ChatGPT

Deep Implementation Extraction and Codex 5.3 Build for dmorazzini23/ai-trading-bot

Executive summary

Enabled connectors requested and used: **GitHub** [1] (only). The analysis below is constrained to the single repository `dmorazzini23/ai-trading-bot` as instructed, plus a small set of high-quality public sources used only to validate time-sensitive or standards-related claims (rate limits, interpreter behavior, systemd semantics, and institutional control frameworks). [2]

Across the three user-provided PDFs, the highest-impact implementation work clusters into six themes:

- **OMS correctness and restart safety:** terminal-status taxonomy drift (notably "FAILED"/"EXPIRED") can keep intents permanently "open," creating reconciliation churn and post-restart misbehavior.
- **Determinism as a release gate:** eliminate nondeterministic primitives (`hash()`, unseeded global RNG, time-derived IDs) from any "deterministic replay / parity" path; CI should exercise a seed matrix.
- **Execution liveness diagnosis:** unify "order attempt outcome" logging so every intent produces exactly one of {submitted, skipped-with-reason, failed-with-exception}; surface broker lockouts and rate limiting explicitly.
- **Security fail-closed posture:** encryption and secret handling must not silently downgrade; rotate leaked credentials and enforce secret scanning gates.
- **Config governance:** converge duplicated environment keys into one canonical contract with conflict detection, a redacted startup summary, and configuration snapshot hashes embedded in decision records.
- **Execution realism & TCA calibration loop:** treat implementation shortfall / TCA as a core KPI and wire a feedback loop from realized fills → cost floor calibration → trade gating when stale.

These themes are strongly aligned with industry guidance emphasizing pre-trade risk controls, kill switches, operational monitoring, post-trade review, and disciplined change/testing processes for automated trading systems. [3]

PDF-derived implementation requirements

The items below are extracted from the attached PDFs only. Each includes: (a) exact location, (b) a short excerpt (≤25 words), (c) priority with rationale, (d) dependencies, and (e) effort estimate (S/M/L/XL).
Effort scale assumptions (explicit, for consistency): **S** ≤0.5 day, **M** 0.5–2 days, **L** 2–5 days, **XL** >5 days (single developer familiar with the repo and CI).

OMS terminal status taxonomy and restart safety

A-OMS-01 — Add missing terminal statuses (at least FAILED and EXPIRED) and align across OMS + migration + execution reconciliation
- **Location:** *Repo-wide Audit of dmorazzini23_ai-trading-bot.pdf*, p. 11, "Migration correctness for terminal statuses"
- **Excerpt:** "*assert 'FAILED' in _TERMINAL_STATUSES*"
- **Priority: High** — directly prevents "open intents" poison and infinite reconciliation loops; the audit labels this high-risk and correctness-critical.
- **Dependencies:** A-OMS-02, A-OMS-03 (tests); repo code alignment in both store and migration.
- **Effort: S–M** (0.5–1.5 days).

A-OMS-02 — Add broker status mapping tests for statuses like expired/replaced/stopped → terminal or explicit handling
- **Location:** *Repo-wide Audit…*, p. 5, "Idempotency + 'exactly once'" / mapping notes
- **Excerpt:** "*Add 'broker status mapping' tests: expired, replaced, stopped…*"
- **Priority: High** — prevents silent drift between broker truth and local truth; reduces stuck "SUBMITTING/SUBMITTED forever" paths.
- **Dependencies:** A-OMS-01; execution adapter mapping points.
- **Effort: M** (1–2 days).

A-OMS-03 — Crash/restart integration test: submit intent → crash point → restart → no duplicate submit; intent goes terminal
- **Location:** *Repo-wide Audit…*, p. 5, "Idempotency + 'exactly once'"
- **Excerpt:** "*Crash/restart integration test… ensure no duplicate submit…*"
- **Priority: High** — directly targets restart-induced duplication and "stuck intents."
- **Dependencies:** A-OMS-01; deterministic store setup; test harness utilities.
- **Effort: M** (1–2 days).

A-OMS-04 — Alerting rules: open intents by status; alert if non-terminal beyond threshold or "FAILED but still open"
- **Location:** *Repo-wide Audit…*, p. 13, "Institutional-grade alert rules to add"
- **Excerpt:** "*alert if any intent remains non-terminal beyond threshold…*"
- **Priority: Medium** — observability; reduces MTTR for restart anomalies.
- **Dependencies:** unified metrics/log events (B-DIAG-01, B-DIAG-02).
- **Effort: M** (0.5–2 days depending on metrics backend).

Determinism and replay parity as a release gate

A-DET-01 — Remove reliance on builtin hash() in deterministic simulation/replay paths; use stable hashes (sha256) + explicit seed injection
- **Location:** *Repo-wide Audit…*, p. 11, "Remediation roadmap…"
- **Excerpt:** "*Remove any reliance on hash()… replace with stable hashes (sha256)…*"
- **Priority: High** — Python hash randomization is on by default unless fixed at interpreter start; determinism cannot be repaired by setting env vars after startup. [4]
- **Dependencies:** A-DET-02, A-DET-03 (tests); A-CONFIG-02 (seed matrix).
- **Effort: M** (1–2 days) if scoped; **L** if repo has many scattered uses.

