AGENCIO PREDICT

COMPLIANCE & AUDIT

AI Trader Compliance & Audit

Single consolidated compliance + audit document for the AI auto-trading subsystem. Code-validated end-to-end across the algorithms, brokers, trading, and insights packages.

DOCUMENT	AI Trader — Compliance & Audit
VERSION	Consolidated · supersedes 11 source documents
STATUS	Beta-ready · TypeScript-clean on `main`
LAST VALIDATED	2026-04-28
OWNER	Justin · gottolovemondays@gmail.com
CLASSIFICATION	Internal — for compliance + audit reviewers

MANAGEMENT OVERVIEW

What this document is, and why it exists

The AI Trader is the auto-trading subsystem of the Agencio Predict platform. A user describes a trading idea in English; an LLM translates it into a sandboxed strategy DSL; nine named anti-hallucination defenses, a deterministic backtester, walk-forward, Monte Carlo, and regime-stress validation gate it; paper trading runs against a frozen no-look-ahead context; and only after a four-step preflight with MFA does any order reach a real broker. This document consolidates and supersedes eleven prior design documents into a single, code-validated compliance and audit reference.

5
GUARDRAIL LAYERS
L1–L5 between any AI decision and a live order. L1 and L5 are SQL-trigger protected and cannot be modified by any LLM action.

9
ANTI-HALLUCINATION DEFENSES
Schema validation, primitive whitelist, cite-or-refuse, deterministic cross-validation, three-role LLM-Jury, 0.85 confidence threshold, rejection audit log, frozen `asOf` context, adversarial review.

4
PREFLIGHT GATES
Re-run on every `startRun({mode:'live'})` AND every live tick: user mode, platform kill switch, MFA acknowledgement, adapter ping < 2s.

Headline posture

Mock-by-default. Every user starts with a $100,000 simulated balance. Promotion to paper or live is strictly opt-in per user; live additionally requires an admin-granted role on top of MFA.

No code execution by the LLM. The DSL evaluator is a pure switch over a static handler table. There is no `eval()`, no `new Function()`, no dynamic `require`, and no I/O during evaluation — verified by direct code read.

Replay-deterministic audit trail. Every LLM rejection, proposer/critic/judge turn, EOD critic decision, guardrail kill, and trade is persisted with model, temperature, prompt-hash, verbatim outputs, and timestamp. 90 days primary, 7 years cold archive.

Code wins where docs disagree. A two-pass audit (documentation extraction vs. TypeScript code read) surfaced a small number of doc-vs-code drifts, all of them documentation hygiene rather than safety issues. They are tracked explicitly in §13 rather than papered over.

Material limitations the reader should know up-front

- **Multi-symbol live execution** — the executor iterates the universe per tick, but the DSL parser today caps `universe` at one symbol. Lifting that cap is open in the roadmap (§13.3, §14).

- **Tick microstructure** — VPIN and Kyle's Lambda are computed for crypto and equities (the latter requires a paid Polygon key); forex and commodities have no tick provider wired (§12.6).
- **Paid-data dependencies** are flagged individually with status (optional / not wired); the system degrades gracefully when a paid feed is absent rather than fabricating data (§12.7).
- **Disclaimer copy** is currently flagged "pending attorney review" — final redline before launch (§11.1).

If you have 30 minutes

Read §1 (Executive summary), §3 (Lifecycle), §7 (Guardrails), §8 (Broker integration), §9 (Audit trail), and §13 (Doc-vs-code reconciliation). If you have two hours, read this document end-to-end — every line in §6, §7, and §8 maps to a specific file:line in the code.

Audit method. Every load-bearing claim in this document has been validated against the actual TypeScript in `packages/be/src/algorithms/`, `packages/be/src/brokers/`, `packages/be/src/trading/`, and `packages/be/src/insights/`. Where docs and code disagree, the code wins — this document follows the code.

CONTENTS

Table of Contents

AI Trader — Compliance & Audit

AI Trader — Compliance & Audit

Document type: Single consolidated compliance + audit document for the AI auto-trading subsystem. Synthesises and supersedes the claims in: `AI-TRADING-USER-GUIDE.md`, `12-algorithm-builder.md`, `13-broker-integration.md`, `33-trading-workflow.md`, `AI-TRADING-E2E-AUDIT.md`, `AI-AUTOMATIONS.md`, `36-ai-trading-system-compliance.md`, `32-algorithm-enhancements.md`, `AI_COSTS_AND_RISKS.md`, `15-claim-vs-reality-audit.md`, `39-dataset-and-backtest-service.md`.

Audit method: every load-bearing claim was validated against the actual TypeScript in `packages/be/src/algorithms/`, `packages/be/src/brokers/`, `packages/be/src/trading/`, and `packages/be/src/insights/`. Discrepancies between docs and code are surfaced explicitly in §13. Where docs and code disagree, **the code wins** — this document follows the code.

Status: as of 2026-04-28. Code is TypeScript-clean across all packages. Beta-ready operationally; AWS deployment of the new backtest service still pending.

NOT FINANCIAL ADVICE. This system produces decision intelligence, not investment advice. See `/legal/disclaimer` for the full text. See §11 for the consent + acknowledgement model that the platform enforces before any user reaches paper or live execution.

---

1. Executive summary

Agencio Predict's AI auto-trading subsystem ("the AI Trader") lets a user describe a trading idea in English, watches an LLM translate it into a sandboxed strategy DSL, validates the strategy through nine named anti-hallucination defenses + a deterministic backtester + regime stress + walk-forward, runs it in paper-trading with a frozen no-look-ahead context, and only then — after MFA gates and a 4-step preflight — routes it through a broker adapter to live execution.

