# Second Codex Prompt (Tailored to your current repo)

## Title

Institutional ops hardening pass: remove live/paper stubs, unify exception handling with error taxonomy + dependency circuit breakers + bounded retries, add OMS pre-trade controls (collars/size/dupes/throttles), cancel-all + order hygiene, alerts + run manifest, plus profitability hardening (TCA, replay, walk-forward, order types, portfolio risk, allocation, liquidity regime, post-trade learning, quarantine, config snapshots)

## Repository Context

Repo: `ai-trading-bot` (Dom). Bot runs 24/7 but trades RTH-only on paper.

Recent PR (first prompt) added/updated:

- `ai_trading/core/horizons.py`
- `ai_trading/core/netting.py`
- `ai_trading/core/data_contract.py`
- `ai_trading/oms/ledger.py`
- `ai_trading/oms/reconcile.py`
- `ai_trading/analytics/attribution.py`
- `ai_trading/core/bot_engine.py` updated but remains monolithic and contains fallback stubs and many broad exception handlers.

Existing relevant infra:

- Kill switch / safety monitoring: `ai_trading/safety/monitoring.py`
- Risk breakers: `ai_trading/risk/circuit_breakers.py`
- Deterministic order id helpers: `ai_trading/core/order_ids.py` (present)
- Order intent decision logic exists inside `ai_trading/core/bot_engine.py` (`OrderIntentDecision`, `_resolve_order_intent`).

Important constraint: **NO SHIMS.** Do not add facades or compatibility wrappers. Refactor directly and fail fast.

## Problem Statement

Even after the "institutional core loop" PR, the bot is not institutional-grade until it has:

1. **Fail-fast runtime contract in paper/live:** no stubbed HTTP/session/execution engine behavior reachable in paper/live.
2. **Exception handling discipline:** replace scattered broad `except Exception` behavior with a consistent taxonomy and actions (retry/skip symbol/disable provider/halt).
3. **Dependency circuit breakers:** separate breakers for broker submit, broker reads, each data provider; open circuits block trading with explicit reason codes.
4. **Safe retry policy:** bounded retries for idempotent operations only; order submit retries only when verified safe via deterministic `client_order_id` and ledger/broker lookup.
5. **OMS pre-trade controls:** price collars, max order size/shares, duplicate suppression, message-rate throttles, cancel/replace loop guard.
6. **Cancel-all & order hygiene:** kill switch can optionally cancel all open orders; startup/close cleanup.
7. **Monitoring + alerts + run manifest:** structured "halt reason", alert thresholds, and a startup run manifest that captures exact config + commit + flags.
8. **Profit hardening:** you cannot maximize profit without measuring true costs (TCA), validating the *live* pipeline by replay, and adding portfolio allocation + liquidity awareness + post-trade learning and quarantine.

## Scope of Work

### A) Remove stub/fallback behavior from paper/live (fail-fast contract)

Current issue: `ai_trading/core/bot_engine.py` still defines `_RequestsStub`, `_HTTPStub`, and `_RuntimeExecutionEngineStub`.

Required changes:

1. Introduce a single helper in a new module:

- `ai_trading/core/runtime_contract.py`
    - `is_testing_mode() -> bool` based on `AI_TRADING_TESTING=1`
    - `require_dependency(condition: bool, message: str)` that raises `RuntimeError` in paper/live
    - `require_no_stubs(ctx)` checks for any `_IS_STUB` flags, `_HTTP_SESSION_STUB`, `_REQUESTS_STUB`, etc.

2. In `ai_trading/core/bot_engine.py`:

- Keep stubs only under `if is_testing_mode(): ...` or in tests.
- In paper/live (default), if `requests` is missing or HTTP session can't be created, raise at startup (or at module import) with a clear message.
- Remove the ability to attach `_RuntimeExecutionEngineStub` in paper/live. If execution engine is missing/unavailable, raise and prevent the bot from starting.

Acceptance check:

- Starting the bot in paper/live with missing `requests` or missing execution engine must fail immediately (not "trade quietly wrong").

## B) Centralize exception classification + actions

Create:

- `ai_trading/core/errors.py`
- `ai_trading/core/retry.py`
- `ai_trading/core/dependency_breakers.py`

`ai_trading/core/errors.py` implement:

- `ErrorCategory` enum:
    - `TRANSIENT_NETWORK`, `RATE_LIMIT`, `AUTH`, `BAD_DATA`, `PROVIDER_SCHEMA`,
    - `ORDER_REJECTED`, `BROKER_STATE`, `INVARIANT_VIOLATION`, `PROGRAMMING_ERROR`, `UNKNOWN`
- `ErrorScope` enum: `SYMBOL`, `PROVIDER`, `GLOBAL`
- `ErrorAction` enum:
    - `RETRY`, `SKIP_SYMBOL`, `DISABLE_PROVIDER`, `HALT_TRADING`

- ErrorInfo dataclass:
    - category, scope, action, retryable
    - dependency (string key, see breakers)
    - reason_code
    - details (small dict)
- classify_exception(exc, *, dependency, symbol=None, sleeve=None) -> ErrorInfo
    - Explicitly map:
        - auth failures (401/403) → AUTH, HALT_TRADING, reason AUTH_HALT
        - rate limit (429) → RATE_LIMIT, RETRY, reason RATE_LIMIT_RETRY
        - timeouts/conn errors → TRANSIENT_NETWORK, RETRY, reason NET_RETRY
        - data validation failures / schema mismatch → BAD_DATA or PROVIDER_SCHEMA
        - any NaNs/inf/negative prices in netting/math → INVARIANT_VIOLATION, HALT_TRADING
        - any TypeError/KeyError in core pipeline → PROGRAMMING_ERROR, HALT_TRADING

ai_trading/core/dependency_breakers.py implement:

Dependency-aware circuit breakers distinct from drawdown/volatility breakers.