A-DET-02 — Add "no accidental hash()" regression test scanning engine source
- **Location:** *Repo-wide Audit…*, p. 8, "Determinism and 'no accidental hash()'"
- **Excerpt:** "*Avoid builtin hash() in deterministic simulation paths*"
- **Priority: High** — enforces the policy mechanically in CI.
- **Dependencies:** A-DET-01.
- **Effort: S** (0.25–0.5 day).

A-DET-03 — Ensure deterministic simulated broker reproducibility test
- **Location:** *Repo-wide Audit…*, p. 8, "test_simulated_broker_is_reproducible"
- **Excerpt:** "*Why: hard requirement for replay parity.*"
- **Priority: Medium** — targeted; likely already passes but should remain pinned.
- **Dependencies:** none (but complements A-DET-01).
- **Effort: S**.

A-DET-04 — Block unseeded global RNG in simulator; require explicit seed or injected Random, or gate legacy simulator behind explicit opt-in
- **Location:** *Repo-wide Audit…*, p. 9, "Simulation path hygiene"
- **Excerpt:** "*execution/simulator.py uses global random; enforce explicit seeding…*"
- **Priority: High** — unseeded randomness undermines deterministic replay and makes CI flaky.
- **Dependencies:** A-SIM-01 (simulation consolidation).
- **Effort: M** (0.5–2 days).

Execution liveness diagnosis (post-restart "submits then stops")

B-DIAG-01 — Normalize order submission outcomes: every intent emits exactly one of {ORDER_SUBMITTED, ORDER_SUBMIT_SKIPPED(reason), ORDER_SUBMIT_FAILED(exception)}
- **Location:** *Diagnosing Post-Restart…pdf*, p. 7–8, "Add a single 'order attempt outcome' log…"
- **Excerpt:** "*ensure every order intent results in exactly one of…*"
- **Priority: High** — without this, "silent skip" guards are indistinguishable from a broken submit path.
- **Dependencies:** B-DIAG-02 (cycle summary), B-DIAG-03 (rate limiting).
- **Effort: M** (1–2 days).

B-DIAG-02 — Add _skip_submit helper and use it for early returns (doesn't change trading behavior; only visibility)
- **Location:** *Diagnosing Post-Restart…pdf*, p. 8, patch sketch
- **Excerpt:** "*add a small helper… use it for early returns*"
- **Priority: High** — fastest path to root-cause clarity.
- **Dependencies:** none.
- **Effort: S–M** (0.5–1.5 days depending on how many early returns).

B-DIAG-03 — Surface broker lockout state and cooldown in every cycle summary
- **Location:** *Diagnosing Post-Restart…pdf*, p. 9, "Surface broker lockout state…"
- **Excerpt:** "*broker_locked_until… broker_lock_reason… counts by reason*"
- **Priority: High** — directly targets "works after restart then stops" if lockout is time-dependent.
- **Dependencies:** B-DIAG-01 reason codes; access to loop driver summary log.
- **Effort: S–M**.

B-DIAG-04 — Make rate limiting explicit and cross-process; honor Retry-After on 429; reduce unnecessary polling
- **Location:** *Diagnosing Post-Restart…pdf*, p. 9, "Make rate limiting explicit…"
- **Excerpt:** "*Implement a shared… rate limiter… honors Retry-After…*"
- **Priority: High** — Alpaca trading APIs are throttled (commonly 200 requests/min/account); 429 is time-dependent and can appear only after steady-state load. [5]
- **Dependencies:** shared storage/lock mechanism (file lock, Redis, etc.); execution submit wrapper entry point.
- **Effort: L** (2–5 days) if truly cross-process; **M** if single-process only.

B-DIAG-05 — Harden systemd environment/logging correctness (validate EnvironmentFile, StandardOutput=journal, restart policy)
- **Location:** *Diagnosing Post-Restart…pdf*, p. 6 & p. 10
- **Excerpt:** "*EnvironmentFile= semantics are defined in systemd.exec*"
- **Priority: Medium** — operational hardening; prevents "credentials present in one-shot, missing in daemon" drift.
- **Dependencies:** packaging docs / systemd unit file updates.
- **Effort: S**.

Security fail-closed posture and secret hygiene

A-SEC-01 — Convert crypto fallbacks to fail-closed in production: require cryptography availability + MASTER_ENCRYPTION_KEY (or equivalent)
- **Location:** *Repo-wide Audit…*, p. 11, "Convert security fallbacks…"
- **Excerpt:** "*require cryptography availability and require MASTER_ENCRYPTION_KEY…*"
- **Priority: High** — avoids false sense of encryption; prevents silent downgrade.
- **Dependencies:** config notion of "production mode" (A-CONFIG-01); tests (A-SEC-02).
- **Effort: S–M**.

A-SEC-02 — Add unit test: crypto missing / missing master key fails closed (in production mode)
- **Location:** *Repo-wide Audit…*, p. 9–10, "Security fail-closed checks"
- **Excerpt:** "*crypto fallbacks must not silently become 'no encryption'*"
- **Priority: High** — enforces fail-closed semantics.
- **Dependencies:** A-SEC-01.
- **Effort: S**.

