

CVNTrade — Design documents

Work sample. Two design documents, reproduced as written, from a proprietary real-time trading platform (50+ crypto assets, several venues, continuous operation) that I built and operated as founder, architect and sole developer. The only edits are the removal of repository links.

Case summary

Problem. Crypto markets were fragmented across 50+ assets and multiple venues, volatile and continuously active, requiring a real-time foundation and a disciplined way to determine whether research signals could survive fees, slippage, missing data and partial fills. Training precision was ~ 0.20 versus ~ 0.25 break-even, while paper trading exposed tail slippage, outages, latency spikes, rate limits and clock drift.

Action. Built the signal-to-execution platform and replaced accuracy-led evaluation with time-ordered, out-of-sample economic validation, learning curves and net-return thresholds based on measured fees and slippage. Kept execution in controlled paper-trading with circuit breakers, freshness guards, depth-aware sizing, idempotent orders and position reconciliation. Developed a versioned agentic studio with human-only gates, ADRs, automated re-review, exact CI and documentation builds, enabling one person to deliver at team scale without automating consequential decisions.

Result. Found negative expected net return at every threshold, recalibrated the cost model upward from observed fills and withheld significant capital rather than deploy an unprofitable signal. Over roughly 12 months, delivered 400,000+ lines across 1,000+ pull requests while reducing idea-to-merge time from weeks to days; automated review findings typically converged from 8 to 3 to 2 to 0, with CI, tests, lint and documentation tracked per change. The result was a safer, continuously supervised platform — not an unsupported trading track record.

How design documents work on this platform

Every structural change starts as a design document: problem restated, goals and non-goals, architecture, invariants, observability, failure-mode coverage, alternatives rejected, open questions, acceptance criteria. The document is submitted to a multi-persona expert review committee (architect, ML engineer, trader, operations, data scientist); each persona scores and dissents are arbitrated explicitly in the next version. A passed design produces Architecture Decision Records (ADRs) that later code must conform to, and the implementation is gated by automated review, exact CI and strict documentation builds. Merges, committee initiation and any move toward real capital remain human actions.

The two documents

1. **Parity certification between the vectorised research path and the streaming production path** — committee-reviewed, passed (average 8.9/10). It answers a question most trading research avoids: do hyperparameters tuned on the fast batch path transfer, bit for bit, to the code that actually trades? The design proves equivalence stage by stage and makes parity a permanent CI gate.
2. **System-wide trading kill-switch** — design submitted for committee review (draft stage), with invariants later codified in an ADR. It shows the control side: single source of truth in the database, multi-channel activation, fail-safe on connectivity loss, operator-only disengagement, sub-second halt latency, full audit trail.

Document 1 is reproduced first because it is the more mature of the two; document 2 is included because it shows what a design looks like at the moment it is handed to the committee.

Document 1 — FTF ↔ Runner Parity Certification — Design document

Status: v2 — committee-reviewed PASSED (Architect 9.0, ML Engineer 9.0, Crypto Trader 9.0, Ops 9.0, Data Scientist 8.5 — avg **8.9/10**, strong consensus). 7 recommendations integrated below. **Authors:** CVNTrade research, 2026-04-21

Sibling issues: #599 (StatefulEnrichment — provides per-feature equivalence-by-construction), #568 (Pipeline runner Phase 1+ — provides the streaming execution path), #614 (execution track for this doc) **Prerequisite:** #599 design v3 PASSED (2026-04-20, committee session). The parity guarantee proposed here composes on top of the stateful refactor.

Changelog

- **v2 (2026-04-21)** — post committee review, PASSED. 7 recommendations integrated:
 1. Tolerance ladder: kept `1e-9` / strict, added per-feature documentation requirement in §4 P4 + inventory.
 2. Vectorised with proof: canonical definition must be **simple and direct** (explicit rule in §4 P3).
 3. E2E fixture: expanded to **3 cryptos (BTC/ETH/SOL) × {quiet, volatile, stress}** scenarios (§5 Eq. 5).
 4. Phase E governance: joint sign-off DRI + senior MLOps/SRE lead or designated committee member (§6 Phase E).
 5. Monolith deprecation: soft-deprecate 2–4 weeks after harness stable, then full removal (new §6 Phase F).
 6. Scoping: folded into #599 Phase 2 with **dedicated engineer owning the parity suite** (§9).
 7. New §11bis “Blind spots addressed” covering all 7 committee-flagged angles:
 - Upstream OHLCV trust boundary + monitoring
 - Model artifact serialisation/deserialisation bit-identity
 - Runtime environment consistency (OS/Python/lib versions)
 - Training determinism (HPO seeds → bit-identical artifacts, ADR-23)
 - Timestamp/timezone consistency (DST, tz boundaries)
 - Property-based testing (Hypothesis) for vectorised features
 - Filter starvation (ADR-45)
- **v1 (2026-04-21)** — initial draft submitted to committee.

1. Executive summary

CVNTrade faces a structural dilemma between **speed of experimentation** and **fidelity to production**:

- **Vectorised batch backtest** — fast (~25–30 min per FTF run), powerful for multi-lever sweeps, but uses a code path that is **not the one trading in production**.
- **Streaming pipeline runner** — matches paper/live behaviour, but 2–5× slower per backtest, turning a 10-crypto FTF cycle into tens of hours.

At today’s FTF volume (~45 cells × 15 HPO trials × multiple levers × re-runs on config changes), running the tuning in streaming mode is not economically viable. Running it in batch mode raises a fundamental question: **do the hyperparameters tuned on the batch path transfer to what will actually trade in production?**

The usual answer — “they should, same code underneath” — is a **foi empirique**, not a demonstration. This document specifies a formal alternative: prove equivalence at each atomic stage (features, inference, filters, execution), and by composition prove that a batch run and a streaming run on the same data produce **bit-identical** trading decisions. The FTF can then stay vectorised with mathematical certainty, and the migration to the runner at the end of tuning is a no-op by construction, not a hope.

Total effort for the certification harness: **~1 week senior engineer**, once. Permanent CI gate afterwards — any regression on parity blocks merge.

2. Problem statement

2.1 The dilemma (quantified)

Empirical measurements on FTF runs of 2026-04-20:

Config	Duration	Notes
Baseline (tw=18, shadow=off)	~30 min / run	vectorised backtest only
Lever #2 (tw=9, shadow=on)	~3h30 / run	monolith + streaming in parallel, enrichment rolling 500-bar window dominates
Baseline-ref (tw=18, shadow=off)	~56 min / run	vectorised only, with autocal=0

Projected full FTF cycle on defi_top5:

- 5 levers × 3 variants per lever × 10 cryptos × 5 folds × 15 HPO trials = ~11 250 cells
- At 25 min / 45-cell run: **~100 h of vectorised compute** for one full sweep
- At 5× streaming overhead: **~500 h of streaming compute** — untenable

We cannot do FTF in full streaming mode. We must do it in batch. So we need a certificate that the batch result transfers.