- Dependency keys (minimum):
    - broker_submit
    - broker_positions
    - broker_open_orders
    - data_primary (alpaca)
    - data_fallback_yahoo (if used)
    - quotes_primary (alpaca quotes if separate)
    - calendar_market_clock (if market-open logic can fail)
- BreakerState per dependency:
    - rolling-window failure counts
    - opened_until
    - last_error_info

Methods:

- allow(dep) -> bool

- `record_success(dep)`
- `record_failure(dep, error_info)`

Escalation thresholds (minimum):

- 3 failures / 60s → open 60s
- 10 failures / 10m → open 900s

`ai_trading/core/retry.py` implement:

Bounded retry for idempotent operations:

- `retry_idempotent(fn, *, dep, breakers, classify_exception, max_attempts, max_total_seconds, base_delay, jitter, context)`

Rules:

- Use only for reads (bars fetch, positions, order status).
- Do NOT auto-retry order submissions unless safe:
  - deterministic `client_order_id` exists AND
  - ledger says not submitted AND/OR broker query confirms not accepted.

## C) Refactor exception handling in bot_engine.py (surgical, not a rewrite)

Goal: reduce "random broad catches" by routing failures through `classify_exception` and enforcing a consistent action.

Required changes in `ai_trading/core/bot_engine.py`:

1. Define a small helper:

- `_handle_error(error_info, *, state, ctx, symbol=None, sleeve=None) -> None`

Applies action:

- `SKIP_SYMBOL`: record reason in decision record and continue
- `DISABLE_PROVIDER`: mark provider disabled (and/or open breaker) and use fallback if allowed

- `HALT_TRADING`: set a global halt flag (or reuse kill switch mechanism) and block new orders

2. Replace patterns like:

- `except Exception: pending_orders = []`
- `except Exception: continue`
- `logger.debug(...); pass`

with:

- `error_info = classify_exception(e, dependency=..., symbol=...)`
- `breakers.record_failure(dep, error_info)`
- `_handle_error(error_info, ...)`

3. Tighten the most dangerous areas first:

- broker order submission
- broker reads (positions/open orders)
- data fetch
- quote fetch (if used for quote gate)
- netting math / risk math

Key policy:

- If exception category is `INVARIANT_VIOLATION` or `PROGRAMMING_ERROR`: halt trading immediately (paper too).

## D) OMS pre-trade controls (build on existing "order intent" logic)

You already have `_resolve_order_intent` and `OrderIntentDecision`. Add pre-trade validation at the OMS boundary.

Create: `ai_trading/oms/pretrade.py`

Implement:

- `OrderIntent` dataclass (or reuse existing fields in decision record):
  - `symbol`, `side`, `qty`, `notional`, `limit_price`, `bar_ts`, `client_order_id`
  - `last_price`/`mid`, `spread`, sleeve attribution (optional)
- `validate_pretrade(intent, *, cfg, ledger, rate_limiter) -> (allowed: bool, reason_code: str, details: dict)`

Pre-trade checks (hard blocks):

1. MAX_ORDER_DOLLARS, MAX_ORDER_SHARES
2. PRICE_COLLAR_PCT against last/mid
3. Duplicate suppression:
    a. if ledger already has same client_order_id OR same (symbol, side, qty, bar_ts)
4. Message-rate throttles:
    a. global orders/min
    b. per-symbol orders/min
    c. cancels/min
5. Cancel/replace loop guard:
    a. if too many cancels without fills, block symbol for N bars (signal-aligned)

Integrate:

- Immediately before the actual broker submit call.
- If blocked, record reason in decision record + metrics.

Reason codes:

- ORDER_SIZE_BLOCK, PRICE_COLLAR_BLOCK, DUPLICATE_ORDER_BLOCK,

RATE_THROTTLE_BLOCK, CANCEL_LOOP_BLOCK

## E) Cancel-all + order hygiene (integrate with existing safety module)

You already have ai_trading/safety/monitoring.py and kill switch keys in config/runtime.

Add:

- ai_trading/oms/cancel_all.py
    - cancel_all_open_orders(ctx) -> CancelAllResult

Wire it:

- If kill switch active and AI_TRADING_CANCEL_ALL_ON_KILL=true, call cancel-all once (rate-limited).

- If reconciliation mismatch or `broker_submit` breaker escalation triggers, optionally cancel all.

Session boundary hygiene:

- At market close:
    - optionally cancel DAY orders
    - reconcile
    - write summary record
- On startup:
    - reconcile open orders and (optionally) cancel unexpected leftovers

Reason codes:

- `CANCEL_ALL_TRIGGERED`, `SESSION_CLOSE_CLEANUP`, `STARTUP_CLEANUP`

## F) Alerts + halt reason + run manifest

Alerts:

- Extend existing monitoring instead of duplicating.
- Add thin module: `ai_trading/monitoring/alerts.py`
    - threshold rules for:
        - breaker open (critical dep)
        - repeated rejects
        - data stale
        - recon mismatch
        - exception category HALT_TRADING
    - emits structured ALERT_* log events
    - sets `state.halt_reason`

Run manifest:

- Create `ai_trading/runtime/run_manifest.py`
- On startup write JSON:
    - timestamp
    - mode (paper/live)
    - account id (safe)

    - resolved config hash (secrets redacted)
    - enabled feature flags
    - git commit hash (best-effort)
    - runtime contract (stubs enabled? false)

If canary mode configured:

- `AI_TRADING_CANARY_SYMBOLS`
- enforce trading only for those symbols, log `CANARY_MODE_ACTIVE`

## Acceptance Criteria (base)

1. Paper/live startup fails fast if HTTP/session or execution engine is missing; no stub fallback reachable unless `AI_TRADING_TESTING=1`.
2. `bot_engine.py` uses centralized exception classification + dependency breakers for broker and data operations.
3. Bounded retries exist and are used only for idempotent reads.
4. Order submit path enforces pre-trade controls (collars/size/dupes/throttles).
5. Kill switch blocks new orders; optional cancel-all works and is logged.
6. Session close/startup hygiene behavior exists and is configurable.
7. Alerts and halt reasons are explicit, structured, and observable.
8. Run manifest is written at startup with config hash + flags + commit id.