A-SEC-03 — Immediate credential rotation & secret store migration; enforce secret scanning gates
- **Location:** *Repo-wide Audit…*, p. 13, "Security actions you should take now" and *Deep review…*, p. 5, "Secret handling needs immediate tightening"
- **Excerpt:** "*Immediately rotate the exposed DB credential… Treat as compromised.*"
- **Priority: High** — incident response; reduces real compromise risk.
- **Dependencies:** org ops; CI gating (A-CI-01).
- **Effort: S** for rotation (but requires operator access), **M** for secret-store migration.

Config governance and “single source of truth”

C-CONFIG-01 — Enforce a typed config contract with conflict detection; converge AI_TRADING_* vs legacy variants
- **Location:** *Deep review…*, p. 4, “too many… sources of truth”
- **Excerpt:** “*typed config schema with validation (fail fast if conflicting keys are set)*”
- **Priority: High** — prevents two operators believing they run the same configuration while actually running different regimes.
- **Dependencies:** existing config schema; decision logging integration (C-CONFIG-03).
- **Effort: M**.

C-CONFIG-02 — Print a redacted startup “effective config” summary and include a config snapshot hash in decision records
- **Location:** *Deep review…*, p. 5, “Add explicit config contract requirements”
- **Excerpt:** “*printed startup effective config summary (redacted)… config snapshot hash…*”
- **Priority: High** — makes every trade reconstructible.
- **Dependencies:** config snapshot structure and hashing function; decision record schema.
- **Effort: M–L** depending on how decision records are stored.

C-CONFIG-03 — Promote determinism to a release blocker: replay determinism across N seeds and N symbols; deterministic walk-forward partitions (purge/embargo)
- **Location:** *Deep review…*, p. 5, “Promote determinism…”
- **Excerpt:** “*replay determinism across N seeds + N symbols… deterministic walk-forward partitions*”
- **Priority: High** — aligns research integrity with production controls (reduces leakage/false positives).
- **Dependencies:** test harness; offline replay tooling; data split utilities.
- **Effort: L**.

Simulation, cost model calibration, and TCA feedback loop

A-SIM-01 — Consolidate simulation modules: deterministic simulator is the only option for deterministic replay; nondeterministic simulator requires explicit opt-in
- **Location:** *Repo-wide Audit…*, p. 12, “Consolidate simulation modules”
- **Excerpt:** “*ensure only the deterministic simulator is used for deterministic replay*”
- **Priority: Medium** — reduces confusion and replay drift; needed once determinism becomes a gate.
- **Dependencies:** A-DET-04, A-DET-01.
- **Effort: M**.

C-TCA-01 — Require a calibration loop: TCA records → parameter updates → next-day cost floors; monitor drift; block trading if stale/out-of-bounds
- **Location:** *Deep review…*, p. 5–6, “Require a calibration loop…”
- **Excerpt:** “*block trading if calibration is stale or inconsistent…*”
- **Priority: High** — directly improves execution quality and risk containment; cost model becomes a control, not a feature.
- **Dependencies:** TCA record formats; persistence; scheduler hook; config gating.
- **Effort: XL** if fully featured; **L** if minimal viable loop (daily calibration + staleness checks).

A-TCA-02 — Add unit test: cost model output bounded (tighten to frozen max once finalized)
- **Location:** *Repo-wide Audit…*, p. 10, “Cost model calibration and bounds invariants”
- **Excerpt:** “*Why: “bounded outputs” requirement.*”
- **Priority: Medium** — guardrail regression test.
- **Dependencies:** cost model API stable.
- **Effort: S**.

CI / supply-chain gates / institutional controls

A-CI-01 — Add SAST/SCA/SBOM/Scorecard to CI; keep secret scanning enforced
- **Location:** *Repo-wide Audit…*, p. 5–6, CI mapping table
- **Excerpt:** “*Add SAST/SCA/SBOM/Scorecard…*”
- **Priority: Medium** — reduces supply-chain and dependency risk; increases “institutional” posture.
- **Dependencies:** GitHub workflow additions; baseline SCA tooling selection.
- **Effort: M**.

A-CI-02 — Add deterministic seed matrix in CI (multiple seeds; multiple symbols)
- **Location:** *Repo-wide Audit…*, p. 5–6 and *Deep review…*, p. 5
- **Excerpt:** “*add deterministic seed matrix*”
- **Priority: High** — catches nondeterminism regressions early.
- **Dependencies:** determinism harness; stable replay test runner.
- **Effort: M–L**.

C-DOC-01 — Add explicit “non-goal” disclaimer: profitability is not guaranteed
- **Location:** *Deep review…*, p. 6
- **Excerpt:** “*Include an explicit non-goal section… profitability is not guaranteed*”
- **Priority: Low** (but important for expectation setting and safety framing).
- **Dependencies:** documentation updates.
- **Effort: S**.

Repo inspection and mapping to code

This section maps each PDF-driven item to concrete repo locations, and marks whether it is implemented, partially implemented, or missing—based strictly on inspection of `dmorazzini23/ai-trading-bot` files fetched via the GitHub connector.