2.2 What “transfers” must mean

A lever conclusion like “isotonic beats platt by 8% on Sortino net-15bps” is only actionable if the winner will actually produce that Sortino when it trades. If the batch path and the streaming path produce different trade sequences on the same OHLCV, the FTF ranking may not transfer.

The certification claim we want: given a fixed (model, hyperparams, feature config), the **trade sequence** produced by the streaming pipeline runner on a given OHLCV window is bit-identical to the trade sequence produced by the vectorised batch path on the same window.

From there, Sortino, PnL, f1_buy, action_rate — all downstream metrics — are equal by construction.

2.3 What “bit-identical” means precisely

- **Feature values:** equal at machine precision ($\leq 1e-12$ relative) or differ only by documented float non-associativity with bounded-magnitude tolerance ($\leq 1e-9$ relative, per-feature).
- **Inference outputs** (p_buy per candle): same tolerance.
- **Trading signals** (signal $\in \{\text{HOLD}, \text{BUY}\}$ per candle): **strictly equal**, no tolerance.
- **Trades** (entry bar, entry price, exit bar, exit price, pnl): **strictly equal**.
- **Aggregate metrics** (Sortino, f1, etc.): equal by construction if the trade list is equal.

Strict equality at the signal level is the key contract — because any signal-level divergence triggers a different trade and cascades.

3. Complexity analysis

3.1 Training

Both modes train the same model on the same training data (assuming the #596 fix lands). Training is **not a parity concern** — it produces a deterministic artifact from an OHLCV window + hyperparameters + seed.

3.2 Backtest

Stage	Batch (vectorised)	Streaming (runner)
Feature generation	O(T·F) pandas/numpy, SIMD-friendly, well-amortised	O(T) candles × O(1) update + O(1) emit — slower constant factor (Python interpreter)
Inference	<code>model.predict_proba(X_batch)</code> — single call, batched	<code>model.predict_proba(X_single)</code> per candle — N function calls
Decision loop	O(T) inherently sequential (positions, cooldown, drawdown gate)	identical — both are already sequential here
Execution engine	O(T) sequential	identical

Observation: the decision loop and the execution engine are **already sequential in both modes**. The batch optimisation applies only to feature generation and inference. So the parity concern narrows to those two stages.

3.3 Inference (paper/live)

Streaming only. O(1) per candle. No batch counterpart — but this also means inference parity is a property of the trained model + feature vector, not of the execution mode. As long as features are equal at T, predictions are equal.

4. Design principles

P1 — Separate “value computation” from “decision loop”

The decision loop (filter chain + execution) is inherently sequential in both modes. We do not try to vectorise it. The parity concern lives strictly in feature generation and inference, which are the only stages where batch wins on speed.

P2 — Prove atomic equivalence, then compose

Instead of proving `batch_pipeline` $\equiv$ `stream_pipeline` globally (intractable), we prove each atomic component equivalent: - `feature_i_batch` $\equiv$ `feature_i_stream` for every i - `inference_batch` $\equiv$ `inference_stream` - `filter_chain_batch` $\equiv$ `filter_chain_stream` (trivially — same code, both sequential) - `execution_batch` $\equiv$ `execution_stream` (trivially — same code)

By composition: if all atomic pairs are equivalent, the full pipelines are equivalent.

P3 — Vectorised path allowed, but certificated

A vectorised implementation of a feature can exist (and should, for speed), but **only if accompanied by a test proving it equivalent to the canonical update+emit definition**. The canonical definition is the streaming path (#599's `update + emit` contract); the vectorised path is an optimisation, not a source of truth.

Committee v2 reinforcement: the canonical definition must be **as simple and direct as possible** — a naive for-loop implementing the feature's mathematical definition line-by-line. Any cleverness (memoisation, batch shortcuts) lives in the vectorised path, proven equivalent. The canonical is the ground-truth oracle, not the performance path. If the canonical is hard to read, the certification chain collapses.

```
def update_ema(state, candle, alpha): ...      # canonical
def emit_ema(state): ...                      # canonical

def batch_ema_vectorised(ohlc, alpha):
    return ohlc['close'].ewm(alpha=alpha, adjust=False).mean().values

def test_ema_vectorised_equals_sequential():
    # property-based test or fixed fixtures
    assert np.allclose(
        batch_ema_vectorised(ohlc, alpha=0.1),
        batch_ema_from_update_emit(ohlc, alpha=0.1),
        atol=1e-12,
    )
```

If no vectorised form can be proven equivalent for feature X, we **keep the loop** for that feature. The cost is localised — we don't lose the SIMD gains on all other features.

P4 — Bit-identical on signals, float tolerance on continuous values

The tolerance ladder is explicit: - Continuous values (feature outputs, probabilities, PnL): `atol=1e-9` relative (default) - Discrete outputs (signal, trade list): strict equality, no tolerance - Aggregate metrics: equal by construction from equal trade lists

Committee v2 refinement: any feature requiring a tolerance looser than `1e-9` must carry a **documented justification in documentation/architecture/ENRICHMENT_INDICATORS.md** (the per-feature line gains a `tolerance_override` column with reason — e.g., *“rolling quantile uses pandas linear interpolation, deviates by 2e-9 on denormalised ranges; proven safe by ...”*). Where feasible, tighten to `1e-12` — the default `1e-9` is a ceiling, not a target. No per-run ad-hoc loosening; tolerance is a versioned feature property.

P5 — CI-gated, not negotiable

All parity tests run on every PR. Breaking parity is a blocker, same level as a failed unit test. No exceptions — a regression on parity is a regression on the scientific foundation of the entire FTF → Production workflow.

P6 — No “empirical proximity”

We explicitly reject claims of the form “the two produce similar enough results”. Either the atomic equivalences hold (and composition gives us bit-identity by construction), or they don't (and we fix the component). There is no middle ground in CI.

5. Architecture — the 4 + 1 equivalences

Eq. 1 — Feature equivalence (per-feature)

Per the #599 taxonomy (Recursive / RingBuffer / Stateless):

For every feature, prove:

```
∀ T, batch_emit(ohlc[:T+1]) == stream_emit ◦ (stream_update)^T (bootstrap(ohlc[:warmup]))
```

Tests: - **Unit per-feature** (`test_feature_<name>_batch_stream_parity`): fixed-seed random OHLCV, run batch, run stream, assert last-value equal within tolerance. Hypothesis-based property tests for extra coverage. - **Sweep over T:** not only the last bar — assert parity at 10 random intermediate T values to catch mid-window drift.

