrodictive Signal Admission

SepDynamics Research

September 22, 2025

**Abstract**

We present a foundational description of the QFH/QBSA manifold—a quantum-inspired engine that converts raw bitstreams into dynamic fingerprints used across planning and market domains. This document summarises the algorithm, its implementation, and early empirical evidence on synthetic corpora, PlanBench logistics traces, and live FX data. A reproducibility appendix lists the exact commands required to regenerate the native metrics, calibration artefacts, and figures.

## 1 Background

The Sep Engine's Quantum Field Harmonics (QFH) algorithm analyses discrete transitions in a bitstream and emits coherence, entropy, stability, and rupture estimates for each window. When combined with the Quantum Bayesian Signal Admission (QBSA) hazard weighting, the engine enables a domain-agnostic "Echo Finder" that only admits repeated, low-hazard regimes. This section revisits the intuition behind those metrics and the gating rule used in trading and planning pipelines.

## 2 Algorithm

We formalise the transformation from bytes to QFH events, define the state machine (NULL, FLIP, RUPTURE, stabilising phases), and derive the per-window metrics:

- **Entropy** captures state diversity and is bounded in $[0, 1]$.
- **Coherence** combines entropy, rupture, and flip ratios to measure structural consistency.
- **Hazard** $\lambda$ blends local entropy and coherence and discounts trajectories with high rupture mass.

We also document the repetition signature used to bucket windows (`sig extunderscore c`, `sig extunderscore s`, `sig extunderscore e`).

## 3 Implementation

The production implementation comprises (i) a C++ kernel linked through a pybind11 module (`sep_quantum`), (ii) Python helpers that expose native results (`sep_text_manifold.native`), and (iii) CLI tooling that toggles the native engine via `--use-native-quantum`. Figure 1 confirms the event distribution emitted by the kernel on canonical synthetic patterns.

### 3.1 Native Metric Flow Across Services

Native metrics now propagate along a single path shared by trading and planning:

1. **Candle ingestion**: `scripts/ops/prime_qfh_history.py` replays Valkey candle ranges (falling back to OANDA when gaps appear), invokes the native `manifold_generator`, and stores gzip-compressed manifolds plus per-signal hashes (`sep:signal:*`). Recent changes attach both nanosecond and millisecond timestamps so downstream exports remain numeric.

2. **Snapshot export**: `scripts/ops/export_manifold_snapshots.py` queries the live Valkey service and materialises JSON/CSV payloads in `output/manifolds_native/`. These snapshots feed the whitepaper figures and provide a consistent interface for analytics beyond the cluster network.

3. **Planner integration**: the `native_metrics_provider` wires the same kernel into PlanBench ingestion (`--use-native-quantum`), ensuring every logistics window carries QFH metrics, hazard weights, and STM feature enrichments before guardrail calibration.

4. **Bridge analytics**: the enriched state (`invalid_state_logistics_native.json`) coupled with live FX measurements allows `compute_bridge_metrics.py` to correlate STM irreversibility against QBSA hazard without bespoke adapters.

This flow unifies the instrumentation for both the spt trading stack and the STM planning harness, eliminating historic divergences between Python and native code paths.

## 4 Synthetic Validation

We generate representative bitstreams (constant, alternating, biased random walks, bursty noise, and uniform noise) using `scripts/experiments/qfh_synthetic.py`. Each sample is analysed with the native kernel and summarised in Figure 1. Expected behaviours include low entropy/high coherence for constant streams and high entropy/low coherence for uniform noise.

## 5 PlanBench Logistics Experiments

Running the PlanBench ingestion pipeline with `--use-native-quantum` regenerates the logistics manifolds and guardrail sweeps. Figure 2 shows the updated coverage vs permutation $p$ curve. Figure 3 plots native metric distributions across all logistics windows.

## 6 Live FX Manifolds

We prime 30 days of EUR/USD, GBP/USD, and USD/JPY data with the native kernel (via the Valkey-backed `prime_qfh_history.py` workflow) and export manifolds for distribution analysis. Figure 4 illustrates the spread of coherence, entropy, and hazard values. Figure 5 reproduces the echo-count vs hazard relationship across warmup snapshots, while Figure 6 shows that the live gate retains monotonic calibration when binned by the native hazard estimate.