OMS terminal statuses and reconciliation loops

Observed code (current): - `ai_trading/oms/intent_store.py` defines terminal statuses as:

```
_TERMINAL_STATUSES: frozenset[str] = frozenset(
    {"FILLED", "CANCELED", "CANCELLED", "REJECTED", "CLOSED"}
)
```

- `scripts/migrate_oms_intent_store.py` duplicates the same set:

```
_TERMINAL_STATUSES: frozenset[str] = frozenset(
    {"FILLED", "CANCELED", "CANCELLED", "REJECTED", "CLOSED"}
)
```

- `ai_trading/execution/engine.py` can close intents with `final_status="FAILED"` during reconciliation when a broker order can't be found:

```
self._intent_store.close_intent(
    intent.intent_id,
    final_status="FAILED",
    last_error="reconcile_missing_broker_order",
)
```

Mapping results: - **A-OMS-01 → Missing / inconsistent.** The execution engine can write `FAILED`, but the OMS store does not consider it terminal, so `get_open_intents()` will keep returning it.
Concrete code changes needed: - Create a single source of truth for terminal statuses (e.g., `ai_trading/oms/statuses.py`) and import it from both `intent_store.py` and `migrate_oms_intent_store.py`. - Expand terminal set to include at least: `FAILED`, `EXPIRED` (and consider `DONE_FOR_DAY`, `REPLACED`, `STOPPED` only if the broker/execution layer can emit them). - Add a safety invariant: **any status written by** `close_intent()` **must be terminal by definition**, else raise in tests.

Suggested tests (repo paths to create): - `tests/unit/test_migration_terminal_statuses.py` (as PDF suggests) asserting `"FAILED" in _TERMINAL_STATUSES`. - `tests/unit/test_intent_store_terminal_statuses.py` ensuring `get_open_intents()` excludes terminal statuses, including FAILED/EXPIRED.

Determinism leaks: builtin hash(), unseeded randomness, time-derived IDs

Observed code (current): - `ai_trading/execution/engine.py` uses builtin `hash()` in `_simulate_market_execution()`:

```
predicted_fill = base_price * (1 + (hash(order.id) % 100 - 50) / 10000)
...
fill_price = base_price * (1 + (hash(order.id) % 100 - 50) / 10000)
```

- `ai_trading/execution/simulator.py` uses global `random.*` directly (no instance RNG injection).
- `ai_trading/utils/ids.py` defines `stable_client_order_id()` but includes a random suffix (`secrets.token_hex(4)`), so it is **not** fully deterministic.
- `ai_trading/execution/simulated_broker.py` is deterministic given its `seed` because it uses `random.Random(seed)` and internal counters.

Mapping results: - **A-DET-01 → Missing.** builtin `hash()` remains in a simulation path; Python's hash randomization behavior requires `PYTHONHASHSEED` at interpreter start for repeatable hashes. [4]
Concrete code changes needed: - Introduce `ai_trading/utils/determinism.py` with a stable hash helper: - `stable_hash32(*parts: str) -> int` using `hashlib.sha256`. - `rng_from_parts(seed: int, *parts) -> random.Random` to produce deterministic jitter without depending on `hash()`. - Replace `hash(order.id)` usage with a deterministic RNG derived from `(TradingConfig.seed, order.id, symbol, side)` or similar stable tuple. - Optional but strongly recommended: add a `DETERMINISTIC_REPLAY` mode that forbids any `secrets.*`, time-derived IDs, and global RNG use in the execution simulation.

- **A-DET-04 / A-SIM-01 → Partially implemented.** A deterministic simulated broker exists, but there is also a nondeterministic simulator module that can drift unless gated or refactored.

Suggested tests: - `tests/unit/test_no_builtin_hash_in_deterministic_paths.py` (PDF-proposed) that fails if the execution engine source contains `hash(`. - `tests/unit/test_no_unseeded_global_random_simulator_usage.py` that either: - enforces injection of `random.Random(seed)` into simulator, or - ensures legacy simulator is not invoked unless an explicit `AI_TRADING_ALLOW_NONDETERMINISTIC_SIM=1` flag is set. - `tests/unit/test_simulated_broker_reproducible.py` as PDF proposes.

Execution liveness: post-restart "skips" vs "failures"

Observed code (current): - `ai_trading/execution/live_trading.py` contains many early-return guards (as described in PDF B), but the repo currently does **not** enforce a single "normalized skip reason" log event everywhere (based on the PDF's diagnosis and patch sketch). - `packaging/systemd/ai-trading.service` already sets `StandardOutput=journal` and `StandardError=journal`, and uses `Restart=always`.

Mapping results: - **B-DIAG-01 / B-DIAG-02 → Missing.** There is no single helper (`_skip_submit`) used uniformly for early returns, so diagnosing time-dependent skip states is harder than necessary. **Concrete code changes needed:** - Add `_skip_submit(reason: str, symbol, side, qty, extra)` to `ai_trading/execution/live_trading.py` and use it for every "skip" early return. - Add a single `ORDER_SUBMIT_ATTEMPT` log emitted right before the actual broker call, including a correlation id (intent id / client_order_id), plus provider state and pacing counters.

- **B-DIAG-03 → Partially implemented / unclear.** The execution engine has lifecycle hooks (`start_cycle`, `end_cycle`) and the config schema supports end-of-cycle summary toggles, but the PDF's explicit "broker_locked_until + counts by reason" cycle summary is not confirmed as present.