Eq. 2 — Inference equivalence

```
∀ X, i, model.predict_proba(X)[i] == model.predict_proba(X[i:i+1])[0]
```

Standard property of sklearn / XGBoost models when features are stateless at the model level (no batch-norm, no recurrent layer). Test: batch 1000 rows, compare to 1000 individual calls, assert element-wise equality.

Eq. 3 — Filter chain equivalence

Each plugin is either: - Stateless per candle (trend, cost, kelly) → trivially batch=stream - Stateful with explicitly-versioned state (cooldown ticker, concurrency tracker, regime tracker) → update the state the same way in both modes

Test: feed a deterministic signal sequence of 500 candles; run through `FilterChainExecutor` in batch accumulating mode vs candle-by-candle mode; assert the output signal sequence is strictly equal.

Eq. 4 — Execution engine equivalence

Given the same signal sequence and the same OHLCV, entry prices, SL/TP exit logic, fees, and slippage are deterministic arithmetic on the inputs.

Test: fixed signal fixture (~50 signals), run `cvntrade_backtest_engine.execute` in both modes, assert trades list bit-identical (entry_bar, entry_price, exit_bar, exit_price, exit_reason, pnl).

Eq. 5 — End-to-end integration certificate

On real OHLCV windows pinned as Parquet fixtures in `tests/fixtures/parity/`. **Committee v2 expansion:** the fixture covers 3 cryptos × 3 scenarios (9 test cases) to broaden market-condition coverage:

Crypto	Scenario	Period	Rationale
BTC	Quiet	2024-09 (mean vol)	baseline case
BTC	Volatile	2024-11 (post-halving rally)	broad swings, many CUSUM events
BTC	Stress	2022-06 (Terra/3AC)	gap days, cross-timezone incidents
ETH	Quiet	2024-07	different asset characteristics
ETH	Volatile	2024-03 (shanghai upgrade)	regime shift
ETH	Stress	2022-05	correlated with BTC stress
SOL	Quiet	2024-08	lower-cap asset
SOL	Volatile	2023-11 (FTX rally)	high-vol small-cap
SOL	Stress	2022-11 (FTX collapse)	extreme volatility + liquidity gaps

```
@pytest.mark.parametrize("crypto,scenario", FIXTURES)
def test_e2e_parity(crypto, scenario):
    ohlcv = load_fixture(crypto, scenario)      # pinned Parquet, ~2000 bars
    model = load_fixture_model(crypto)         # pinned MLflow artifact

    batch_trades, batch_metrics = run_batch_pipeline(ohlcv, model, config)
    stream_trades, stream_metrics = run_stream_pipeline(ohlcv, model, config)

    assert batch_trades == stream_trades        # strict
    assert batch_metrics.n_trades == stream_metrics.n_trades # strict
    assert abs(batch_metrics.sortino - stream_metrics.sortino) < 1e-9
    assert abs(batch_metrics.total_return - stream_metrics.total_return) < 1e-9
```

Aggregate runtime target: < 90 s for the 9 cases (parallelised). Part of pre-merge CI. If runtime grows past 90 s as features accrete, either parallelise harder or rotate scenarios weekly (always 3 cases per PR, full 9 nightly).

6. Rollout plan

Phase A — Per-feature parity tests (2 days)

Extends the #599 test suite. One test per feature class (R / B / S / T). Leverages the `update / emit` contract that #599 already introduces. For vectorised variants, add a parity test against the sequential reference.

Exit criterion: every feature in the enrichment inventory (`documentation/architecture/ENRICHMENT_INDICATORS.md`) has a passing parity test.

Phase B — Inference + filter chain + execution parity (1.5 days)

Three targeted tests, independent of each other. Small surface, quick to write.

Exit criterion: the three parity tests run in CI and pass.

Phase C — End-to-end integration harness (2 days)

Builds the fixture (a pinned 2000-bar OHLCV window from BTC/ETH/SOL, stored as Parquet in tests/fixtures/), implements the side-by-side runner, writes the assertions.

Exit criterion: E2E test runs in CI on every PR, < 60 s, passes on main.

Phase D — CI gate + documentation (1 day)

- CI config: parity tests run on every PR, failing = merge blocked
- Update CLAUDE.md with the “parity contract” section
- Write the operator-facing runbook: “what to do if a parity test fails”

Exit criterion: parity test results visible in PR checks, documented recovery path.

Phase E — Certification ceremony at FTF close (ad-hoc, ~1 hour)

When FTF P0-A is closed and we’re ready to move to production:

1. Run the full E2E parity test on the *exact* config snapshot that was selected at the end of FTF.
2. Run the streaming pipeline runner on the top-3 crypto × test-window slices that drove the decision.
3. Confirm trade lists and metrics are bit-identical to the batch results FTF used.
4. **Joint sign-off** (committee v2 recommendation 4) — the certification document requires two signatures:
 - **DRI** (Directly Responsible Individual) for the MLflow backbone / FTF flow
 - **Senior MLOps / SRE lead** (or a committee-designated member acting as governance reviewer) No solo sign-off. Shared accountability technical + operational + governance.

Exit criterion: a one-page certification document (timestamp, config git_sha, fixture SHAs, test outputs, two signatures) archived in `committee/sessions/` as the production-ready bill of health.

Phase F — Monolith deprecation (1 sprint after Phase E, committee v2 recommendation 5)

Once the certification harness is fully live and stable in CI (**Phase D exit + Phase E first successful execution**):

- **Week 0:** `src/backtest/cvntrade_backtest_engine.py` gets a `DeprecationWarning` at import — 2-4 week soft-deprecation window starts.
- **During the soft-deprecation window:**
 - The pipeline runner is the declared primary; the monolith remains callable for comparison / rollback.
 - Production FTF / paper / live do **not** call the monolith.
 - CI parity tests stay green (regression monitor on the runner).
 - Any warning-on-import hit during the window triggers a call-site investigation.
- **End of window** (2-4 weeks, operator decision):
 - If no warnings hit and parity tests have been green continuously: full removal of `cvntrade_backtest_engine.py` and all its callers.
 - If any warning hit: investigate, extend window, or open a regression issue.

Exit criterion: monolith code removed from repo, all historical references migrated to `git log` archaeology only.