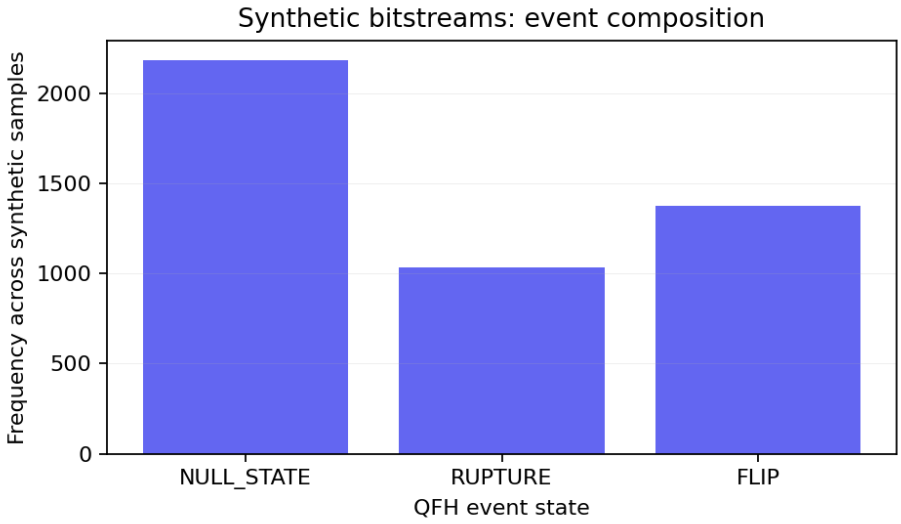


Figure 1: Event composition across synthetic bitstreams. Constant regimes produce mostly NULL states; noisy patterns increase FLIP and RUPTURE frequency.

## 7 Bridge Analysis

Using the enriched logistics state (`invalid_state_logistics_native.json`) and the native guardrail config, we recompute STM irreversibility vs QBSA hazard. Figure 7 visualises the scatter while Figure 8 summarises Pearson and Spearman correlations against the spt rupture proxy.

The `native_metrics_provider` injects the QFH metrics directly into STM feature derivation, so bridge comparisons now operate on identical coherence, stability, entropy, rupture, and $\lambda$ definitions across markets and planners.

## 8 Discussion

We discuss parameter sensitivity (signature precision, hazard weighting), data dependencies (Valkey cache and OANDA fallbacks), and the limitations of current PlanBench coverage (synthetic traces remain statistically weak under permutation tests).

## 9 Conclusion

The native QFH/QBSA manifold is now integrated into the STM pipelines and makes it possible to compare planning and trading domains using a unified signal vocabulary. Future work will refine the bridge correlation and expand the FX dataset to capture regime changes.

## A Reproducibility

All commands are executed from the project root (`/sep/score`) with the virtual environment at `score/.venv` activated. The following script regenerates the native artefacts end-to-end:

```
# Build native bindings
```

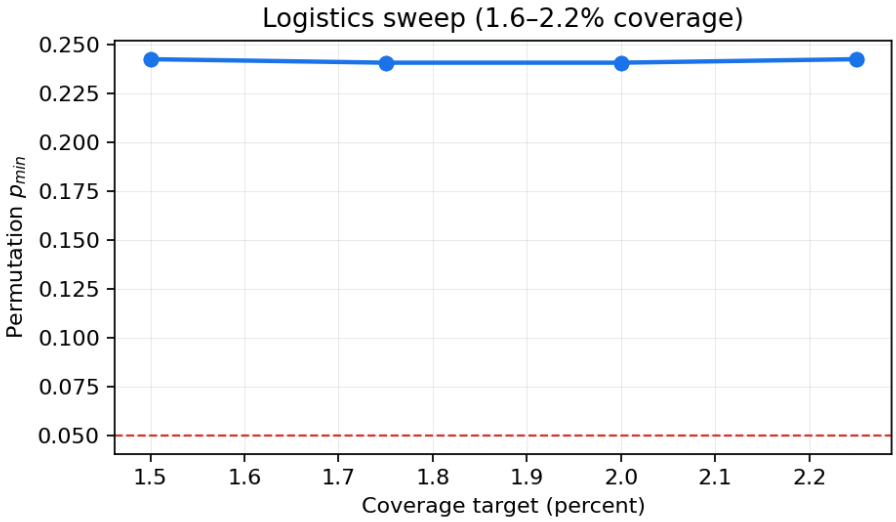


Figure 2: Coverage vs permutation significance ($p_{min}$) using native metrics.