## Validation Steps (base)

Run:

1. `python -m compileall ai_trading`
2. `pytest -q`

Add tests (keep narrow and deterministic):

- `tests/test_runtime_contract_no_stubs.py`
- `tests/test_error_classification.py`
- `tests/test_dependency_breakers.py`

- `tests/test_retry_idempotent_reads_only.py`
- `tests/test_pretrade_price_collar.py`
- `tests/test_pretrade_max_order_size.py`
- `tests/test_pretrade_duplicate_block.py`
- `tests/test_kill_switch_cancel_all.py`
- `tests/test_run_manifest_written.py`

Paper smoke checks:

- Off-hours: no submit; logs `MARKET_CLOSED_BLOCK`
- Force provider failures: breaker opens; trading blocks with `CIRCUIT_OPEN_<dep>`
- Restart mid-bar: no duplicate submit due to ledger + duplicate suppression

## Constraints & Standards

- **NO SHIMS.**
- Avoid destructive replace of `bot_engine.py`; make surgical edits and extract modules.
- All blocks produce reason codes + metrics/logs.
- In paper/live, unexpected exceptions in core decision/order pipeline must halt trading, not "continue."

# APPENDIX G) Trading mode correctness fixes (conservative / balanced / aggressive)

## Background / Verified Issues in Current Repo

The repo currently has mode-related inconsistencies that can cause silent wrong mode selection, especially when launching via `python -m ai_trading`:

1. **TRADING_MODE is overloaded**
- `ai_trading/__main__.py` sets `os.environ["TRADING_MODE"] = "paper" if args.paper else "live"` in multiple places.
- Elsewhere, TRADING_MODE is used as a strategy profile selector (`conservative|balanced|aggressive`) in `ai_trading/config/runtime.py` and `ai_trading/config/__init__.py`.
- Result: launching with `--paper` can overwrite `.env` and break mode selection or trigger fallback behavior.
2. **Aggressive profile missing in regime routing table**
- In `ai_trading/core/bot_engine.py`, `_REGIME_SIGNAL_COMPONENTS` defines entries for `"balanced"` and `"conservative"` but not `"aggressive"`.
- If `AI_TRADING_REGIME_SIGNAL_PROFILE=aggressive`, code falls back to balanced.
3. **Liquidity analyzer uses moderate key instead of balanced**
- `ai_trading/execution/liquidity.py` uses: `{'conservative': ..., 'moderate': ..., 'aggressive': ...}`
- If code supplies `"balanced"`, it may miss the intended thresholds or fall back incorrectly.

These must be corrected to ensure "institutional-grade" behavior and to prevent silent misconfiguration.

## Required Changes (NO SHIMS)

### G1) Fix TRADING_MODE env collision (paper/live should not overwrite mode profile)

Goal: TRADING_MODE must ONLY be used for `conservative|balanced|aggressive`.

Paper/live selection must use `EXECUTION_MODE` (already present in `.env`) or a new env var like APP_ENV, but do NOT reuse TRADING_MODE.

Tasks:

1. In `ai_trading/__main__.py`:

- Remove any code that sets `os.environ["TRADING_MODE"] = "paper"` or `"live"`.
- If a CLI flag needs to force paper/live behavior, set:
  - `os.environ["EXECUTION_MODE"] = "paper"|"live"` (preferred), OR
  - introduce `os.environ["APP_ENV"] = "paper"|"live"` (only if required)
- Ensure no other paths write `TRADING_MODE` to `"paper"` or `"live"`.

2. In `ai_trading/config/runtime.py`:

- Ensure `_infer_paper_mode()` (or equivalent) does NOT interpret `TRADING_MODE` as an execution environment marker.
- Execution environment inference must come from `EXECUTION_MODE` only.

Acceptance:

- Launching with CLI `--paper` does NOT change `cfg.trading_mode`.
- `TRADING_MODE=balanced` remains balanced regardless of paper/live mode selection.

Tests:

- Add `tests/test_trading_mode_not_overwritten_by_cli.py`:
  - Simulate CLI parse (or call entry function) with paper mode
  - Assert `TRADING_MODE` remains balanced (or as set)
  - Assert `EXECUTION_MODE` is paper.

## G2) Add aggressive entry to _REGIME_SIGNAL_COMPONENTS (or hard reject)

Goal: If `AI_TRADING_REGIME_SIGNAL_PROFILE=aggressive`, it must be a real profile — not silently mapped to balanced.

Tasks:

- In `ai_trading/core/bot_engine.py`:
  - Add `"aggressive"` key to `_REGIME_SIGNAL_COMPONENTS` with explicit component set.
  - Aggressive enables more components and/or reduces dampening compared to balanced.

    - Conservative remains strictest.
    - If profile not in table, log explicit warning with reason code `UNKNOWN_REGIME_PROFILE_FALLBACK` and fallback to balanced.

Acceptance:

- Setting `AI_TRADING_REGIME_SIGNAL_PROFILE=aggressive` results in aggressive routing.
- No silent fallback to balanced without visible log/decision record reason.

Tests:

- Add `tests/test_regime_profile_aggressive_present.py`.

### G3) Fix liquidity analyzer mode key mismatch (moderate → balanced)

Goal: Use consistent naming: `conservative|balanced|aggressive`.

Tasks:

- In `ai_trading/execution/liquidity.py`:
    - Rename key `"moderate"` to `"balanced"`.
    - Ensure any call sites pass balanced rather than moderate.
    - Update any other internal reference to `"moderate"`.

Acceptance:

- Liquidity mode thresholds selected when `TRADING_MODE=balanced`.
- No lingering `"moderate"` in runtime paths.

Tests:

- Add `tests/test_liquidity_mode_balanced_supported.py`.