7. Validation criteria

Automated (CI)

- ☐ All per-feature parity tests pass (classes R / B / S / T from #599 inventory)
- ☐ Inference parity test passes
- ☐ Filter chain parity test passes
- ☐ Execution engine parity test passes
- ☐ End-to-end integration parity test passes (on the pinned fixture)

Operational

- ☐ Parity tests run in ≤ 90 s total (not blocking CI throughput)
- ☐ Clear failure messages: on test failure, output which feature / stage diverged, with first-divergence candle index (same pattern as the shadow diagnostic in #599)

Process

- ☐ Pre-merge CI gate active
 - ☐ Documentation in CLAUDE.md + operator runbook
 - ☐ Phase E ceremony executed once before P0-A conclusion lands in production
-

8. Risks

Risk	Mitigation
A feature has no feasible vectorised equivalent → forced to stay in the slow path	Acceptable if localised; the FTF stays fast on the 99% of features that do vectorise. Documented in the inventory per feature.
Float non-associativity causes spurious test failures	Per-feature tolerance calibration ($\leq 1e-9$ relative, documented in the test). If a feature consistently requires a higher tolerance, the tolerance is recorded in its inventory entry + justified.
Parity test runtime grows as features are added	Test parallelisation; keep per-test runtime < 1 s; the aggregate can grow linearly with feature count without impact.
Developer bypasses CI by marking parity tests as <code>xfail</code> to merge fast	Governance: <code>@pytest.mark.xfail</code> on a parity test requires a one-line reference to a tracking issue that commits to removing the xfail within a defined sprint. Reviewed in CR.
E2E test depends on a fixture that rots (OHLCV changes, exchange delisting)	Fixture is pinned (Parquet in-repo), does not fetch from exchange at test time. Independent of external state.
False sense of security if parity tests pass but FTF config differs from Phase E config	The Phase E ceremony explicitly re-runs parity on the exact config that is being promoted — not on a historical one.

9. Interfaces with in-flight work + scoping (v2)

- **#599 StatefulEnrichment**: provides Eq. 1 largely by construction (the `update/emit` contract is a shared definition). This document extends #599's §7 validation with formal per-feature tests.
- **#568 Pipeline runner Phase 1+**: provides the streaming side of the equation. Eq. 2-5 require the runner being callable as a library in tests (already the case).
- **#596 train_window fix** (merged as commit 0a1cdc71): pre-requisite, [OK] done. The “same config → same training” assumption is now guaranteed.
- **#608 ThresholdCalibrator**: orthogonal. The calibrator lives inside `apply_thresholds` and is version-pinned in MLflow — equivalent in both paths by construction.
- **#612 Inference gate audit**: orthogonal. Confirms that the decision gate location is consistent between paths, which is a precondition of Eq. 2.

Scoping (committee v2 recommendation 6)

Folded into #599 Phase 2 as a corollary — shared test suite, same `update/emit` contract, same week of wall-clock. BUT with an explicit constraint:

- A **dedicated engineer** owns the parity test suite within the #599 scope. Not a junior, not a rotation slot. The same person drives E1 (feature parity) and E2-5 (inference / filter / execution / E2E) — so the test architecture stays coherent and the composition proof isn't split across contributors.
- Naming this person is part of the Phase A kickoff for #599. Without a named owner, the parity track becomes an afterthought folded into unrelated PRs, which is exactly what we're trying to avoid (see Phase 2.9a refactor's leftover surface that produced #612 last week).

10. Decisions taken (v2 — was “open questions” in v1)

All v1 open questions are resolved post committee review. Kept here as a decision log.

1. **Tolerance ladder** (§4 P4) — **DECISION**: $1e-9$ relative on continuous / strict on discrete. Any looser tolerance requires a per-feature justification in the enrichment inventory. Default is a ceiling, not a target; tighten to $1e-12$ where feasible.
2. **Vectorised path allowed with proof** (§4 P3) — **DECISION**: allowed with mandatory parity test. Additional rule: canonical definition must be simple and direct (committee v2 rec. 2).
3. **Fixture for E2E** (§5 Eq. 5) — **DECISION**: 3 cryptos × 3 scenarios = 9 cases, pinned Parquet, < 90 s aggregate CI runtime. Rotate nightly if needed (committee v2 rec. 3).
4. **Phase E governance** (§6 Phase E) — **DECISION**: joint sign-off DRI + senior MLOps/SRE (or committee-designated). No solo sign-off (committee v2 rec. 4).
5. **Monolith deprecation** (§6 Phase F) — **DECISION**: soft-deprecate 2-4 weeks after Phase D stable + Phase E first success, then remove. Safer than immediate removal, cheaper than keeping as permanent fallback (committee v2 rec. 5).
6. **Scoping vs #599** (§9) — **DECISION**: folded into #599 Phase 2 with dedicated engineer named at kickoff (committee v2 rec. 6).
7. **Blind spots** — 7 covered explicitly in new §11bis (committee v2 rec. 7).

Remaining open questions (for future iterations)

1. **Parity test runtime ceiling:** aggregate target is < 90 s today. As the feature inventory grows, at what point do we pivot from “all 9 scenarios per PR” to “3 per PR + 9 nightly”? Proposal: trigger the pivot at aggregate > 180 s, monitored via CI metrics.
2. **Continuous parity monitoring in production:** the Phase E certification is a one-shot ceremony. Should we extend with continuous parity sampling (e.g. 1% of live trades shadow-run through the batch path and compare)? Not in scope of v1 implementation, tracked as a potential Phase G.

11. Acceptance criteria for this plan document

- ☒ Problem statement quantified with real FTF durations
- ☒ Complexity decomposition across training / backtest / inference
- ☒ Six design principles (P1-P6) explicit
- ☒ Four atomic equivalences + one integration equivalence specified with concrete test shapes
- ☒ Rollout in 6 phases (A-F) with exit criteria
- ☒ Risks + mitigations
- ☒ Interfaces with #599, #568, #596, #608, #612 + scoping (§9)
- ☒ Decisions converted to firm commitments (v2, §10)
- ☒ Blind spots addressed (v2, §11bis — 7 committee-flagged angles)
- ☒ **Committee review PASSED** (scores 9.0 / 9.0 / 9.0 / 9.0 / 8.5, avg 8.9/10, strong consensus)

Plan is now the spec of child issue **#614 — FTF-to-Runner Parity Certification**, executed as a corollary of #599 Phase 2.

11bis. Blind spots addressed (committee v2 rec. 7)

Seven angles flagged by the committee. Each becomes an explicit test or operational control.

11bis.1 Upstream OHLCV trust boundary

Concern: the parity proof assumes “same OHLCV in” → “same everything out”. But if the data ingestion path between batch and live produces even slightly different OHLCV (rounding, exchange aggregation, late bars), the certificate is vacuous.

Control: - Document the trust boundary explicitly: parity starts at the OHLCV DataFrame handed to the enricher. Anything upstream (ETL, Feast, exchange adapter) is out of scope of this document but must satisfy its own parity (tracked under a separate observability issue). - Add a **smoke test** at pipeline entry: checksum the first 100 rows of OHLCV ingested by each path; emit `event=ohlcv_checksum` to Loki. Drift → alert on the shadow dashboard. - The Phase E ceremony explicitly reads the OHLCV checksum from both the FTF run that selected the config AND the streaming run used for certification — they must match.

11bis.2 Model artifact serialisation / deserialisation bit-identity

Concern: a trained XGBoost model saved in environment A, loaded in environment B, might produce slightly different predictions due to library version differences or serialisation-format drift.