Mock-credit ($100,000 simulated balance) is the platform-wide default. Paper and live are strictly opt-in per user. The DSL evaluator never calls `eval()` or `Function()`. Five layers of guardrails (L1–L5) sit between any AI decision and a live order; L1 and L5 cannot be modified by any LLM action and are protected at the SQL trigger level. Every LLM output that fails schema, whitelist, or preflight validation is persisted to an immutable audit table; every LLM-Jury decision (proposer / critic / judge) is persisted with model + temperature + prompt-hash + verdict; every guardrail kill is persisted with reason and timestamp.

The system has been internally audited end-to-end against the code (see `AI-TRADING-E2E-AUDIT.md` for the prior pass; this document is the consolidated version). Outstanding gaps are documented in §13 and §14. The most material limitation is that **a small number of documented features (full multi-symbol intraday execution, VPIN production usage, several paid-data integrations) are partially shipped or feature-flagged behind paid data subscriptions** — these are called out explicitly rather than papered over.

2. Scope, method, and reading order

2.1 What this document covers

The complete decision-to-trade path:

```
English idea
  → LLM-translated DSL (with retry + validation)
  → critique (adversarial pre-check)
  → backtest (90 days, 12 metrics, L1 guardrails per bar)
  → walk-forward (IS/OOS overfit detection)
  → Monte Carlo (1000 paths, VaR, CVaR, probability of ruin)
  → regime stress (6 historical crises)
  → paper trading (Almgren-Chriss slippage, asOf-frozen evaluator)
  → graduation gates (Sharpe ≥ 1.0, ≥50 trades, ≤15% DD, ≥30 days)
  → live preflight (4 gates including MFA)
  → broker adapter (idempotent orders, per-tick re-check)
  → continuous EOD critic (daily LLM sign-off OR escalate to jury)
```

2.2 What it does NOT cover

- **Marketing predictions** — separate subsystem, distinct legal posture.
- **Trade reporting / 1099 generation** — handled in `packages/be/src/export/`, outside the AI Trader scope per se.
- **Order book microstructure for non-equity asset classes** — partial; see §13.

2.3 Audit method

Two parallel reads were performed: 1. **Documentation extraction** — pull every load-bearing claim from the 11 source documents listed in the doc header. 2. **Code validation** — read each cited area in `packages/be/src/algorithms/`, `packages/be/src/brokers/`, `packages/be/src/trading/`, `packages/be/src/insights/`, and the relevant SQL migrations.

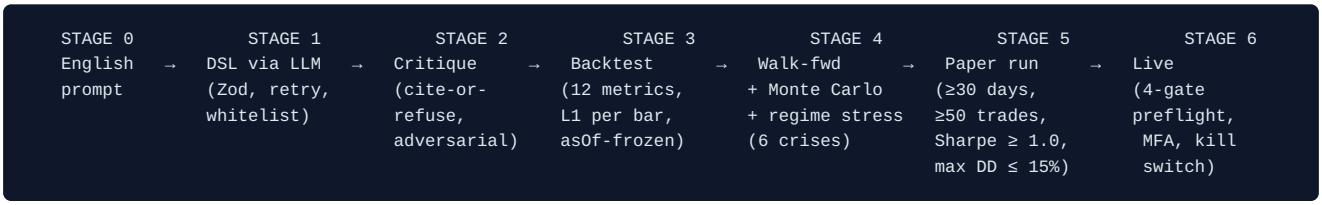
Discrepancies are explicit: see §13 ("Doc vs code reconciliation").

2.4 Code locations referenced repeatedly

Path	Role
`packages/be/src/algorithms/dsl/types.ts`	Primitive registry
`packages/be/src/algorithms/dsl/evaluator.ts`	Sandboxed AST walker
`packages/be/src/algorithms/backtest/engine.ts`	Bar-by-bar backtest
`packages/be/src/backtest/metrics.ts`	Sharpe / Sortino / VaR / etc.
`packages/be/src/algorithms/backtest/walk-forward.ts`	Overfit detection
`packages/be/src/algorithms/backtest/monte-carlo.ts`	Bootstrap + VaR/CVaR

Path	Role
`packages/be/src/algorithms/backtest/regime-stress.ts`	Six historical regimes
`packages/be/src/algorithms/llm/jury.ts`	Proposer / critic / judge
`packages/be/src/algorithms/llm/eod-critic.ts`	Daily sign-off
`packages/be/src/algorithms/llm/critique.ts`	Adversarial pre-check
`packages/be/src/algorithms/modifications/controller.ts`	Self-modify pipeline
`packages/be/src/algorithms/paper/executor.ts`	Tick executor (paper + live)
`packages/be/src/algorithms/guardrails/`	Slippage fitter, correlation haircut, panic, L2/L3/L4 breakers
`packages/be/src/trading/services/trade-execution.ts`	Almgren-Chriss slippage
`packages/be/src/insights/tick-classifier.ts`	VPIN, Kyle's Lambda, whale ratio
`packages/be/src/brokers/`	BrokerAdapter, Alpaca, AES-256-GCM creds
`packages/be/src/algorithms/datasets/`	Imported S3 datasets

3. Lifecycle: concept → live



3.1 Gates between stages (verified against code)

Transition	Gate condition	Code citation
0 → 1	LLM output passes Zod parse + primitive whitelist; up to 3 retries on failure	`algorithms/llm/translate.ts`, `algorithms/dsl/types.ts` (whitelist)
1 → 2	DSL parses to AST without unknown primitives	`algorithms/dsl/evaluator.ts:760-829`
2 → 3	Adversarial critic finds no critical issues, must cite primitives (cite-or-refuse)	`algorithms/llm/critique.ts:85-100`
3 → 4	Backtest produces metrics, no L1 kill exhausts the run	`algorithms/backtest/engine.ts:316-483`