# APPENDIX H) Transaction Cost Analysis (TCA) + execution-quality feedback loop

## Goal

Maximize profit by measuring and minimizing *real* trading costs and execution degradation. This is not optional: without TCA you don't know whether you're profitable after costs.

## Required changes

### H1) Add TCA module

Create:

- `ai_trading/analytics/tca.py`

Implement:

- `ExecutionBenchmark` dataclass:
  - `arrival_price` (decision/submit time)
  - `mid_at_arrival` (if available; else best-effort proxy)
  - `bid_at_arrival`, `ask_at_arrival` (if available)
  - `bar_close_price` (if decision is bar-close)
  - timestamps (decision_ts, submit_ts, first_fill_ts)
- `FillSummary` dataclass:
  - VWAP fill price, total qty, fees, status, partial-fill flags
- Core functions:
  - `implementation_shortfall_bps(side, arrival_price, fill_vwap, fees, qty)`
  - `spread_paid_bps(side, mid_at_arrival, fill_vwap)`
  - `fill_latency_ms(submit_ts, first_fill_ts)`
  - `cancel_replace_rate(window)` (derived metric)
- Output record:
  - One JSON record per order/fill keyed by `client_order_id`

## H2) Add daily execution report

Create:

- `ai_trading/analytics/execution_report.py`

Implement:

- Daily aggregation by:
    - symbol
    - sleeve
    - regime profile
    - order type
    - provider (primary/fallback)
- Produce:
    - JSONL + CSV summary (or JSON only if simpler)
    - percentiles for IS bps and spread paid bps
    - counts for rejects, cancels, partial fills, “blocked by gates”

## H3) Wire to decision records

In your core loop / OMS boundary:

- Ensure each decision record stores:
    - `arrival_price` definition used
    - `mid/bid/ask` if available
    - order submit timestamp
    - order fill summary
    - TCA metrics once fills occur

Acceptance:

- After one trading day, you can answer:
    - “Are we profitable before costs but losing after costs?”
    - “Which symbols/sleeves bleed the most IS bps?”
    - “Are cancels/rejects harming PnL?”

# APPENDIX I) Replay harness using the live pipeline

## Goal

Backtests lie. Validate the *actual live decision+OMS pipeline* deterministically on historical bars (same gating, dedupe, stop locks, cost gates, turnover caps, ledger behavior, and decision records).

## Required changes

Create:

- `ai_trading/replay/replay_engine.py`

Implement:

- `ReplayConfig`:
  - symbols, start/end, timeframes, RTH-only behavior, seed, speed controls
  - paper-only execution (no broker submits by default; allow "simulate fills" with deterministic fill model)
- `ReplayEngine`:
  - feeds bar-close events to the same sleeve → netting → order intent → pretrade gate pipeline
  - writes the same decision records + OMS ledger outputs
  - produces TCA metrics using:
    - either recorded historical bid/ask (if you have it)
    - or conservative proxy (spread from high/low or configured bps) — must be explicit and logged as proxy

Rules:

- Replay must be deterministic:
  - same inputs produce identical decision records across runs
- Replay must use same reason codes and gating:
  - market closed blocks

    - cost gate
    - turnover cap
    - stop lock
    - dedupe (per bar + ledger)

Acceptance:

- “30-day replay” runs and produces:
    - trades/day, flip rate, blocked-by-cost-gate %, IS bps distribution
    - identical outputs across repeated runs

# APPENDIX J) Walk-forward evaluation + leakage guards

## Goal

Ensure research results match out-of-sample trading reality. Prevent leakage and optimize for post-cost outcomes.

## Required changes

Create:

- `ai_trading/research/walk_forward.py`
- `ai_trading/research/leakage_tests.py`

### J1) Walk-forward evaluation

Implement:

- rolling windows:
    - train window length
    - test window length
    - optional embargo buffer
- outputs:

    - distribution of:
        - post-cost return
        - turnover
        - drawdown
        - hit rate
        - stability metrics (variance across folds)

### J2) Leakage guards

Implement tests that fail CI if:

- any feature uses future timestamps relative to label horizon
- any overlapping label horizon contaminates training
- embargo/purging is not applied when horizons overlap

Acceptance:

- Walk-forward produces a *distribution* (not single best run)
- CI fails loudly when leakage is introduced

# APPENDIX K) Order-type upgrades to reduce churn + enforce exits

## Goal

Reduce churn and enforce exits cleanly with broker-native order types when supported; otherwise fail fast or keep simple order types but do not fake support.

## Required changes

Create:

- `ai_trading/oms/orders.py`

Implement:

- explicit constructors/builders:
    - `build_limit(...)`
    - `build_market(...)` (if allowed by your execution policy)
    - `build_stop_limit(...)`
    - `build_stop(...)`
    - `build_trailing_stop(...)`
    - `build_bracket(...)` (entry + take profit + stop loss), **only if broker supports**
    - `build_oco(...)` / `build_oto(...)` **only if broker supports**

Integrate:

- OMS boundary chooses order family based on:
    - sleeve
    - volatility regime (from Appendix O)
    - risk policy
    - broker capabilities (queried or configured; must not be guessed silently)

Rules:

- If an order type is configured but broker does not support it in the configured way:
    - fail fast at startup (paper/live), or block trading with explicit reason code
- Decision record must include:
    - order family details (legs if any)
    - exit parameters used (stop distance, trailing amount)
    - reconciliation outcome for open legs

Acceptance:

- Exits become consistent and measurable (via TCA + decision records)
- Stop-out re-entry loops are reduced by:
    - structured exits + stop locks

# APPENDIX L) Portfolio-level risk targeting

## Goal

Max profit with survivability requires portfolio-level controls beyond per-symbol caps.