Control: - New unit test `test_model_save_load_bit_identical_inference`: `model = train(X_train, y_train)` `preds_before = model.predict_proba(X_test)` `save_to_mlflow(model, path)` `loaded = load_from_mlflow(path)` `preds_after = loaded.predict_proba(X_test)` `assert np.allclose(preds_before, preds_after, atol=0)` # STRICT - Plus a cross-environment variant: save on CI, load in a fresh container, compare. Run nightly.

11bis.3 Runtime environment consistency

Concern: OS, Python version, and key library versions (numpy, pandas, XGBoost, scikit-learn) drifting between batch (CI runner) and production (airflow pod or runtime pod).

Control: - Document the “runtime contract” in `documentation/runtime_contract.md`: pinned versions of numpy, pandas, XGBoost, LightGBM, CatBoost, scikit-learn, Python. - CI uses the **same base Docker image** as production (`airflow_docker/Dockerfile.k8s`), not a lighter CI image. Shared Dockerfile = shared runtime invariants. - Pre-merge CI gate: any `pip install` or dependency-change PR must pass the full parity suite (not just linted).

11bis.4 Training determinism

Concern: HPO with different seeds or non-deterministic parallelism produces different hyperparameters → different models → different predictions → parity violated upstream of the harness.

Control: - ADR-23 (version-pinned MLflow artifacts) already enforces this at the artifact level. Extension: a new test `test_training_determinism`: `model_a = train(ohlcv, hpo_seed=42)` `model_b = train(ohlcv, hpo_seed=42)` # re-run with

`same seed assert serialise(model_a) == serialise(model_b) # byte-identical` - All HPO configs must fix CVN_HPO_SAMPLER_SEED (already the case via `regime_trainer.py:525`). - Any future introduction of GPU training must document determinism controls or be gated out.

11bis.5 Timestamp / timezone consistency

Concern: OHLCV timestamps in batch may be tz-aware while a live stream delivers naïve tz — features that depend on `dt.hour` or `dt.dayofweek` would diverge around DST transitions.

Control: - CLAUDE.md already mandates tz-aware UTC indexes. Extension: add an explicit assertion in the E2E fixture loader: `assert ohlcv.index.tz is not None assert str(ohlcv.index.tz) == "UTC"` - Dedicated parity test `test_cyclic_features_across_dst_transition`: a fixture covering an autumn DST event, assert `hour_sin`, `hour_cos`, `day_sin`, `day_cos` are bit-identical across both paths.

11bis.6 Property-based testing for vectorised features

Concern: hand-crafted fixed fixtures may miss edge cases (zero-variance windows, all-NaN stretches, discontinuities).

Control: - All per-feature parity tests (Eq. 1) gain a **Hypothesis strategy** that generates randomised but well-formed OHLCV sequences (monotonic timestamps, positive volumes, bounded prices). - Runtime budget: 10 Hypothesis examples per feature per CI run. Nightly: 100 examples per feature. - Any finding shrinks to a minimal failing input; the failing fixture gets pinned as a fixed-fixture test for fast regression coverage afterwards.

11bis.7 Filter starvation (ADR-45)

Concern: filters with internal state (cooldown, concurrency) may exhibit different behaviour under rapid-fire signal bursts vs sparse signals. A batch path may see a burst collapsed into a single pandas operation while streaming sees the same burst as sequential discrete decisions.

Control: - Dedicated test `test_filter_chain_under_starvation`: a signal sequence with (a) a 20-candle burst of BUYS, (b) a 500-candle drought, (c) another 20-candle burst. Run through the filter chain in both modes. Assert strictly-equal output signal sequence. - The cooldown and concurrency plugins are already explicitly ADR-45-aware; this test pins the behaviour as regression-proof.

12. References

- **Parent track:** #568 (pipeline runner), #599 (enrichment refactor)
- **Prereq PR:** #613 (train_window fix) — must merge before Phase E ceremony
- **Related ADRs:**
 - ADR-40 (paper/live same kernel) — this document is the formal enforcement mechanism
 - ADR-23 (version-pinned MLflow artifacts) — supports the assumption that “same config → same model”
 - ADR-25 (no silent fallback) — parity test failure must never be silent
- **Prior art:**
 - Google’s “Training-Serving Skew” chapter in *Reliable ML* (Kreuzberger et al.) — same problem, different domain
 - Deep learning: TF Serving’s “same graph at train and serve” — structurally analogous
 - Ray Train + Ray Serve parity tests — industry pattern we’re adapting

13. Committee submission — CLOSED, PASSED

v1 submitted 2026-04-21. Verdict: **PASSED** (strong consensus). - Architect: 9.0 / 10 - ML Engineer: 9.0 / 10 - Crypto Trader: 9.0 / 10 - Ops: 9.0 / 10 - Data Scientist: 8.5 / 10 - **Average: 8.9 / 10**

All 7 recommendations integrated in this v2 (see §Changelog at the top). No resubmission planned unless the v2 revisions themselves require review.

Archived artefact: `committee/sessions/` (session json to be committed alongside implementation PR).

14. (historic) v1 Submission template — kept for reference

Title: FTF ↔ Runner Parity Certification — plan review

Question (target score: 8+):

Review of `documentation/.../design/CVN-N005-parity-certification.md` before we fold this work into the #599 stateful refactor scope. The underlying problem is the classic training-serving skew recast as FTF-vs-production: we must tune FTF in batch for speed but need mathematical certainty that the results transfer to the streaming runner.

The plan codifies six design principles (§4), four atomic equivalences + one integration (§5), and a 5-phase rollout (§6).

Committee review requested on:

1. **Tolerance ladder** (§4 P4): $1e-9$ relative on continuous, strict on discrete signals and trades. Right bar? Too loose? Per-class justification preferred?
2. **Vectorised allowed with proof** (§4 P3): we allow vectorised feature implementations as long as they carry a parity test against the canonical `update/emit` definition. Alternative is banning them outright for safety, paying a 3-5x runtime penalty. What's the committee's tradeoff?
3. **Fixture selection for Eq. 5** (§5): one quiet + one volatile BTC window? Three cryptos? The fixture is the basis of the pre-merge gate — the choice shapes what "parity" means in practice.
4. **Phase E ceremony governance** (§6 Phase E): who signs off on the certification before production? Operator alone, DRI, committee?
5. **Monolith deprecation** (§9): once parity is certified, the monolith backtest engine is redundant. Immediate removal, soft-deprecate, or keep as a third reference path forever?
6. **Relationship to the enrichment refactor #599**: we propose folding this work into #599 Phase 2 as a corollary (same engineer, same test suite). Is that the right scoping, or should it be a sibling issue with its own engineer?
7. **Blind spots**: what's the most expensive mistake we could make executing this plan that isn't surfaced in §8 Risks?