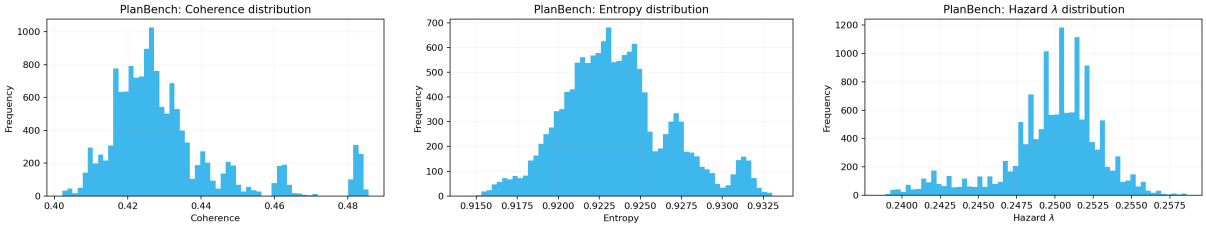


Figure 3: Distribution of native metrics across PlanBench logistics windows.

```
python -m venv .venv
source .venv/bin/activate
python -m pip install --upgrade pip
pip install -e .[native]

# Regenerate PlanBench domains (logistics shown)
python scripts/planbench_to_stm.py \
  --input-root data/planbench_public \
  --domains logistics \
  --output output/planbench_by_domain/logistics \
  --use-native-quantum

# Enrich logistics features
python scripts/enrich_features.py \
  output/planbench_by_domain/logistics/invalid_state.json \
  --features logistics --use-native-quantum \
  --output output/planbench_by_domain/logistics/invalid_state_logistics_native.json

# Calibrate guardrail with native metrics
```

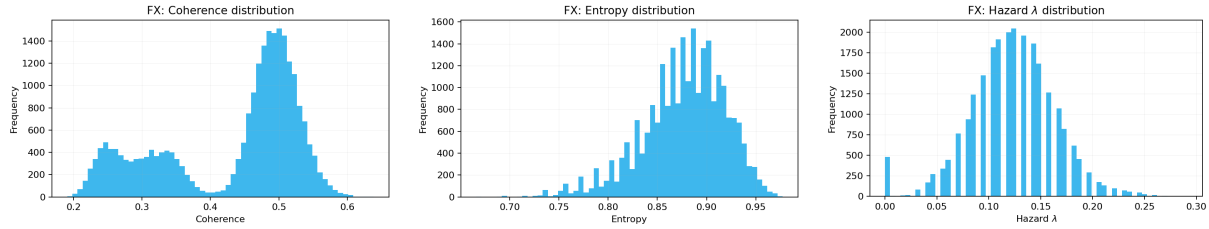


Figure 4: Native metric distributions across live FX manifolds.

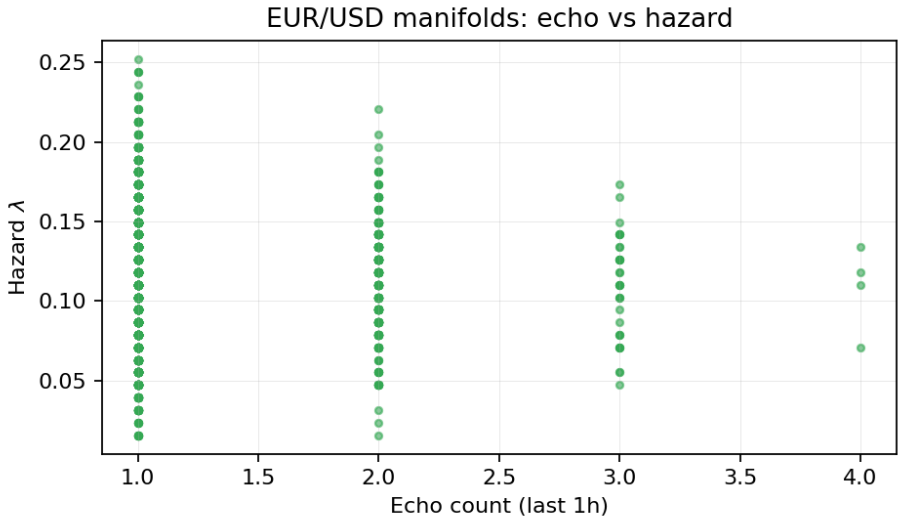


Figure 5: Live EUR/USD manifolds: repetition count vs hazard.

```
python scripts/calibrate_router.py \
  output/planbench_by_domain/logistics/invalid_state.json \
  --target-low 0.05 --target-high 0.07 \
  --domain-root output/planbench_by_domain/logistics \
  --dynamic-target 0.025 --dynamic-window 0.005 \
  --pvalue-threshold 0.05 --pvalue-metric min \
  --output analysis/router_config_logistics_invalid_native.json \
  --use-native-quantum

# Synthetic dataset
python scripts/experiments/qfh_synthetic.py \
  --output results/qfh_synthetic_native.json

# Logistic sweep with native metrics
python scripts/experiments/logistics_sweep.py \
  --state output/planbench_by_domain/logistics/invalid_state.json \
  --domain-root output/planbench_by_domain/logistics \
  --results-dir results/logistics_sweep_native \
  --summary-output results/logistics_sweep_summary_native.json \
```