## Required changes

Create:

- `ai_trading/risk/portfolio_limits.py`

Implement:

- Volatility targeting:
    - compute realized vol proxy from recent returns
    - scale net exposure to target vol (configurable)
- Concentration caps:
    - limit exposure per symbol and per cluster
- Correlation proxy caps:
    - basic rolling correlation across held symbols
    - cap aggregate exposure to highly correlated group

Integrate:

- after netting but before order intent:
    - apply portfolio scaling and caps
    - record reason codes:
        - `RISK_CAP_PORTFOLIO`, `VOL_TARGET_SCALE`, `CORR_CLUSTER_CAP`

Acceptance:

- Portfolio-level caps observable and enforced before submitting orders
- Decision records show risk scaling/caps applied

# APPENDIX M) Model governance + artifact security

## Goal

Prevent unsafe model loads, tampering, and “auto-train in live.” Make model artifacts verifiable.

## Required changes

Create:

- `ai_trading/models/artifacts.py`

Implement:

- Artifact manifest:
  - model version
  - checksum(s)
  - created timestamp
  - training data range (non-sensitive metadata)
- Verification:
  - verify checksums before loading
  - optional signature verification if you add keys later

Live behavior:

- If verification fails in live:
  - fail closed (halt trading)
  - reason `MODEL_VERIFICATION_FAILED`
- Auto-train:
  - allowed only if `AI_TRADING_ALLOW_PAPER_TRAIN=1` and execution mode is paper
  - never auto-train in live

Acceptance:

- Live cannot silently swap models
- Tampered artifacts are rejected deterministically

# APPENDIX N) Strategy capital allocation / AUM routing

## Goal

Pros allocate capital dynamically based on sleeve performance and stability, not fixed weights.

## Required changes

Create:

- `ai_trading/portfolio/allocation.py`

Implement:

- `SleevePerfState`:
    - rolling expectancy (post-cost)
    - drawdown
    - stability score (variance of outcomes)
    - trade count and confidence
- Weighting logic:
    - base weights per sleeve
    - adjust within bounds:
        - decrease weight when expectancy < threshold or drawdown > threshold
        - increase weight when stable positive expectancy is observed
    - hard floors/ceilings to avoid overreaction
- Update schedule:
    - daily at close (or first run after close), and logged

Integrate:

- netting phase uses `allocation_weights` to scale sleeve proposals before summing
- every decision record includes the weights used

Acceptance:

- Sleeve weights evolve deterministically based on measured outcomes
- Risk-weighted performance improves vs static weights

# APPENDIX O) Market impact / liquidity regime awareness

## Goal

Control market impact and avoid trading too large relative to liquidity.

## Required changes

Create:

- `ai_trading/risk/liquidity_regime.py`

Implement:

- rolling liquidity features:
  - rolling volume
  - spread proxy
  - volatility proxy
- participation cap:
  - block or slice if `abs(order_qty)` exceeds X% of rolling volume
- liquidity regime classification:
  - `THIN`, `NORMAL`, `THICK` (or simple numeric score)

Integrate:

- pretrade gate and cost model:
    - widen cost assumptions in THIN regime
    - enforce stricter collars / smaller max size in THIN regime
- decision records include liquidity regime

Reason codes:

- `LIQ_PARTICIPATION_BLOCK`, `LIQ_REGIME_THIN_SCALE`

Acceptance:

- Orders respect participation caps
- Thin-liquidity periods reduce churn and slippage

# APPENDIX P) Post-trade learning loop (measurement → adaptation)

## Goal

Use TCA + outcomes to adjust gates and reduce unprofitable behavior automatically (within safe bounds).

## Required changes

Create:

- `ai_trading/analytics/post_trade_learning.py`

Implement:

- daily learning job that reads:
    - decision records
    - TCA records
    - realized outcomes per sleeve/symbol

- safe bounded adjustments:
  - if IS bps is persistently high for a symbol → increase `ENTRY_COST_BUFFER_BPS` (per symbol)
  - if flip rate too high → increase deadband / require more confirmation signals
  - if turnover is too high → tighten turnover caps / cost multipliers
- store learned adjustments:
  - write to `runtime/learned_overrides.json` (or similar)
  - must include timestamp and rollback fields

Rules:

- no unbounded parameter drift
- cap daily parameter deltas
- always log changes with reason + metrics

Acceptance:

- Learning updates are logged and versioned
- Decision records include “overrides applied”

# APPENDIX Q) Strategy quarantine logic

## Goal

If a sleeve is misbehaving, disable it temporarily to protect capital.

## Required changes

Create:

- `ai_trading/runtime/quarantine.py`

Implement:

- per sleeve/symbol quarantine state:
  - start_ts, end_ts

    - trigger reason
    - metrics snapshot at trigger time
- triggers (configurable):
    - repeated breaker opens
    - reject rate above threshold
    - persistent negative expectancy (post-cost)
    - repeated cancel-loop blocks
- behavior:
    - quarantined sleeves do not submit proposals (or proposals are forced to zero)
    - decision record notes quarantine and reason codes

Reason codes:

- `SLEEVE_QUARANTINED`, `SYMBOL_QUARANTINED`

Acceptance:

- Quarantine is deterministic, logged, and time-bounded
- Prevents extended bleed during bad regimes

# APPENDIX R) Deterministic configuration snapshot in decision records

## Goal

You must be able to audit exactly what rules produced each trade.

## Required changes

Update decision record schema to include:

- resolved trading mode (`conservative|balanced|aggressive`)
- regime signal profile and components chosen
- sleeve configs at decision time:

    - thresholds, deadbands, cost_k, edge scale, turnover caps
- global caps:
    - `GLOBAL_MAX_SYMBOL_DOLLARS`, gross/net caps
- portfolio risk settings:
    - vol target, correlation caps
- learned overrides applied (Appendix P)
- allocation weights used (Appendix N)
- liquidity regime and participation settings (Appendix O)

Acceptance:

- Every decision record is fully reproducible from its snapshot

## Updates to Acceptance Criteria (extend your base list)

Add:

9) TRADING_MODE is no longer overwritten by CLI paper/live selection; paper/live uses EXECUTION_MODE only.