Artifacts referenced: - `documentation/design/CVN-N005-parity-certification.md` (this doc, ~400 lines) -
`documentation/design/CVN-N005-stateful-enrichment.md` v3 (committee PASSED 2026-04-20) -
`documentation/architecture/ENRICHMENT_INDICATORS.md` (Phase 0 deliverable)

Deliverable expected: per-expert opinion (score, confidence, findings, risks, recommendations) + consolidated verdict (PROCEED / REVISE / REJECT). If PROCEED, child issue #614 opened and folded into #599 scope.

Document 2 — Design — System-wide trading kill-switch

(CVN-N001-EF-S02)

story_id: CVN-N001-EF-S02 **epic_id:** CVN-N001-EF (#729) **need_id:** CVN-N001 (F1 mission) **Authors:** Dominique (operator) + Claude **Reviewers:** Expert Committee (plan_review mode B) **Status:** draft **Created:** 2026-04-28 **Last updated:** 2026-04-28

Changelog

Date	Change
2026-04-28	Initial draft for committee review

1. Problem restated

Per F1_buy boost committee finding (session 8db2529d, Ops persona) : > *Implement a System-Wide Kill-Switch: Develop and integrate an explicit, easily accessible mechanism to immediately halt all trading activity or revert the system to a predefined safe state in an emergency.*

Today the project has **partial** kill-switch primitives :

- RiskManager.activate_kill_switch(reason) at src/paper_trading/safety/cvntrade_risk_manager.py:201 — sets in-memory _killed=True
- Kernel checks _killed at src/paper_trading/cvntrade_paper_trading_engine.py:1667 (every ~1s tick)
- Pre-trade check rejects when killed at line 2225
- Auto-activation on global drawdown breach (line 175)
- One activation channel : Airflow DAG dag_pte__7_killswitch.py

Gap : the state is **per-process in-memory**. Multiple paper / live engine instances each hold their own _killed flag. Restart loses state. Activation requires triggering one specific Airflow DAG. There is no single source of truth, no Console button, no env-var safety, no Grafana auto-trigger, no audit table beyond Loki logs.

2. Goals (from #708, refined)

1. **Single source of truth** : kill-switch state lives in PostgreSQL ; every trading decision reads it (no per-process cache surviving > 500ms).
2. **Multi-channel activation** : Console UI button, CLI command, K8s env var (boot-time), Grafana alert webhook, existing Airflow DAG.
3. **Reversible** : disengagement uses the same channels (except env var, which requires a pod restart).
4. **Auditable** : every flip emits structured log + writes to history table + OpenTelemetry span.
5. **Fail-safe on connectivity loss** : if PG is unreachable for > N consecutive polls, behave as if engaged (do NOT trade in the dark).
6. **Halt latency < 1 s** : from operator click to pre-trade rejection.
7. **Operator-only disengagement** : auto-engage exists (drawdown / circuit breaker / connectivity) ; auto-disengage does NOT exist (prevents flapping).

3. Non-goals

- **Not a per-strategy / per-crypto pause** — that lives in the FTF factor toggles (ADR-56) or in ftf_config.cryptos.
- **Not a graceful drain by itself** — in-flight orders are NOT cancelled by the kill-switch transition. **However**, per committee session 4c388b4c BLOCKER, a companion **flatten_all** **Story is now MANDATORY before any live trading** — it ships in the implementation Epic CVN-N001-EG (see §7) and provides a separate channel (Console button + CLI) to cancel open positions on demand. Until **flatten_all** ships, this engine is allowed only in paper mode.
- **Not a partial halt** — there is no “pause new trades but allow exits”. Exits (TP/SL/timeout) continue to run on existing positions because that’s the safer behavior ; the switch halts **new BUY signals + new allocations**.
- **Not a per-account / per-API-key gate** — that’s exchange-side rate limiting / risk controls, out of our scope.

4. Architecture

4.1 Storage (PostgreSQL)

New table kill_switch_state (single-row, like ftf_config) :

```
-- infra/migrations/0XX_kill_switch.sql
CREATE TABLE IF NOT EXISTS kill_switch_state (
  id          INT PRIMARY KEY DEFAULT 1 CHECK (id = 1), -- single row
  engaged     BOOLEAN NOT NULL DEFAULT FALSE,
```

```

engaged_at    TIMESTAMPTZ,
engaged_by    TEXT,                                -- operator handle, "system:drawdown", "system:circuit_breaker", etc.
reason        TEXT,
lock_version  INT NOT NULL DEFAULT 0, -- optimistic concurrency
updated_at    TIMESTAMPTZ NOT NULL DEFAULT NOW()
);
INSERT INTO kill_switch_state (id, engaged) VALUES (1, FALSE) ON CONFLICT DO NOTHING;

CREATE TABLE IF NOT EXISTS kill_switch_history (
  id          BIGSERIAL PRIMARY KEY,
  transition  TEXT NOT NULL CHECK (transition IN ('engage', 'disengage')),
  actor       TEXT NOT NULL,          -- operator handle or "system:..."
  reason      TEXT,
  channel     TEXT NOT NULL,          -- 'console_ui' | 'cli' | 'env_var' | 'grafana_webhook' | 'airflow_dag' |
'auto_drawdown' | 'auto_circuit_breaker' | 'auto_pg_unreachable'
  occurred_at TIMESTAMPTZ NOT NULL DEFAULT NOW()
);
CREATE INDEX kill_switch_history_occurred_at_idx ON kill_switch_history (occurred_at DESC);

```

Why a dedicated table and not `ftf_config.base_env.kill_switch.*` : - `ftf_config` carries pipeline parameters (ADR-59 scope) ; mixing safety state into the same blob risks an operator editing the wrong key. - A separate table allows row-level audit trigger (`kill_switch_history`) without polluting `ftf_config_history`. - The 1-row check + `lock_version` makes concurrent writes from multiple channels safe.

4.2 Read path — KillSwitchClient

New module `src/commun/safety/kill_switch_client.py` exposing :

```

class KillSwitchClient:
    def is_engaged(self) -> bool: ...          # synchronous, < 10ms (in-memory)
    def state(self) -> KillSwitchState: ...    # full row (cached)
    def start_polling(self, interval_s: float = 0.5) -> None: ...
    def stop_polling(self) -> None: ...

```

- Background thread polls `SELECT engaged, engaged_at, engaged_by, reason FROM kill_switch_state WHERE id=1` every 500 ms (configurable).
- In-memory cache holds last successful read.
- **Fail-safe**: if N=3 consecutive polls fail (PG unreachable, exception, timeout > 200ms), the in-memory state flips to `engaged=True, engaged_by="system:pg_unreachable"`. A `kill_switch_history` row is written when PG comes back (deferred audit). Per ADR-25, a CRITICAL log fires on every fail-safe activation — no silent fallback.
- `KillSwitchClient` is a process-global singleton initialised in the trading kernel boot path.
- Deactivation polls the same way — PG flips back to `engaged=False`, next poll picks it up within < 500 ms, total worst-case halt-to-resume < 1 s.