- **B-DIAG-04 → Partially implemented.**

- The repo includes HTTP retry + host concurrency caps in `ai_trading/utils/http.py`, and `alpaca-trade-api` itself can retry on 429 depending on configuration. [6]
- What appears missing (per PDF guidance) is **cross-process** rate limiting and explicit surfacing of `Retry-After` handling for submission endpoints.

Suggested tests and artifacts: - `tests/unit/test_order_submit_outcome_logging.py`: - simulate each major skip reason path and assert a single `ORDER_SUBMIT_SKIPPED` event is logged with the correct reason code. - `tests/unit/test_broker_lockout_is_visible_in_cycle_summary.py`: - force broker lockout; assert cycle summary exposes lock reason and TTL.

Security fail-closed behavior and secret scanning

Observed code (current): - `ai_trading/security.py` contains fallback behavior that can effectively become "no encryption" if crypto init fails; it also auto-generates a master key if `MASTER_ENCRYPTION_KEY` is not set:

```
env_key = os.getenv('MASTER_ENCRYPTION_KEY')
if env_key:
    return env_key
key = secrets.token_urlsafe(32)
self.logger.warning('Generated new master encryption key ...')
return key
```

- CI workflow `.github/workflows/ci.yml` runs `make secret-scan` and `make institutional-gates`, which is strong already.

Mapping results: - **A-SEC-01 / A-SEC-02 → Partially implemented but not fail-closed.**
Required changes: - Define "production mode" (likely `EXECUTION_MODE=live` or `APP_ENV=prod`) and then: - hard-fail if cryptography is unavailable or master key is missing, - forbid auto-generating master keys in production, - ensure failures do not return plaintext silently. - **A-SEC-03 → Partially implemented (CI) + operational work required.** The CI already includes secret scanning, but PDFs require enforced rotation and secret-store migration when leaks occur.

CI hardening and supply-chain controls

Observed code (current): - `.github/workflows/ci.yml` exists and runs lint, mypy, and several "institutional gate" targets. - There is no evidence in the fetched files of: - CodeQL workflow, - pip-audit/OSV scanning workflow, - SBOM generation workflow, - OSSF Scorecard workflow.

Mapping results: - **A-CI-01 → Missing.** Add dedicated or integrated workflows for SAST/SCA/SBOM/Scorecard.

Config governance and snapshot hashes

Observed code (current): - `ai_trading/config/runtime.py` already defines a typed configuration schema (`CONFIG_SPECS`) with deprecated aliases, plus a sanitized snapshot method:

```
def snapshot_sanitized(self) -> dict[str, Any]:
    data = {"risk": {...}, "data": {...}, "execution": {...}, "auth":
```

```
{"alpaca_api_key": "***", ...}}
    return data
```

- However, conflict detection (“fail fast if conflicting keys are set”) is not clearly present; deprecated keys trigger warnings but may still allow ambiguous precedence.

Mapping results: - **C-CONFIG-01** → **Partially implemented.** Add explicit conflict detection and deterministic hashing of a sanitized config snapshot. - **C-CONFIG-02** → **Partially implemented.** A snapshot exists, but embedding a snapshot hash into decision records and printing a startup redacted summary is not confirmed.

Actionable implementation plan with sprints, acceptance criteria, and risk controls

Sprint structure and milestones

Sprint A

Goal: eliminate the highest-risk correctness failures and make diagnosis unambiguous.

Scope (must land together): - A-OMS-01, A-OMS-02 (minimum mapping), A-OMS-03 - A-DET-01 + A-DET-02 - B-DIAG-01 + B-DIAG-02 + B-DIAG-03 (at least log lockout state) - A-SEC-01 + A-SEC-02

Acceptance criteria (Sprint A): - No terminal status written by execution/OMS can appear in `get_open_intents()`. - “FAILED” and “EXPIRED” (at minimum) are terminal everywhere (OMS + migration + any reconciliation logic). - CI fails if builtin `hash()` exists in deterministic execution simulation paths. - Every order intent emits exactly one outcome event: `ORDER_SUBMITTED` / `ORDER_SUBMIT_SKIPPED` / `ORDER_SUBMIT_FAILED`. - In “production mode,” encryption cannot silently degrade (missing master key == hard error).

Key risks & mitigations: - *Risk:* changing terminal sets could break migrations or existing tests.
Mitigation: centralize status constants and add explicit regression tests. - *Risk:* log normalization touches many early returns.
Mitigation: mechanical refactor (helper) + tests asserting **exactly once** logging.

Sprint B

Goal: institutionalize determinism and config correctness; reduce “operator ambiguity.”

Scope: - C-CONFIG-01, C-CONFIG-02 - A-CI-02 (seed matrix) - A-SIM-01 + A-DET-04 (simulation gating) - A-CI-01 (security/supply chain workflows baseline)

Acceptance criteria (Sprint B): - Startup prints a redacted config summary + a config snapshot hash. - Decision records include `config_snapshot_hash`. - CI runs deterministic replay across at least 3 seeds and a small symbol set. - Legacy nondeterministic simulator is blocked unless explicitly enabled.

Key risks & mitigations: - *Risk:* config conflict detection might break existing deployments using deprecated keys.
Mitigation: staged rollout: warn → error in live mode only → later error everywhere.