10) “aggressive” regime profile is implemented (or explicitly rejected) — no silent fallback.

11) Liquidity execution mode keys are consistent (balanced, not moderate).

12) TCA metrics are produced per order and daily aggregated.

13) Replay harness runs deterministically and emits decision records + TCA.

14) Walk-forward + leakage checks run and fail CI on leakage.

15) Post-trade learning updates are logged, bounded, versioned, and snapshotted into decision records.

16) Quarantine logic blocks unstable sleeves deterministically with explicit reason codes.

17) Decision records contain deterministic config snapshots including allocation weights and learned overrides.

## Updates to Validation Steps (extend)

Add tests:

- `tests/test_trading_mode_not_overwritten_by_cli.py`
- `tests/test_regime_profile_aggressive_present.py`
- `tests/test_liquidity_mode_balanced_supported.py`
- `tests/test_tca_implementation_shortfall.py`
- `tests/test_execution_report_daily_rollup.py`
- `tests/test_replay_engine_deterministic.py`
- `tests/test_walk_forward_no_leakage.py`
- `tests/test_order_types_supported_and_failfast.py`
- `tests/test_portfolio_limits_vol_targeting.py`
- `tests/test_allocation_weight_updates.py`
- `tests/test_liquidity_participation_block.py`
- `tests/test_post_trade_learning_bounded_updates.py`
- `tests/test_quarantine_triggers_and_blocks.py`
- `tests/test_decision_record_config_snapshot.py`

Paper smoke checks (extend):

- Run replay over last N days and compare:
  - trade counts, flip rate, blocked reasons distributions
- Force high spread / thin liquidity conditions and verify:
  - participation block and stricter collars engage
- Trigger quarantine and verify:
  - sleeve proposals zero out with reason codes