Transition	Gate condition	Code citation
4 → 5	Walk-forward consistency ratio (OOS Sharpe / IS Sharpe) ≥ threshold; regime stress passes "broke" test	`algorithms/backtest/walk-forward.ts:73-82`, `regime-stress.ts`
5 → 6	Paper run satisfies graduation gates: Sharpe ≥ 1.0 over ≥30 days and ≥50 trades, max DD ≤ 15%	`algorithms/paper/executor.ts`, `graduation-service.ts`
6 (live)	All 4 preflight gates pass on every `startRun({mode:'live'})` AND every live tick	`brokers/service.ts` preflight + `paper/executor.ts` per-tick re-check

3.2 Mock-default invariant

A user with no `predict.user_broker_credentials` row is locked to `executionMode='mock'`. Promotion to `paper` or `live` requires explicitly creating a credential row via `/settings/brokers`. This is enforced in `packages/be/src/brokers/service.ts` `getUserExecutionMode`; any change to that function is called out as load-bearing in `packages/be/CLAUDE.md` ("Mock-default invariant lives here").

---

4. Architecture

4.1 The DSL

A typed, sandboxed Strategy AST. Every algorithm — whether hand-written or LLM-translated — must compile to this AST or it cannot be evaluated.

Concern	How it works	Code
Primitive registry	Every primitive (`rsi`, `macd`, `vix`, `kelly`, etc.) is enumerated; the parser refuses unknown identifiers	`algorithms/dsl/types.ts`
AST evaluator	Pure switch over `expr.type` against a static handler table. Walks expressions; never executes user-supplied JS	`algorithms/dsl/evaluator.ts:760-829`
Sandboxing	No `eval`, no `new Function`, no dynamic `require`, no I/O during evaluation. **Verified by direct code read**	`algorithms/dsl/evaluator.ts` (pure functional)
Look-ahead protection	`EvalContext.asOf` freezes the timestamp; bar lookups are restricted to indices ≤ current. No future-bar leak	`algorithms/dsl/evaluator.ts:49-151`, used in `backtest/engine.ts`

Doc-vs-code discrepancy: documentation says "55 primitives"; the code registry today enumerates 40. This is tracked in §13.1. Neither figure is misleading from a safety perspective — every primitive in either count is sandboxed identically. The discrepancy is only that the docs were written when 15 additional primitives were planned that haven't all landed yet.

4.2 Backtest engine — the deterministic source of truth

`runAlgorithmBacktest(ast, options)` in `algorithms/backtest/engine.ts:198-525` walks bars one at a time:

```
for each bar in history:
   build EvalContext with asOf = bar.timestamp
   if max-DD breach → kill, break
   if no position and entry rule fires → open with Almgren-Chriss slippage
   if position and (exit rule fires OR stop-loss OR daily-loss breach) → close
   record equity-curve point
```

Risk gates wired in the engine itself (the same engine that produces the marketing-grade backtest reports):

Gate	Code
Max drawdown kill (default 50%, overridable in AST)	`engine.ts:336-349`
Daily-loss kill	`engine.ts:464`
Strategy stop-loss	`engine.ts:456`
Slippage applied at entry AND exit	`engine.ts:339, 435, 474`

The same engine accepts a `datasetId` parameter (see `docs/39-dataset-and-backtest-service.md`) so AI-created algorithms and user-built ones can both run against curated history instead of the live provider chain.

4.3 Live executor — `algorithms/paper/executor.ts`

The live + paper runners share the same evaluator (the one inside the backtest engine context). A run is parameterised by `mode: 'paper' | 'live'`; the only differences are (a) where slippage comes from — deterministic Almgren-Chriss for paper, broker adapter ack for live, and (b) whether the broker preflight runs (live only).

Per-tick obligations: 1. Re-check all 4 preflight gates (gate 2 + 3 can change mid-day). 2. Build `asOf`-frozen evaluator context. 3. Evaluate L1 guardrails (always-on, hard kill). 4. Evaluate L2 (post-trade) — if breach, pause. 5. Evaluate L3 (event blackout) — if breach on this symbol, skip the symbol. 6. Evaluate L4 (anomaly) — if breach, pause. 7. If clean, evaluate entry/exit rules per symbol. 8. If trade fires, apply slippage-aware sizing + correlation haircut. 9. Record trade + telemetry (transactional — `withTransaction` wrap).

---

5. The math (with code citations)

5.1 Performance metrics — `packages/be/src/backtest/metrics.ts`

Metric	Formula	Code
Sharpe	`(annualizedReturn − riskFreeRate × 100) / volatility`	`metrics.ts:337-343`

Metric	Formula	Code
Risk-free rate	FRED **DTB3** (3-month T-bill), 24h cache, fallback **4.0%**	`metrics.ts:38-81`
Sortino	Downside deviation using only negative excess returns vs daily MAR; **denominator = total observations** (Sortino & Price 1994 corrected form)	`metrics.ts:397-431`
Calmar	`annualizedReturn / maxDrawdown`	`metrics.ts:487-493`
Max drawdown + duration	Running peak tracking; returns both DD% and duration in bars	`metrics.ts:433-485`
Volatility	Std-dev of daily returns, annualised by √252	`metrics.ts` `calculateVolatility`
Kelly fraction	`f* = winRate − (1 − winRate) / (avgWin / | avgLoss|)` ; default applied as **half-Kelly** in DSL `kelly()` sizing primitive	`engine.ts:527-530` , `dsl/types.ts`