Sprint C

Goal: execution quality loop: rate limiting, TCA calibration, and trade gating based on stale/out-of-bound cost models.

Scope: - B-DIAG-04 (rate limiting explicit; ideally cross-process) - C-TCA-01 + A-TCA-02 - A-OMS-04 (alerts/metrics)

Acceptance criteria (Sprint C): - 429 handling is explicit; `Retry-After` is honored or conservative backoff is used. [5] - Cost model calibration pipeline exists (even minimal daily batch) and trading blocks when calibration stale/out-of-range. - Metrics/logs support alert rules listed in PDFs (open intents stuck, broker error budgets, idempotency collision spikes).

Key risks & mitigations: - *Risk:* truly cross-process rate limiting adds complexity.
Mitigation: deliver a single-process limiter first, then swap backend to file-lock or Redis.

Table mapping PDF items to repo tasks, priority, effort, and sprint

PDF Item ID	Repo task (deliverable)	Priority	Effort	Sprint
A-OMS-01	Centralize terminal statuses; add FAILED/EXPIRED; align migration + store	High	M	A
A-OMS-02	Broker status mapping tests + explicit handling	High	M	A
A-OMS-03	Restart/idempotency integration test	High	M	A
A-DET-01	Replace `hash()` in deterministic paths with sha256-seeded RNG	High	M	A
A-DET-02	Add test forbidding builtin `hash()` in deterministic paths	High	S	A
A-DET-04	Refactor/gate nondeterministic simulator; require seed injection	High	M	B
B-DIAG-01	Normalize order attempt outcomes in logs	High	M	A
B-DIAG-02	`_skip_submit()` helper used for all early-return skip guards	High	M	A
B-DIAG-03	Cycle summary includes broker lockout + reason + counts	High	S–M	A
B-DIAG-04	Rate limiting explicit; honor Retry-After/backoff; reduce polling	High	L	C

PDF Item ID	Repo task (deliverable)	Priority	Effort	Sprint
A-SEC-01	Fail-closed crypto + required master key in production mode	High	S–M	A
A-SEC-02	Unit tests for fail-closed crypto behavior	High	S	A
A-CI-01	Add CodeQL / pip-audit(or OSV) / SBOM / Scorecard workflows	Medium	M	B
A-CI-02	Deterministic seed matrix in CI	High	M–L	B
C-CONFIG-01	Config conflict detection + canonicalization	High	M	B
C-CONFIG-02	Startup redacted config summary + config snapshot hash in decisions	High	M–L	B
C-TCA-01	Calibration loop + staleness gating	High	L–XL	C
A-TCA-02	Tight bounded-cost test to match frozen model bounds	Medium	S	C
A-OMS-04	Alert/metric emitters for OMS and broker health	Medium	M	C
C-DOC-01	Add explicit non-goal disclaimer (profitability not guaranteed)	Low	S	B

Mermaid diagrams for module relationships, data flow, and sprint timeline

Module relationships and control surfaces

```
flowchart LR
  subgraph Config
    CFG[TradingConfig schema + conflict detection]
    SNAP[Redacted config snapshot + hash]
  end

  subgraph Decision
    FEAT[Features + signals]
    DEC[Decision record JSONL]
  end

  subgraph Execution
    GATE[Pretrade gates + skip reasons]
    OMS[Durable IntentStore + idempotency]
    ENG[ExecutionEngine]
    BRK[Broker adapter]
  end
```

```
  subgraph Quality
    TCA[TCA / slippage log]
    COST[Bounded cost model + calibration]
  end

  CFG --> GATE
  CFG --> ENG
  CFG --> COST
  FEAT --> DEC --> SNAP
  SNAP --> OMS
  GATE --> OMS --> ENG --> BRK
  BRK --> TCA --> COST --> GATE
```

Order submission outcome semantics (diagnostic visibility)

```
flowchart TD
  A[Order intent created] --> B{Pretrade & policy gates}
  B -->|pass| C[ORDER_SUBMIT_ATTEMPT]
  B -->|skip| D[ORDER_SUBMIT_SKIPPED<br/>reason=...]
  C --> E{Broker call}
  E -->|accepted| F[ORDER_SUBMITTED<br/>broker_order_id, client_order_id]
  E -->|exception/429/401| G[ORDER_SUBMIT_FAILED<br/>status_code, error_code]
  F --> H[OMS mark_submitted + track fills]
  G --> I[OMS record_submit_error + lockout TTL if auth]
```

Sprint timeline

```
gantt
  title Implementation sprints (2-week cadence)
  dateFormat  YYYY-MM-DD
  axisFormat  %b %d

  section Sprint A
  OMS terminal statuses + tests           :a1, 2026-02-24, 14d
  Determinism: remove hash() + tests      :a2, 2026-02-24, 14d
  Unified submit outcomes + lockout logs  :a3, 2026-02-24, 14d
  Fail-closed crypto + tests             :a4, 2026-02-24, 14d

  section Sprint B
  Config conflict detection + snapshot hash :b1, 2026-03-10, 14d
  CI seed matrix + determinism gates        :b2, 2026-03-10, 14d
  Simulation consolidation/gating           :b3, 2026-03-10, 14d
  CI security workflows (SAST/SBOM/etc)     :b4, 2026-03-10, 14d
```

```
  section Sprint C
  Rate limiting explicit + Retry-After      :c1, 2026-03-24, 14d
  TCA calibration loop + staleness gating   :c2, 2026-03-24, 14d
  OMS/broker health alerts/metrics          :c3, 2026-03-24, 14d
```