Defaults are conservative and paper-safe.

```
# ==============================================================
# ADDITIONS for Second Codex Prompt Appendices H–R
# (TCA, Replay, Walk-forward, Order Types, Portfolio Risk,
#  Allocation, Liquidity Regime, Post-trade Learning, Quarantine,
#  Decision Record Config Snapshots)
# ============================================================

# ===== APPENDIX H: TCA / EXECUTION QUALITY =====
AI_TRADING_TCA_ENABLED=1
AI_TRADING_TCA_PATH=runtime/tca_records.jsonl
AI_TRADING_EXECUTION_REPORT_ENABLED=1
AI_TRADING_EXECUTION_REPORT_DIR=runtime/execution_reports
AI_TRADING_EXECUTION_REPORT_FORMATS=json,csv
AI_TRADING_EXECUTION_REPORT_ROLLUP_TZ=America/New_York

# Benchmark definition for implementation shortfall
# arrival = "decision" (bar close / signal time) or "submit" (order
submit time)
AI_TRADING_TCA_ARRIVAL_BENCHMARK=decision

# If NBBO not available, allow a proxy mid/spread model (must be
explicit + logged)
AI_TRADING_TCA_ALLOW_PROXY_QUOTES=1
AI_TRADING_TCA_PROXY_SPREAD_BPS_DEFAULT=12
AI_TRADING_TCA_PROXY_MID_SOURCE=last_trade  #
last_trade|bar_close|last_close

# How long to wait for fills before writing a "pending" TCA record
AI_TRADING_TCA_PENDING_WRITE_SEC=60
AI_TRADING_TCA_UPDATE_ON_FILL=1

# ===== APPENDIX I: REPLAY HARNESS (LIVE PIPELINE REPLAY) =====
AI_TRADING_REPLAY_ENABLED=0
AI_TRADING_REPLAY_DATA_DIR=runtime/replay_data
AI_TRADING_REPLAY_OUTPUT_DIR=runtime/replay_outputs
AI_TRADING_REPLAY_SEED=42
AI_TRADING_REPLAY_RTH_ONLY=1
AI_TRADING_REPLAY_SPEEDUP=1         # 1 = as fast as possible
AI_TRADING_REPLAY_TIMEFRAMES=5Min,1Hour,1Day
AI_TRADING_REPLAY_SYMBOLS=          # optional CSV; empty uses normal
universe
AI_TRADING_REPLAY_START_DATE=       # YYYY-MM-DD
```

```
AI_TRADING_REPLAY_END_DATE=         # YYYY-MM-DD

# Replay fill modeling
AI_TRADING_REPLAY_SIMULATE_FILLS=1  # default paper-safe; no broker
submits
AI_TRADING_REPLAY_FILL_MODEL=next_bar_mid  #
next_bar_mid|next_bar_vwap|close
AI_TRADING_REPLAY_FILL_SLIPPAGE_BPS=5
AI_TRADING_REPLAY_FILL_FEE_BPS=0

# If simulate fills, still enforce the full OMS gates (pretrade,
dedupe, etc.)
AI_TRADING_REPLAY_ENFORCE_OMS_GATES=1

# ===== APPENDIX J: WALK-FORWARD + LEAKAGE GUARDS =====
AI_TRADING_WALK_FORWARD_ENABLED=0
AI_TRADING_WF_TRAIN_DAYS=180
AI_TRADING_WF_TEST_DAYS=30
AI_TRADING_WF_STEP_DAYS=30
AI_TRADING_WF_EMBARGO_DAYS=5
AI_TRADING_WF_COST_MODEL=from_tca  # from_tca|fixed
AI_TRADING_WF_FIXED_COST_BPS=15
AI_TRADING_WF_OUTPUT_DIR=runtime/walk_forward

# Hard fail CI if leakage checks fail
AI_TRADING_LEAKAGE_GUARDS_ENABLED=1
AI_TRADING_LEAKAGE_FAIL_HARD=1

# ===== APPENDIX K: ORDER TYPES / STRUCTURED EXITS =====
AI_TRADING_ORDER_TYPES_ENABLED=1

# Default order type policy (must match broker capabilities; fail-fast
if invalid)
AI_TRADING_DEFAULT_ENTRY_ORDER_TYPE=limit        # limit|market (if
allowed)
AI_TRADING_DEFAULT_EXIT_ORDER_TYPE=stop_limit    #
stop|stop_limit|trailing_stop
AI_TRADING_ALLOW_BRACKET_ORDERS=0
AI_TRADING_ALLOW_OCO_OTO=0
```

```
# Exit distances (sleeve-specific overrides can be added later)
AI_TRADING_STOP_LOSS_ATR_MULT=1.6
AI_TRADING_TAKE_PROFIT_ATR_MULT=2.4
AI_TRADING_TRAILING_STOP_ATR_MULT=1.2

# Guardrails for structured orders
AI_TRADING_EXIT_LEG_RECONCILE_REQUIRED=1
AI_TRADING_ORDER_TYPE_FAILFAST_ON_UNSUPPORTED=1

# ===== APPENDIX L: PORTFOLIO-LEVEL RISK TARGETING =====
AI_TRADING_PORTFOLIO_LIMITS_ENABLED=1
AI_TRADING_VOL_TARGETING_ENABLED=1
AI_TRADING_TARGET_ANNUAL_VOL=0.18
AI_TRADING_VOL_LOOKBACK_DAYS=20
AI_TRADING_VOL_MIN_SCALE=0.25
AI_TRADING_VOL_MAX_SCALE=1.25

# Concentration / correlation caps
AI_TRADING_CONCENTRATION_CAP_ENABLED=1
AI_TRADING_MAX_SYMBOL_WEIGHT=0.12            # fraction of gross
exposure
AI_TRADING_MAX_CLUSTER_WEIGHT=0.25           # if clusters available
AI_TRADING_CORR_CAP_ENABLED=1
AI_TRADING_CORR_LOOKBACK_DAYS=30
AI_TRADING_CORR_THRESHOLD=0.80
AI_TRADING_CORR_GROUP_GROSS_CAP=0.35         # cap for highly
correlated cluster

# ===== APPENDIX N: SLEEVE CAPITAL ALLOCATION / AUM ROUTING =====
AI_TRADING_ALLOCATION_ENABLED=1
AI_TRADING_ALLOCATION_UPDATE_SCHEDULE=market_close  #
market_close|daily_first_run|manual
AI_TRADING_ALLOCATION_OUTPUT_PATH=runtime/allocation_state.json

# Allocation bounds + inertia
AI_TRADING_ALLOC_DAY_BASE_WEIGHT=0.40
AI_TRADING_ALLOC_SWING_BASE_WEIGHT=0.35
AI_TRADING_ALLOC_LONGSHORT_BASE_WEIGHT=0.25
```

```
AI_TRADING_ALLOC_MIN_WEIGHT=0.10
AI_TRADING_ALLOC_MAX_WEIGHT=0.60
AI_TRADING_ALLOC_DAILY_MAX_DELTA=0.05         # prevents overreaction

# Performance inputs
AI_TRADING_ALLOC_USE_POSTCOST_EXPECTANCY=1
AI_TRADING_ALLOC_EXPECTANCY_WINDOW_TRADES=60
AI_TRADING_ALLOC_MIN_TRADES_FOR_ADJUST=15
AI_TRADING_ALLOC_DRAWDOWN_TRIGGER=0.06
AI_TRADING_ALLOC_EXPECTANCY_FLOOR_PCT=-0.0010  # below this reduce
weight

# ===== APPENDIX O: LIQUIDITY REGIME / MARKET IMPACT =====
AI_TRADING_LIQ_REGIME_ENABLED=1
AI_TRADING_LIQ_LOOKBACK_BARS=60

# Participation cap: block/scale if order > X% of rolling volume
AI_TRADING_PARTICIPATION_CAP_ENABLED=1
AI_TRADING_MAX_PARTICIPATION_PCT=0.015        # 1.5% of rolling volume
AI_TRADING_PARTICIPATION_BLOCK_MODE=block     # block|scale
AI_TRADING_PARTICIPATION_SCALE_MIN=0.25       # if scale mode, min
scaling

# Regime thresholds (simple defaults)
AI_TRADING_LIQ_THIN_SPREAD_BPS=25
AI_TRADING_LIQ_THIN_VOL_MULT=1.8
AI_TRADING_LIQ_THIN_MAX_ORDER_DOLLARS=5000

# Liquidity regime impacts on costs/collars
AI_TRADING_LIQ_THIN_COST_MULT=1.3
AI_TRADING_LIQ_THIN_COLLAR_MULT=0.8           # tighter collars in
thin regime

# ===== APPENDIX P: POST-TRADE LEARNING (BOUNDED ADAPTATION) =====
AI_TRADING_POST_TRADE_LEARNING_ENABLED=1
AI_TRADING_LEARNING_OUTPUT_PATH=runtime/learned_overrides.json
AI_TRADING_LEARNING_APPLY_OVERRIDES=1
AI_TRADING_LEARNING_RUN_SCHEDULE=market_close  #
```

```
market_close|daily_first_run|manual

# Safety bounds on parameter drift
AI_TRADING_LEARNING_MAX_DAILY_DELTA_BPS=3
AI_TRADING_LEARNING_MAX_DAILY_DELTA_FRAC=0.05
AI_TRADING_LEARNING_MAX_OVERRIDE_AGE_DAYS=30

# Learning triggers
AI_TRADING_LEARNING_IS_BPS_TRIGGER=18
AI_TRADING_LEARNING_FLIP_RATE_TRIGGER=0.25    # flips / trades
AI_TRADING_LEARNING_TURNOVER_TRIGGER_DOLLARS=100000

# What learning is allowed to adjust
AI_TRADING_LEARNING_ADJUST_COST_BUFFER=1
AI_TRADING_LEARNING_ADJUST_DEADBANDS=1
AI_TRADING_LEARNING_ADJUST_CONFIRM_SIGNALS=1
AI_TRADING_LEARNING_ADJUST_TURNOVER_CAPS=0    # keep off until stable

# ===== APPENDIX Q: QUARANTINE (AUTO-DISABLE MISBEHAVING SLEEVES)
=====
AI_TRADING_QUARANTINE_ENABLED=1
AI_TRADING_QUARANTINE_STATE_PATH=runtime/quarantine_state.json

# Quarantine durations
AI_TRADING_QUARANTINE_DURATION_BARS_DAY=6
AI_TRADING_QUARANTINE_DURATION_BARS_SWING=3
AI_TRADING_QUARANTINE_DURATION_DAYS_LONGSHORT=2

# Quarantine triggers
AI_TRADING_QUARANTINE_BREAKER_OPEN_COUNT=2
AI_TRADING_QUARANTINE_REJECT_RATE_TRIGGER=0.15
AI_TRADING_QUARANTINE_COST_GATE_BLOCK_RATE_TRIGGER=0.60
AI_TRADING_QUARANTINE_EXPECTANCY_FLOOR_PCT=-0.0015
AI_TRADING_QUARANTINE_MIN_TRADES=12

# Behavior when quarantined
AI_TRADING_QUARANTINE_MODE=block  # block|zero_targets
AI_TRADING_QUARANTINE_APPLIES_TO=sleeve,symbol  # sleeve|symbol|both
```

```
# ===== APPENDIX R: DECISION RECORD CONFIG SNAPSHOTS =====
AI_TRADING_DECISION_RECORD_SNAPSHOT_ENABLED=1
AI_TRADING_DECISION_RECORD_SNAPSHOT_REDACT_SECRETS=1
AI_TRADING_DECISION_RECORD_SNAPSHOT_INCLUDE_LIMITS=1
AI_TRADING_DECISION_RECORD_SNAPSHOT_INCLUDE_ALLOCATION=1
AI_TRADING_DECISION_RECORD_SNAPSHOT_INCLUDE_LIQ_REGIME=1
AI_TRADING_DECISION_RECORD_SNAPSHOT_INCLUDE_LEARNED_OVERRIDES=1

# Optional: write separate snapshot stream (in addition to decision
records)
AI_TRADING_CONFIG_SNAPSHOT_PATH=runtime/config_snapshots.jsonl

# ===== EXTRA: PROFIT-SAFETY DEFAULTS THAT PAIR WELL WITH H-R =====
# (kept minimal; can be removed if you already set these elsewhere)
AI_TRADING_BLOCK_TRADING_IF_TCA_STALE=0
AI_TRADING_MIN_TCA_SAMPLES_FOR_ADAPT=20
AI_TRADING_REQUIRE_TCA_FOR_ALLOCATION=0
```