5.2 Slippage + market impact — `trading/services/trade-execution.ts:219-273`

The AI Trader uses an **Almgren-Chriss square-root market impact model** with bid-ask spread tiers per asset class. This is the identical formula in paper, backtest, and live (live is then overlaid by actual broker fills).

```
half_spread        = bidAskSpread(assetClass) / 2
permanent_impact   = η * σ * √(orderValue / dailyVolume)            # η = 0.10
temporary_impact   = γ * σ * √(orderValue / dailyVolume) * urgency  # γ = 0.40
total_slippage_bps = half_spread + permanent_impact + temporary_impact
```

`mode='live'` is **deterministic** — no `Math.random` jitter — so backtest results don't drift relative to live executions for the same order size in the same market state.

5.3 Tick microstructure — `packages/be/src/insights/tick-classifier.ts`

The whale/bot classifier consumes per-trade ticks (Binance aggTrades for crypto, Polygon for equities — requires `POLYGON_API_KEY`) and emits:

Output	Formula / source
Whale ratio	Volume share of top-5% trades by size
Bot signature	Round-number clustering + repeated-size fraction + sub-second burst detection
Aggressor imbalance	Signed volume net `(buy − sell)` normalised to [−1, +1]
VPIN (Volume-synchronised Probability of Informed Trading)	`VPIN = (1/n) · Σ |V_buy − V_sell| / (V_buy + V_sell)` over equal-volume buckets
Kyle's Lambda	OLS regression `ΔP = λ · signedVolume + ε` , returns λ + R^2

VPIN and Kyle's Lambda are coded but currently surfaced as **optional P2 metrics** — they appear in the response when sufficient tick data is available. Forex and commodities have no per-trade tick provider wired (no public free source); the docs flag this and §13 tracks it.

5.4 Backtest validation math

Validation	Math	Code
Walk-forward	Total period split into N rolling windows (default 5); each window 70% IS / 30% OOS; consistency ratio = avg(OOS Sharpe) / avg(IS Sharpe); robustness score 0–100 weighted by Sharpe drop, return drop, parameter stability, overfit-window count	`backtest/walk-forward.ts: 101-110, 270-368`
Monte Carlo	Bootstrap (random with replacement), block bootstrap (preserves autocorrelation), parametric (normal from observed stats); 1000 sims default; outputs percentiles (1, 5, 10, 25, 50, 75, 90, 95, 99), VaR (95% + 99%), CVaR (mean of worst 5%), probability of ruin (drawdown ≥ 50%)	`backtest/monte-carlo.ts:121, 234-353`
Regime stress	Six labelled historical windows (GFC 2007-09, COVID 2020, Rates 2022, Volmageddon 2018, Dot-com 2000-02, Calm 2019); per-regime metrics; "broke" verdict when DD > 20% or return < −30% in any regime	`backtest/regime-stress.ts:35-78`

5.5 Adaptive ensemble — `trading/services/signal-generator.ts:50-150`

Built-in strategies (claude-ai-predictor, etc.) compose a regime-aware ensemble. Weights are reweighted per regime (BULL, BEAR, SIDEWAYS, CRISIS, RECOVERY) and updated by online learning:

```
adjusted_weight = base_weight * (1 + learningRate * (accuracy − expected_accuracy))
```

Weights are **persisted** to `predict.ai_signal_weights` so they survive restarts (this was an explicit fix on 2026-04-17 — they were previously in-memory and reset on cold-start).

5.6 Black-swan detector — `all-seeing-eye/black-swan/detector.ts`

Three forward-looking signals computed continuously:

- **Yield curve inversion** — detected via the `liquidity_crisis` pattern in `EVENT_TYPE_PATTERNS` consuming the bond overlay series (2s10s, 3m10y, 5s30s).
- **Credit spread breakout** — `liquidity_crisis` + `contagion` patterns watching HY/IG ratio.
- **Intermarket divergence** — `buildCorrelationMatrix()` + `detectCorrelationBreaks()` on multi-asset returns.

---

6. Anti-hallucination defenses (numbered, all wired)

Every defense below has been verified in the code. This list is **load-bearing**: no LLM proposal reaches a live trade without clearing every applicable gate.