4.3 Kernel integration

- `_main_loop()` at `paper_trading_engine.py:1667` replaces `self._risk_manager.is_killed()` with `self._kill_switch_client.is_engaged()`.
- `RiskManager.check_pre_trade()` at line 89 also reads from the client (defense in depth — caught at both layers).
- When engaged : skip `_generate_and_execute_signals()` (line 1693) ; **continue** `check_exits()` so existing positions can hit TP/SL.
- Boot-time check : if `CVN_KILL_SWITCH=engaged` env var is set OR the PG state is engaged, the engine logs a WARNING and starts in halted mode (does NOT exit ; the operator may disengage at runtime).
- **Friendly env-var error** (per committee reco 3): if `CVN_KILL_SWITCH` is set to any value other than `engaged` or empty (e.g. `disengaged`, `off`, `0`), the engine FAILS FAST at boot with an explicit message : `"CVN_KILL_SWITCH supports only 'engaged' (force-engagement at boot) or unset. Disengagement always goes through PostgreSQL — see ADR-71 I8."`. This addresses falsifiable hypothesis H5 (operator confusion).

4.4 Activation channels (write paths)

All channels POST to a single internal helper `engage_kill_switch(actor, reason, channel)` that : 1. Performs the optimistic-concurrency UPDATE on `kill_switch_state` (with `lock_version` check). 2. Inserts a row in `kill_switch_history` in the same transaction. 3. Emits structured log : `event=kill_switch_engaged actor=<...> reason=<...> channel=<...>` per ADR-32/33. 4. Emits OpenTelemetry span `kill_switch.engage` per ADR-62.

Channel inventory :

Channel	Surface	Backend	Auth	Use case
Console UI	red "EMERGENCY KILL" button at <code>scripts/ftf_config_ui.py</code> top bar (port 8501)	direct psycopg2 UPDATE	Streamlit session	operator at desk
CLI	make kill-switch ON REASON="..." / OFF (Makefile target → <code>scripts/kill_switch_cli.py</code>)	direct psycopg2 UPDATE	local shell + <code>OPENPROJECT_API_KEY</code> -style env	operator on phone via SSH

			(KILL_SWITCH_ACTOR=\$USER)	
K8s env var	CVN_KILL_SWITCH=engaged in pod spec, picked up at engine boot	bypasses PG ; engine refuses to start trading	K8s RBAC	restart all pods in safe mode (mass incident)
Grafana webhook	Grafana alert → AlertManager → POST /api/v1/kill-switch on a tiny FastAPI sidecar services/kill_switch_webhook/main.py	direct psycpg2 UPDATE	shared secret in Authorization header	auto-trigger on expectancy_net < 0 over 24h, pnl_drawdown > X%, etc.
Airflow DAG	existing dag_pte_7_killswitch.py refactored to call engage_kill_switch(...) instead of RiskManager.activate_kill_switch(...)	direct psycpg2 UPDATE	Airflow Connection	scheduled or manual trigger
Auto (system)	drawdown / circuit breaker / pg_unreachable	direct psycpg2 UPDATE (except pg_unreachable, which writes deferred)	n/a	safety-net

4.5 Disengagement (disengage_kill_switch)

Same shape as engagement. Only operator-initiated channels (Console UI, CLI, Airflow DAG) can disengage. Grafana webhook, env var, and auto channels CANNOT disengage (would create flapping). The fail-safe system:pg_unreachable clears automatically when PG comes back (with a kill_switch_history row recording the auto-clear).

5. Invariants (codified in ADR-71)

These MUST hold forever (or until ADR-71 is superseded) :

- **I1 — Single source of truth** : the only authoritative kill-switch state is the kill_switch_state row in PostgreSQL. No code path may set a process-local killed flag without polling PG.
- **I2 — Operator-only disengage** : auto channels can engage but never disengage. Disengagement requires a human actor named in engaged_by / kill_switch_history.actor.
- **I3 — Fail-safe on connectivity loss** : if PG is unreachable for > 3 consecutive polls (default 1.5 s), the client behaves as engaged and emits CRITICAL log + writes deferred history row when PG returns. Per ADR-25, no silent fallback.
- **I4 — No graceful drain** : engaging the switch does NOT cancel in-flight orders. It halts new BUY signals + new allocations only ; exit checks (TP/SL) continue to run.
- **I5 — Audit per flip** : every transition (engage / disengage / fail-safe / fail-safe-clear) writes a row in kill_switch_history AND emits structured log per ADR-32/33 AND emits OTEL span per ADR-62.
- **I6 — Halt latency < 1 s** : from successful engagement transaction to next pre-trade reject is bounded by polling_interval_s (default 500 ms) + one PG round-trip. Implementation MUST measure this via OTEL span and emit a metric.
- **I7 — Boot-time honors PG state** : on engine start, the kernel reads kill_switch_state once synchronously (before accepting any trade) ; only then does it start the polling thread. A True state at boot means the engine starts in halted mode without ever sending an order.
- **I8 — Env-var bypass is engage-only** : CVN_KILL_SWITCH=engaged env var forces engaged at boot regardless of PG. There is no CVN_KILL_SWITCH=disengaged ; disengagement always goes through PG (otherwise the env var would override operator decisions silently).

6. Observability

- **Grafana panel** “Kill switch state” : gauge (0/1) + timestamp of last flip + actor + channel.
- **Loki query** for audit feed : {job="kill_switch"} | json | line_format "{{.actor}} {{.channel}} {{.transition}}".
- **Prometheus metric** cvntrade_kill_switch_engaged{} (gauge 0/1) scraped from the FastAPI sidecar's /metrics.
- **Prometheus metric** cvntrade_kill_switch_pg_unreachable_total{} (counter) emitted by KillSwitchClient on every fail-safe activation. **Alert** : rate(cvntrade_kill_switch_pg_unreachable_total[1h]) > 0.1 (per committee reco 6 — validates H1 over time).
- **PostgreSQL load monitoring** (per committee reco 4) : add pg_stat_statements query for the kill_switch_state SELECT p99 latency. Alert if > 50 ms or if QPS × engine count overwhelms PG.
- **SLI** : kill_switch_halt_latency_seconds p99 < 1 s (alert when violated).
- **Console UI Quick Actions panel** (per committee reco 8) : top-bar widget with current state indicator + one-click engage button + CLI cheat sheet + clear disengage confirmation workflow (2-step : type “DISENGAGE” + reason text + confirm).
- **MLOps readiness §1** for any future Story modifying this surface MUST include a metric for the channel distribution (which channel triggers most).