Codex 5.3 prompt tailored to dmorazzini23/ai-trading-bot

```
You are Codex 5.3. Implement missing/partial requirements from three user-
provided PDFs for repo: dmorazzini23/ai-trading-bot.

Hard constraints:
- Work ONLY inside this repo.
- Do not invent requirements not supported by PDFs; if ambiguous, mark as
unspecified and add minimal safe assumptions.
- Preserve existing behavior unless explicitly changing for safety/correctness.
- Add tests for every change (unit + at least one integration test where
required).
- Maintain determinism: no builtin hash() for deterministic simulation/replay;
no unseeded global RNG in deterministic mode.
- Fail closed in production mode for security invariants.

Repo context summary (from inspection):
- Python 3.12 project, primary package: ai_trading/
- Key modules:
  - ai_trading/execution/engine.py: ExecutionEngine + OrderManager +
reconcile_open_intents()
  - ai_trading/oms/intent_store.py: SQLAlchemy-based durable intent store
  - scripts/migrate_oms_intent_store.py: legacy migration script with duplicated
status constants
  - ai_trading/execution/simulated_broker.py: deterministic simulated broker
(seeded Random)
  - ai_trading/execution/simulator.py: nondeterministic simulator using global
random (must be gated/refactored)
  - ai_trading/security.py: SecureConfig and encryption logic (must be fail-
closed in production)
  - ai_trading/config/runtime.py: typed TradingConfig schema + deprecated env
aliases + snapshot_sanitized()
  - packaging/systemd/ai-trading.service: systemd unit baseline (journal
logging, Restart=always)
- CI: .github/workflows/ci.yml runs secret-scan, institutional-gates, ruff,
mypy, pytest.

Primary implementation tasks (do in order, respecting dependencies):

1) OMS terminal status taxonomy: prevent infinite reconciliation loops
   - Create ai_trading/oms/statuses.py defining:
```

```
     - TERMINAL_INTENT_STATUSES (frozenset[str]) including at least:
       FILLED, CANCELED, CANCELLED, REJECTED, CLOSED, FAILED, EXPIRED
     - Optional: document any additional statuses only if execution layer can
emit them.
   - Update ai_trading/oms/intent_store.py to import this constant instead of
its local _TERMINAL_STATUSES.
   - Update scripts/migrate_oms_intent_store.py to import the shared constant,
or keep in sync by importing.
   - Ensure any status written by close_intent() is terminal; add a unit test
enforcing this invariant.
   Tests to add:
   - tests/unit/test_intent_store_terminal_statuses.py:
     - create intents in various statuses and assert get_open_intents() excludes
terminals.
   - tests/unit/test_migration_terminal_statuses.py:
     - assert "FAILED" and "EXPIRED" are terminal in migration constant/source.

2) Reconciliation mapping tests for broker statuses (expired/replaced/stopped)
   - Identify where broker order statuses are normalized/mapped (likely in
execution adapters or engine reconciliation).
   - Implement explicit mapping so that when broker reports an order is expired/
replaced/stopped, the corresponding intent is closed terminally and not retried
indefinitely.
   Tests:
   - tests/unit/test_broker_status_mapping_terminal.py:
     - simulate broker status payloads and assert intents become terminal and
are no longer considered open.

3) Determinism: remove builtin hash() from deterministic simulation paths
   - Add ai_trading/utils/determinism.py with:
     - def stable_sha256_int(*parts: str, bits: int = 32) -> int
     - def rng_from_parts(seed: int, *parts: str) -> random.Random
   - In ai_trading/execution/engine.py, replace any use of hash(order.id) (and
similar) with deterministic RNG derived from:
     - TradingConfig.seed (or get_env("SEED") via config helper)
     - stable inputs: order.id, symbol, side, maybe client_order_id
   - Ensure simulation jitter is stable across processes for same inputs.
   Tests:
   - tests/unit/test_no_builtin_hash_in_deterministic_paths.py:
     - use inspect.getsource(ai_trading.execution.engine) and assert "hash(" not
present.
   - tests/unit/test_simulated_broker_reproducible.py:
     - ensure SimulatedBroker(seed=42) produces identical submission results for
identical inputs.

4) Simulation consolidation / no unseeded global RNG
   - Decide a policy:
     - Deterministic replay path MUST use deterministic simulator/broker only.
```

```
     - Nondeterministic simulator (execution/simulator.py) must require explicit
opt-in:
       env AI_TRADING_ALLOW_NONDETERMINISTIC_SIM=1 (default 0).
   - Refactor execution/simulator.py:
     - inject RNG: FillSimulator(rng: random.Random) or FillSimulator(seed: int)
     - remove direct calls to global random.* OR hard-fail if used without
explicit opt-in.
   Tests:
   - tests/unit/test_no_unseeded_global_random_simulator_usage.py:
     - assert deterministic mode fails if simulator uses global random or lacks
seed injection.
   - Add/adjust any existing replay tests to ensure deterministic paths never