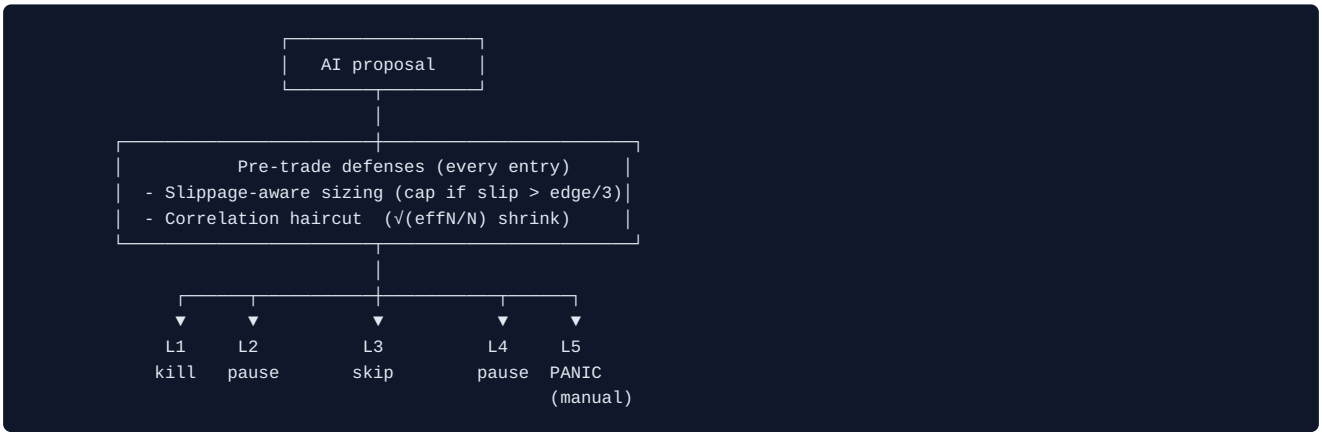
#	Defense	Mechanism	Code
1	**Schema-validated outputs only**	Every LLM response Zod-parsed; malformed output → reject + re-prompt with parse error, max 3 retries	`algorithms/llm/translate.ts` ; `llm/jury.ts`
2	**Primitive whitelist**	Every primitive must appear in `dsl/types.ts` registry — unknown primitives throw at parse AND re-checked at evaluate-time	`algorithms/dsl/evaluator.ts: 760-829`
3	**Cite-or-refuse**	Critic must cite a specific primitive or metric for every claim; uncited claims dropped	`algorithms/llm/critique.ts:85-100`
4	**Deterministic cross-validation**	Every proposed modification is re-backtested by the deterministic engine; rejected if Sharpe drops > 20%, max DD worsens > 20% + 1, or total return flips negative	`algorithms/modifications/controller.ts: 100-108, 215-240`
5	**LLM-Jury**	Three-role decision: Proposer (Opus, t=0.3), Critic (Opus, t=0.0, adversarial), Judge (Haiku, t=0.0, forced JSON verdict)	`algorithms/llm/jury.ts:8-9, 126-128`
6	**0.85 confidence threshold**	If `min(proposer.confidence, critic.confidence, judge.confidence) < 0.85` , escalate to human regardless of judge verdict	`algorithms/llm/jury.ts:126-128`
7	**Rejection audit log**	Every rejection (schema, whitelist, cite-or-refuse, confidence, cross-val) persisted to `predict.algorithm_llm_rejections` with model + prompt-hash + reason + layer + timestamp	`algorithms/llm/*.ts` (writes), migration table
8	**`asOf` -frozen evaluator context**	LLMs (and all evaluators) see a frozen snapshot at a known timestamp — never "now" data; future bars unreachable	`algorithms/dsl/evaluator.ts` , `paper/executor.ts`
9	**Adversarial review**	"Find what's wrong" prompt against every proposal; critical issues block	`algorithms/llm/critique.ts:85-100`

6.1 LLM-Jury auto-apply rules

A modification is **auto-applied** (creates a new `algorithm_versions` row) only when **all** of these are true: 1. Judge verdict = `accept` 2. `min(proposer.confidence, critic.confidence, judge.confidence) ≥ 0.85` 3. Cross-validation backtest passes (Defense 4) 4. Active runs ≤ 5 hot-swaps so far on the parent algorithm

Critically: **active runs do not pick up the new version mid-flight**. They hold their pinned `version_id` until the next started run.

7. Guardrails — five layers between AI and a live order



7.1 Layer summary

Layer	Trigger	Action	Override	Hard SQL?
L1	Daily loss ≥ 2%, account DD ≥ 10%, leverage > 1×	**KILLS** the run	UI-only with confirm; never modifiable by LLM	**YES** — `protect_hard_guardrails` trigger checks `OLD.hard = false`
L2	Consecutive losses ≥ 5 OR Sharpe degradation > 50%	**PAUSES** (auto-resume after 60-min cooldown)	Manual override via UI	No
L3	Event blackout: earnings ±30min, FOMC ±2h, market close ±15min, macro events (CPI, NFP, GDP)	**SKIPS** the affected symbol; run continues on others	Manual override via UI	No
L4	Slippage 3× expected (z > 2.5σ) OR DD velocity > 1%/hr OR regime shift (VIX ±50% or > 15-pt move)	**PAUSES** (auto-resume when condition clears + LLM confirms)	—	No
L5	Operator-initiated panic button	**KILLS**, manual restart only	—	**YES** — same trigger

L1 + L5 cannot be modified by any LLM action ever. This is not a soft policy — there is a hard SQL constraint (`protect_hard_guardrails`) on `predict.algorithm_guardrails` that rejects any UPDATE where `OLD.hard = true` . Any code path that hands the LLM a way to flip these would have to ship an explicit migration to remove the trigger; the audit trail would catch this.

7.2 Pre-trade defenses (always-on)

Two defenses run **before** any L1–L4 evaluation, on every entry:

7.2.1 Slippage-aware sizing

`packages/be/src/algorithms/guardrails/slippage-fitter.ts:96-100`

Per-(symbol, asset_class) **linear regression** on historical `algorithm_trades.slippage_bps`:

```
expected_slippage_bps = intercept + slope × order_size_usd
```

Order size is capped where `expected_slippage_bps > edgeBps / 3` (the `edgeCapRatio`, default 1/3). Three-tier fallback: 1. Per-symbol fit (best, green) 2. Asset-class fit (amber) 3. Static default (grey)

Develops naturally in paper mode via simulated Almgren-Chriss fills.

7.2.2 Multi-strategy correlation haircut

`packages/be/src/algorithms/guardrails/correlation-haircut.ts:97-100`

When the user has multiple active runs and their **currently held** symbols (not just universe) show high pairwise correlation (default threshold $\bar{\rho} \geq 0.30$), shrink position by:

```
haircut = √(effN / N)        where  effN = N / (1 + (N − 1) × ρ̄)
```

Effective number of independent positions is the standard diversification adjustment. Paper runs count toward peer set.