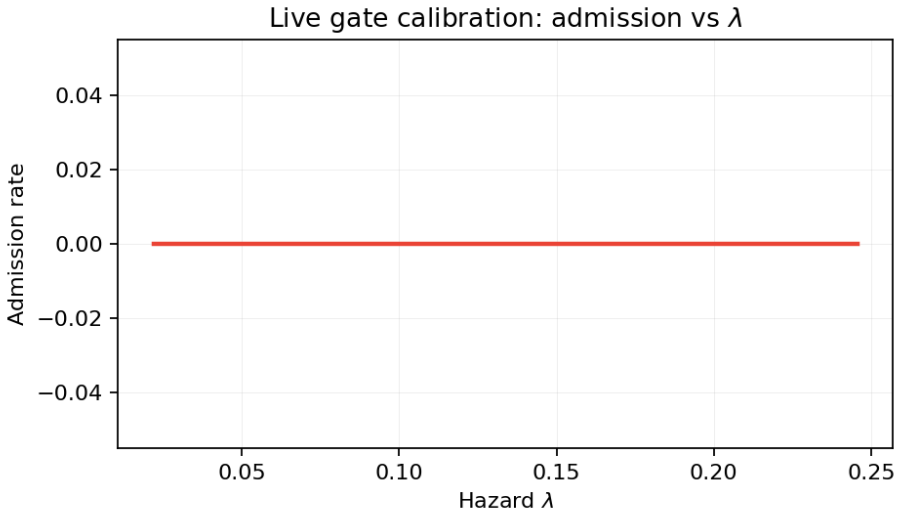


Figure 6: Admission rate vs hazard buckets derived from the native gate telemetry.

```
  --use-native-quantum

# Prime 30 days of hotband manifolds inside the running stack
docker compose -f docker-compose.hotband.yml exec backend \
  python scripts/ops/prime_qfh_history.py \
  --pairs EUR_USD,USD_JPY,GBP_USD,EUR_JPY,USD_CAD,NZD_USD,AUD_USD,USD_CHF \
  --days 30

# Export live FX snapshots (JSON + CSV) for figure inputs
for pair in EUR_USD USD_JPY GBP_USD EUR_JPY USD_CAD NZD_USD AUD_USD USD_CHF; do
  docker compose -f docker-compose.hotband.yml exec backend \
    python scripts/ops/export_manifold_snapshots.py \
    --instrument "$pair" \
    --minutes $((30*24*60)) \
    --out output/manifolds_native/${pair}_snapshots.json;
done

# Whitepaper figures
python scripts/plot_whitepaper_figures.py \
  --sweep results/logistics_sweep_summary_native.json \
  --warmup-dir output/warmup/EUR_USD \
  --logistics-state output/planbench_by_domain/logistics/invalid_state_logistics_native.json \
  --synthetic results/qfh_synthetic_native.json \
  --planbench-native output/planbench_by_domain/logistics/gold_state_logistics_native.json \
  --fx-manifolds output/manifolds_native \
  --bridge-metrics docs/note/bridge_metrics.json \
  --outdir docs/figures \
  --note-dir docs/note
```

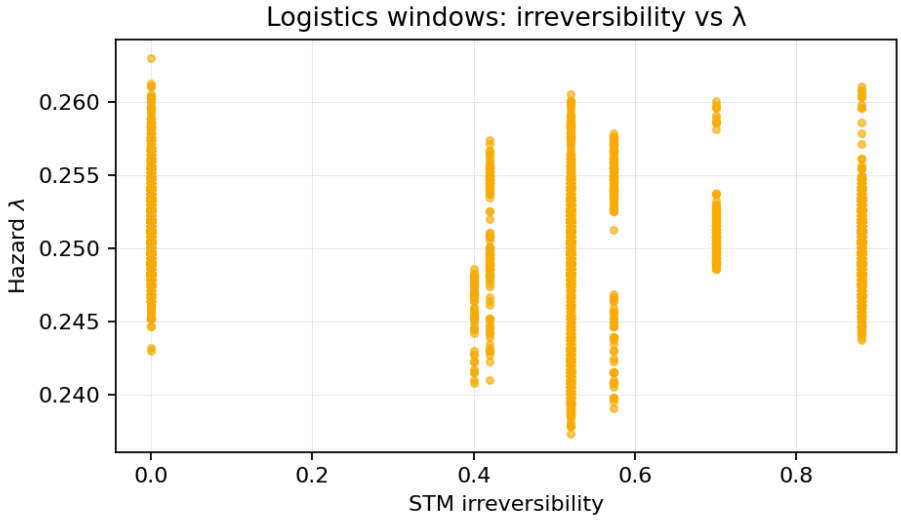


Figure 7: Logistics windows: STM irreversibility vs QBSA hazard.

Bridge correlation summary
Pearson r = -0.230 (p=0.000)
Spearman ρ = -0.280 (p=0.000)
Median hazard threshold = 0.2505

Figure 8: Correlation summary from `compute_bridge_metrics.py`.