## Notes so you don’t get bitten later (practical, not theory)

- **Replay** is defaulted to “simulate fills” so you can validate determinism without broker calls.
- **Order types** default to conservative support. Brackets/OCO are off until you explicitly decide you want them.
- **Learning + allocation** are bounded so you don’t “overfit yourself into a hole” week-to-week.
- **Liquidity regime** defaults to *block/scale* larger orders relative to volume—this is one of the most common real profit leaks.

Perfect.

These are **precision upgrades**, not scope explosions.

We are strengthening correctness, preventing ambiguity, and protecting replay realism — without making Codex build anything new.

Below are the **exact insertions** written to match your prompt’s tone and structure.

You can paste them into your existing document under small new sections.

# APPENDIX Z — Implementation Sequencing (P0 → P1)

**Purpose:**

Reduce integration risk, preserve determinism, and maximize first-pass merge success.

Even though this document specifies a unified transformation, implementation must proceed in **two ordered phases**.

## Phase P0 — Operational Correctness & Safety (must land first)

Focus only on:

- runtime contract (no stubs reachable in paper/live)
- centralized exception taxonomy
- dependency circuit breakers
- bounded retry framework
- OMS pre-trade validation
- cancel-all wiring
- session hygiene
- alerts & halt reasons
- run manifest
- trading-mode correctness fixes

**Rule:**

P0 must compile, pass tests, and run in paper before P1 features are enabled.

No alpha, allocation, replay, or adaptive logic should be required for P0 success.

## Phase P1 — Measurement & Profit Optimization (after P0 stable)

Then implement:

- TCA / execution analytics
- replay harness
- walk-forward

- order type upgrades
- portfolio limits
- dynamic allocation
- learning loops
- quarantine logic
- config snapshots

### Acceptance requirement

Codex must ensure P0 can run independently with feature flags if needed.

## APPENDIX AA — Canonical Definition of Arrival / Decision Price

**Purpose:**

Implementation Shortfall must be computed consistently or the numbers become meaningless.

### Required definitions

For every order:

- **decision_price** = close (or last trade) of the bar that produced the forecast/signal
- **submit_price_reference** = market price observed immediately prior to broker submission
- **fill_price** = actual execution price returned by broker or simulator

### Implementation Shortfall

Minimum calculation:

```
IS_bps = sign * (fill_price - decision_price) / decision_price *
10_000
```

Where sign = +1 for buys, -1 for sells.

### Why this matters

If this definition drifts between replay and live, performance comparisons become invalid.

### Acceptance requirement

All TCA modules must use this exact definition unless explicitly overridden by config.

## APPENDIX AB — Replay Must Never Hit Real Broker

**Purpose:**

Guarantee deterministic simulation and prevent accidental live behavior during research or CI.

### Hard rule

When in replay mode:

- broker submit functions MUST NOT execute
- network calls for orders MUST NOT occur
- simulated fills must be used
- ledger entries must be marked `replay=true`

If any real submission path is invoked → raise RuntimeError.

## Allowed during replay

- market data loading
- strategy computation
- netting
- pre-trade validation
- decision recording
- TCA simulation

## Acceptance requirement

A unit test must fail if a broker submission is reachable during replay.