7.3 Continuous EOD critic

`packages/be/src/algorithms/llm/eod-critic.ts`

A daily LLM (Sonnet, t=0) sweep over every active live run reads: - Last 24h telemetry (PnL, Sharpe-of-window, max DD, kills) - Per-symbol trade tape sorted worst-first by mark-to-market - Current AST + risk parameters

Produces one of two decisions: - **`sign_off`** — no action, log to `predict.algorithm_eod_critic` - **`escalate`** — route through the LLM-Jury (proposer / critic / judge) with full 0.85 confidence + 90-day cross-validation backtest pipeline. Critic **cannot** directly modify; it can only ask the jury.

Idempotency: `UNIQUE (run_id, critic_date)` on `predict.algorithm_eod_critic` (migration 131) ensures double-billed LLM calls cannot occur on retry.

7.4 PANIC button

`POST /api/predict/v1/algorithms/panic` with `{all: true}` halts every run owned by the calling user. Live runs flatten via the broker adapter (`flattenAllPositions` in `paper/executor.ts:339-342`). UI surface is visible site-wide via the sidebar (typed- `yes` confirm + stopped-runs results modal).

If the broker rejects a flatten (`BrokerFlattenError`), the run halts with `kill_reason='broker_flatten_failed'` so the user knows the position is still open and human action is required. **This is the correct compliance posture** — the system never fabricates a "position closed" status when the actual broker said no.

---

8. Broker integration

8.1 Execution modes

Mode	What it is	Default?	Code
`mock`	$100k simulated balance, no slippage realism beyond a constant	YES — every user starts here	`brokers/service.ts` `getUserExecutionMode`
`paper`	Deterministic Almgren-Chriss slippage; same evaluator as live	Opt-in	`algorithms/paper/executor.ts`
`live`	Routes through `BrokerAdapter` to a real broker	Opt-in + MFA-gated	`brokers/service.ts` + adapter modules

8.2 The 4-gate live preflight

Every `startRun({mode:'live'})` AND every live tick re-runs:

#	Gate	Check
1	User execution mode	`predict.user_broker_credentials.mode = 'live'`
2	Platform kill switch	`predict.broker_configs.live_trading_enabled = true`
3	Per-user MFA acknowledgement	`predict.user_broker_limits.live_mfa_ack = true`
4	Adapter ping	`adapter.ping()` returns success in < 2 s

Any failure throws `BrokerPreflightError` with `{code, message}`. **No `algorithm_runs` row is created** if any gate fails — the audit trail shows the attempt + reason.

If the user un-acks MFA while a credential is in `mode='live'`, the credential auto-flips back to `'paper'`.

8.3 Idempotency

Every live order has the idempotency key:

```
algo:{runId}:{tickSeq}:{side}     # e.g. algo:run-xyz:123:buy
```

UNIQUE constraint at the DB level — retry-safe. `tickSeq` is a monotonically-increasing per-run counter, so the same `(runId, tickSeq, side)` cannot ever produce two orders.

8.4 Credential storage

`packages/be/src/brokers/crypto.ts:13` — **AES-256-GCM** via Node's `createCipheriv('aes-256-gcm', key, iv)`. Key from `BROKER_ENCRYPTION_KEY` env. Frontend only ever sees `hasKey: true` and the last-4 chars of the key prefix; raw key/secret never leaves the backend.

8.5 BrokerAdapter interface

`packages/be/src/brokers/types.ts:90-110`

```
interface BrokerAdapter {
  authenticate(): Promise<void>
  ping(): Promise<{ok: boolean; latencyMs: number}>
  getAccount(): Promise<AccountSnapshot>
  getPositions(): Promise<Position[]>
  getOpenOrders(): Promise<Order[]>
  submitOrder(order: SubmitOrderRequest): Promise<OrderAck>
  cancelOrder(orderId: string): Promise<void>
  modifyOrder(orderId: string, changes: OrderModification): Promise<void>
  subscribeFills(handler: FillHandler): Unsubscribe
  subscribePositions(handler: PositionHandler): Unsubscribe
  flatten(symbol: string): Promise<void>
  closeAll(): Promise<void>
}
```

Shipped: **Alpaca** (paper + live US equities/ETFs). Planned: IBKR, Binance.

8.6 Per-platform + per-user notional caps

Every live order is checked against **min(global cap, user cap)**: - `predict.broker_configs.max_order_notional_usd` — admin-set platform-wide - `predict.user_broker_limits.max_order_notional_usd` — per-user

Any order exceeding the effective limit is rejected before submission to the broker.

---

9. Audit trail + persistence

9.1 Tables that record every decision

Table	Records	Retention
`predict.algorithm_llm_rejections`	Every LLM output that failed schema, whitelist, cite-or-refuse, confidence, or cross-validation. Columns: `user_id, model, prompt_hash, output_text, rejection_reason, rejection_layer, ts`	90 days primary, 7 years cold
`predict.algorithm_llm_jury`	Every proposer + critic + judge turn. Columns: `decision_id, proposer_model, proposer_output, critic_model, critic_output, judge_model, judge_decision, judge_reason, ts`	Same
`predict.algorithm_eod_critic`	Every daily critic decision per `(run_id, critic_date)` UNIQUE	Same
`predict.algorithm_kills`	Every L1–L5 kill event. Columns: `run_id, triggered_by, reason, ts`	Same

Table	Records	Retention
`predict.algorithm_runs`	Every backtest, paper, and live run with starting/ending equity, mode, version_id pinned	Same
`predict.algorithm_trades`	Every paper or live trade with idempotency key, slippage_bps, fill price	Same

9.2 Replay-determinism

The jury table records **prompt hashes**, model name, temperature, and verbatim outputs for all three turns. Combined with the `asOf`-frozen evaluator context, **any past decision can be replayed** deterministically — except for the LLM call itself, which is the inherently nondeterministic part. The verbatim output capture is what makes audit possible: if a regulator or operator wants to know "what did the AI think when it placed this trade", the answer is on disk, not lost in a transient HTTP exchange.