7. Implementation plan (out of this Story's scope — future Epic)

This Story ships the DESIGN + ADR-71 only. Implementation lives in a future Epic CVN-N001-EG “Kill-switch

implementation”, broken into Stories following ADR-69 :

- EG-S01 : PG schema + KillSwitchClient + kernel integration + boot-time honor (the core)
- EG-S02 : CLI + Makefile target + Console UI Quick Actions panel (per committee reco 8)
- EG-S03 : Grafana webhook sidecar + AlertManager wiring
- EG-S04 : Airflow DAG refactor + auto fail-safe path
- EG-S05 : Observability (panel + SLI + metrics including pg_unreachable_total)
- **EG-S06 : flatten_all companion (cancel open positions on demand) — MANDATORY before live deployment per committee session 4c388b4c BLOCKER** (separate channel from kill-switch ; Console button + CLI ; integrates with exchange API)
- **EG-S07 : Chaos engineering harness for I3 (PG blips) + I7 (K8s pod kills) validation** (per committee reco 9 ; runs as part of pre-live gate)
- **Cross-cutting : operator runbook** in documentation/runbooks/kill_switch_<channel>.md for each channel (per committee reco 5) + ADR-70 amendment to add kill-switch-specific MLOps checks (per committee reco 7)

Each EG Story must fill TEMPLATE_mlops_readiness.md per ADR-70.

8. Failure mode coverage

Mapping each invariant to the failure modes it prevents :

Invariant	Prevented failure mode
I1 single PG source	“We restarted the pod and the kill-switch was forgotten”
I2 operator-only disengage	Auto-disengage flapping during transient PG hiccup
I3 fail-safe on PG loss	Trading silently in the dark when PG goes down
I4 no graceful drain	Operator confusion about “did the switch cancel my orders?” — no, by design
I5 audit per flip	“Who killed the engine at 3am and why?” — answer in kill_switch_history
I6 latency < 1 s	“I clicked but trades kept happening for 10 s”
I7 boot honors PG	Pod restart in middle of engaged window starts trading again
I8 env-var engage-only	An operator setting CVN_KILL_SWITCH=disengaged to bypass another operator’s engage decision silently

9. Alternatives considered (full list rejected)

- **Single in-memory flag, broadcast via Redis pub/sub** : faster (~10 ms latency) but introduces a second source of truth ; if Redis loses the message during reconnect, state diverges. PG poll is slower but simpler and survives any infra blip.
- **Filesystem-flag (e.g., /tmp/kill_switch)** : doesn’t survive pod restart, doesn’t work across pods.
- **K8s ConfigMap watch** : per-namespace blast radius is fine but ConfigMap eventual consistency under load can take seconds, and there’s no audit trail equivalent to kill_switch_history.
- **Exchange-side risk controls only** : doesn’t cover paper-trading or the in-house signal generation ; also doesn’t help when the bug is in our position-sizing code, not the exchange.
- **Cancel all in-flight orders on engage** : tempting but introduces a partial-state failure mode (some cancelled, some not) and adds exchange API dependency to the safety path. Operator handles via exchange UI.

10. Open questions for committee

The committee plan_review should explicitly answer :

1. **Storage** : separate kill_switch_state table vs nesting under ftf_config.base_env.kill_switch.* — is the dedicated table justified?
2. **Polling interval** : 500 ms default. Too aggressive (PG load) or too slow (1 s p99 halt latency feels too long for a kill switch)?
3. **Fail-safe threshold** : 3 consecutive failures (1.5 s). Should it be tighter (e.g., 1 failure = engage immediately)?
4. **Env-var asymmetry** : engaged works but disengaged doesn’t. Are there scenarios where this hurts (e.g., a botched PG state from a previous engage that you cannot un-stick because the Console is down)?
5. **Channel completeness** : 6 channels listed. Missing a channel (e.g., a phone-app webhook, an exchange-side webhook)?
6. **Auto-engage triggers** : drawdown + circuit breaker + connectivity loss are listed. Should “P1 alert from Grafana on expectancy_net < 0 over 24h” be an auto channel, or stay manual to avoid over-aggressive engagement?
7. **Disengage authorization** : currently any operator can disengage if they have Console / CLI access. Should there be a 2-person rule for disengagement (analogous to nuclear-launch keys)?
8. **In-flight orders policy** : the design explicitly does NOT cancel in-flight orders. Is “operator handles via exchange UI” acceptable, or should there be a follow-up Story for flatten_all?

11. Acceptance criteria (Story level)

- ☐ Design doc merged at `documentation/design/CVN-N001-EF-S02-kill-switch-design.md`
- ☐ ADR-71 file merged codifying I1-I8
- ☐ ADR index + CLAUDE.md ADR cluster updated
- ☐ Committee verdict $\geq$ ACCEPTED or ACCEPTED-WITH-CHANGES (≥ 8.0 avg, no blockers)
- ☐ Open questions §10 answered by committee or recorded as deferred-to-implementation
- ☐ Story OP wp#56 closed with PR + commit reference
- ☐ Sprint F1B-S0-Prereqs closeable (S01 + S02 both done)
- ☐ **Live deployment gate** : engine MUST NOT trade live until companion `flatten_all` Story (EG-S06) ships per committee session `4c388b4c` BLOCKER. Paper trading not gated.

12. References

- Need : CVN-N001 (F1 mission)
- Epic : CVN-N001-EF (#729) — F1 mission operational prereqs
- Sibling Story : CVN-N001-EF-S01 (#709) — MLOps readiness template (merged in PR #730)
- ADRs touched / built upon :
 - ADR-25 — No silent fallback (the basis of I3 fail-safe)
 - ADR-32, ADR-33 — `log_event` format + closed event catalogue (I5 audit)
 - ADR-39 — Runtime standalone, API façade (kernel boundary)
 - ADR-40 — Paper and live share the same kernel (one switch covers both)
 - ADR-56 — Pipeline change must be FTF-testable (kill-switch is NOT FTF-testable, intentionally — it's a safety layer outside the A/B surface)
 - ADR-59 — All pipeline params in PostgreSQL (the data plane this design builds on, but with a dedicated table)
 - ADR-62 — Unified observability via OpenTelemetry (I5 + I6 spans / metrics)
 - ADR-68 — Committee = default review channel (this dossier)
 - ADR-69 — OpenProject orchestrator (Story discipline)
 - ADR-70 — MLOps readiness template (future EG Stories must fill it)
- Code surfaces referenced :
 - `src/paper_trading/cvntrade_paper_trading_engine.py` — kernel
 - `src/paper_trading/safety/cvntrade_risk_manager.py` — existing in-memory primitive
 - `scripts/ftf_config_ui.py` — Console UI host for the new button
 - `infra/migrations/` — schema migrations land here
- Issues : #708 (this Story), #729 (Epic), #608 (Need), session `8db2529d` (committee finding source)