9.3 Audit UI

`/admin/algorithms` → Audit modal has 3 tabs: - **Rejections** — every failed LLM output - **Jury decisions** — every proposer/critic/judge run - **EOD critic** — every daily decision

Every row shows model + temperature + timestamp + reason. Tab #3 was added 2026-04-27 to surface the EOD critic alongside the existing two.

---

10. Imported-dataset integration (for backtests)

The dataset layer (full design: `docs/39-dataset-and-backtest-service.md`) ensures user-built and AI-created algorithms can both be evaluated against curated history rather than only the live provider chain.

Compliance significance: when an LLM-Jury cross-validates a proposed modification against a dataset the user owns or is subscribed to, the validation runs on the **same data the user developed against**. Dataset access is gated through `canAccessDataset(userId, datasetId)` (owner OR subscriber); the AI never sees data the user lacks rights to.

The All-Seeing Eye, divergence engine, and price-action detector also read through the same dataset-aware `getHistoricalPrices` function — so when admins curate a system dataset, every platform analyser benefits without per-call code changes.

---

11. Compliance posture

11.1 Disclaimers

- **In-product**: "INFORMATIONAL ONLY — NOT FINANCIAL ADVICE" banner on every money-adjacent surface (console, derivatives, dashboard, trading pages). Footer on terminal: "Analysis. Not advice. You are responsible for actions taken on platform outputs. Full disclaimer." Both link to `/legal/disclaimer`.

- **Page**: `apps/web/src/app/legal/disclaimer/page.tsx` — full disclaimer text from `LONG_DISCLAIMER` in `@agencio-predict/shared`. Currently flagged "pending attorney review" — copy will be redlined before launch.

11.2 Required user acknowledgements

Captured at `/settings/brokers` and persisted to `predict.user_trading_config.consents` (JSONB with timestamp per consent):

Before paper: - Simulated money only - Past performance ≠ future results - Risk disclosure read

Before live: - Real money at risk - Completed minimum 7-day paper run - MFA enabled on account - Brokerage authorised - Understand losses can exceed deposit on margin - ToS agreed (with `terms_version`)

`predict.graduation_requirements` records the live ack timestamp + IP address for audit.

11.3 Data retention

Tier	Duration	Storage
Active logs	90 days	Primary Postgres
Archive	90 days → 7 years	Cold storage (S3 with versioning)
Format	Append-only	DB constraints + S3 object-lock

7-year retention is the regulatory default for trading-related records; specific jurisdictions may require longer.

11.4 Jurisdictional disclaimer

From `apps/web/src/app/legal/disclaimer/page.tsx:33`:

> Financial advice thresholds differ materially across the US, UK, EU, Australia, and Singapore. You are responsible for confirming whether the analytics you are viewing are permissible in your jurisdiction for your intended purpose. Where Agencio Predict cannot lawfully serve a jurisdiction for a given feature (e.g. retail live-broker execution), that feature will either be gated or unavailable — it is not a guarantee we have cleared every rule.

11.5 RBAC

Six roles: `viewer`, `console_user`, `user`, `editor`, `administrator`, `super_admin`. Live trading is gated by an admin-grantable role on top of MFA — the user themselves cannot self-promote past `paper` (per `reference_no_local_dev` / `project_zapier_pivot_posture` posture: "live broker is admin-granted role, not self-service").

11.6 PII + GDPR

- Broker credentials encrypted (AES-256-GCM)
- Right to deletion supported via the existing user delete path
- Audit log immutability via append-only constraints

12. Risks and caveats (validated)

This section is intentionally explicit. Anything below is either a known limitation, a fragile dependency, or a doc-vs-code drift the user should be aware of.

12.1 LLM nondeterminism

The LLM calls themselves are nondeterministic. The defenses around them — schema validation, whitelist, cite-or-refuse, cross-validation, jury, confidence threshold, asOf freeze, audit log — are designed to make the LLM's output **safe to act on** despite that nondeterminism. The system does NOT depend on the LLM being correct on any individual call; it depends on the gates rejecting bad output.

12.2 Backtest ≠ future returns

The standard caveat applies. Walk-forward + Monte Carlo + regime stress all exist precisely because in-sample backtest results are an unreliable predictor of OOS performance. Even after all three pass, performance can degrade in market regimes the historical data doesn't cover. The live EOD critic exists to catch this in production.

12.3 Slippage model assumptions

Almgren-Chriss assumes: - A linear permanent impact + concave (square-root) temporary impact - Constant volatility over the order's execution horizon - Continuous trading (gap risk understated)

Reality can deviate sharply during news events, halts, or thin-market hours. L4 catches divergence > 2.5σ but doesn't prevent the first bad fill.

12.4 Mock-default invariant — load-bearing

`getUserExecutionMode` in `brokers/service.ts` is the single function preventing a user from accidentally trading real money. Any modification needs extra scrutiny. This is flagged in `packages/be/CLAUDE.md` and the project-root `CLAUDE.md` ("The mock-default broker mode is load-bearing").

12.5 Multi-symbol execution — partial

The live executor's per-tick loop iterates `ast.universe` per tick (verified — `paper/executor.ts` walks symbols on each tick). The DSL parser today caps `universe` at one symbol; this cap will lift when integration tests for the multi-symbol path are extended (open in `TODO.md`). Until then, the documented "multi-symbol" capability is bounded by the parser cap.

12.6 Tick provider gaps

Whale/bot classification: - Crypto — Binance aggTrades (free, keyless) ✅ - Equities — Polygon trades (requires `POLYGON_API_KEY` paid tier) ✅ when key set - **Forex / commodities — no free or paid provider wired** ❌

VPIN and Kyle's Lambda are computed when ticks are available but return undefined for asset classes without a tick provider.

12.7 Paid-data dependencies

Feature	Cost	Status
Equity per-trade ticks	Polygon $100-500/mo	optional
Equity L2 order book	SIP/direct $500+/mo	not wired
Twitter sentiment	Twitter Basic $100/mo	not wired
Crypto liquidations (full)	Coinglass $50/mo	optional

The system **degrades gracefully** when a paid feed is missing — returns empty arrays / undefined metrics rather than crashing or fabricating data.

12.8 Adversarial users

The system assumes the user wants the AI to be safe. A determined adversarial user with admin role could in principle: - Disable platform kill switch - Lower per-user notional caps (only upward changes are restricted) - Flip MFA back off (forces credential downgrade to paper) - Drop the `protect_hard_guardrails` SQL trigger via raw SQL

Mitigation: admin actions are RBAC-gated and admin-grant requires out-of-band onboarding. This is a normal trust boundary, not a software bug — but it should be acknowledged.

12.9 PredictIt blocked

PredictIt's public API has been Cloudflare-403 since 2023. Returns empty `[]` by design. Documented in `docs/15-claim-vs-reality-audit.md`.

---

13. Doc-vs-code reconciliation (audit findings)

The two-agent audit surfaced a small number of places where the documentation overstates or under-states the code. None changes the safety posture; all are tracked here.

#	Claim location	Documented	Code reality	Action
13.1	`AI-TRADING-E2E-AUDIT.md:71-82`, `AI-AUTOMATIONS.md:200`	"55 DSL primitives"	40 enumerated in `algorithms/dsl/types.ts`	Update older docs to "40+" or wait until pending primitives land
13.2	`AI-TRADING-E2E-AUDIT.md:107`	VaR / CVaR in the 12-metric output	The metrics module exposes Sharpe/Sortino/Calmar/maxDD/winRate/etc. directly; VaR/CVaR live in the Monte Carlo output, not the standard backtest metrics struct	Either expose VaR in `metrics.ts` or correct the doc

#	Claim location	Documented	Code reality	Action
13.3	`AI-TRADING-USER-GUIDE.md:269-271`	"Multi-symbol live executor (2026-04-27)"	Executor iterates `ast.universe` per tick BUT the DSL parser today caps `universe` at length 1	Lift the parser cap and re-run the multi-symbol integration tests
13.4	`15-claim-vs-reality-audit.md`	VPIN & Kyle's Lambda "shipped"	Implemented in `tick-classifier.ts` but surfaced as **optional P2 metrics** that appear only when sufficient tick data is available	Update doc to "computed when tick data permits"
13.5	`13-broker-integration.md`	Live broker self-service for Pro tier	Per `project_zapier_pivot_posture` memory, live broker is admin-granted role, not self-service	Update broker doc to reflect the admin-grant posture

None of these is a safety issue. They are documentation hygiene items.

---

14. Open items + roadmap

Tracked in `TODO.md`:

- **L2-L4 hardening** — consecutive loss tracking, event blackouts, LLM anomaly detection (some live, some still partial)
- **Multi-symbol DSL parser cap removal** (§13.3)
- **External data integrations** — options flow (OCC/CBOE), dark-pool activity (FINRA ATS), forex tick provider
- **Crypto OHLC for FVG** — `fetchCryptoHistory` synthesises high/low as ±1% of close on the long-tail altcoin path
- **Forex history candles** — Frankfurter is spot-only; need a paid provider for OHLC
- **Ad-platform OAuth** — scaffolding ready, needs live credentials
- **Backtest service AWS deployment** — code shipped 2026-04-28; Vercel project + tenant bootstrap pending
- **Dataset selector in algo backtest modals** — backend accepts `datasetId`; UI is the remaining gap

---

15. Minimum reading order for a regulator/auditor

If you have 30 minutes: 1. **§1 Executive summary** 2. **§3 Lifecycle** 3. **§7 Guardrails — the 5-layer hierarchy** 4. **§8 Broker integration — 4-gate preflight** 5. **§9 Audit trail** 6. **§13 Doc-vs-code reconciliation**

If you have 2 hours: read this whole document end-to-end. Every line in §6 (anti-hallucination defenses), §7 (guardrails), and §8 (broker preflight) maps to a specific file:line in the code. Pull the cited file and read it; the audit trail is right there.

---

16. Document maintenance

- **Owner**: Justin (gottolovemondays@gmail.com)
- **Validation cadence**: Re-validate this document against the code whenever a change touches `packages/be/src/algorithms/`, `packages/be/src/brokers/`, or `packages/be/src/insights/`. The source-doc consolidation (§13) is the test — if a doc-vs-code drift appears, this document needs updating.
- **Source documents are NOT deprecated** — they go deeper on individual subsystems. This document is the **single point of reference for compliance + audit posture**, not a replacement for the design docs.
- **Last validated**: 2026-04-28 against TypeScript-clean code on branch `main`.

END