import/use nondeterministic simulator unintentionally.

5) Exactly-once semantics across restart (integration)
   - Implement tests/integration/test_exactly_once_submit_across_restart.py:
     - create sqlite-backed IntentStore, create intent, mark_submitted,
"restart" by re-instantiating store, confirm broker_order_id persists.
   - Add a second integration test for reconcile_open_intents:
     - create a SUBMITTED intent that is missing from broker open-orders;
reconcile should mark FAILED and then it must be terminal (not returned as
open).

6) Unified order submission outcome logging (diagnostics)
   - In ai_trading/execution/live_trading.py:
     - add helper:
       def _skip_submit(reason: str, *, symbol: str|None=None, side: str|
None=None, qty: int|None=None, extra: dict|None=None) -> None
       It must emit one structured log event: ORDER_SUBMIT_SKIPPED with reason
and context.
   - Refactor early-return guards to call _skip_submit().
   - Ensure exactly one outcome log per intent:
     - ORDER_SUBMIT_ATTEMPT (just before broker submit)
     - ORDER_SUBMITTED (on success; include broker_order_id, client_order_id if
present)
     - ORDER_SUBMIT_FAILED (on exceptions; include status_code/error_code if
available)
   Tests:
   - tests/unit/test_order_submit_outcome_logging.py:
     - monkeypatch logger/caplog and simulate representative skip paths.

7) Cycle summary: expose broker lockout + reason + counts by skip reason
   - Add per-cycle aggregation (in ai_trading/main.py loop end or
ExecutionEngine.end_cycle()):
     - broker_locked_until (monotonic deadline)
     - broker_lock_reason
     - counts: submitted / skipped grouped by reason
   Tests:
```

```
   - tests/unit/test_broker_lockout_visible_in_cycle_summary.py:
     - force lockout state and assert summary log includes it.

8) Security fail-closed behavior in production mode
   - Define production mode predicate (minimal safe):
     - EXECUTION_MODE == "live" OR APP_ENV in {"prod","production"}.
   - In ai_trading/security.py:
     - Do not auto-generate MASTER_ENCRYPTION_KEY in production mode.
     - If cryptography init fails in production mode, raise RuntimeError.
     - encrypt_value/decrypt_value must not silently return plaintext/ciphertext
on failure in production; raise.
   Tests:
   - tests/unit/test_security_crypto_required_in_production.py:
     - set production mode env, unset MASTER_ENCRYPTION_KEY, assert
initialization fails.
     - optionally simulate crypto unavailable (monkeypatch flag) and assert
fail-closed.

9) CI hardening: SAST/SCA/SBOM/Scorecard + determinism seed matrix
   - Add workflows:
     - .github/workflows/codeql.yml (Python CodeQL)
     - .github/workflows/dependency-audit.yml (pip-audit or OSV scanner)
     - .github/workflows/sbom.yml (generate SBOM artifact)
     - .github/workflows/scorecard.yml (OSSM Scorecard)
   - Add CI matrix for determinism:
     - run selected deterministic tests under SEED={1,42,1337}
   Validate:
   - CI passes on main branch; workflows are minimal and do not require secrets.

10) Config governance: conflict detection + snapshot hash in decision records
   - In ai_trading/config/runtime.py:
     - Implement conflict detection: if both canonical env and deprecated alias
are set and differ, fail fast (at least in production mode).
   - Use TradingConfig.snapshot_sanitized() to produce a deterministic JSON
serialization and compute a sha256 hash.
   - Emit startup log:
     - CONFIG_EFFECTIVE_SUMMARY (redacted) with config_snapshot_hash
   - Ensure decision records include config_snapshot_hash (find decision record
writer; if none, add minimal field to JSONL writer used for decisions).
   Tests:
   - tests/unit/test_config_conflict_detection.py
   - tests/unit/test_config_snapshot_hash_stable.py

11) Rate limiting explicit (single-process MVP, then cross-process optional)
   - Implement a token-bucket limiter for broker trading endpoints:
     - default: 200 requests/minute/account (configurable), respecting Retry-
After when present.
   - Optional cross-process backend:
```

```
     - file-based state + portalocker OR Redis if configured (placeholder).
   Tests:
   - tests/unit/test_rate_limiter_token_bucket.py

Deliverables: commit code + tests. Update docs minimally:
- Add documentation note "profitability not guaranteed" and clarify determinism/
replay goals.

Files to create/modify (minimum expected):
- ai_trading/oms/statuses.py (new)
- ai_trading/oms/intent_store.py (modify)
- scripts/migrate_oms_intent_store.py (modify)
- ai_trading/utils/determinism.py (new)
- ai_trading/execution/engine.py (modify)
- ai_trading/execution/simulator.py (modify/gate)
- ai_trading/execution/live_trading.py (modify)
- ai_trading/main.py (modify for cycle summary, as needed)
- ai_trading/security.py (modify)
- .github/workflows/{codeql.yml,dependency-audit.yml,sbom.yml,scorecard.yml}
(new)
- tests/unit/... (new tests listed above)
- tests/integration/... (new tests listed above)
- docs/ or README.md (add non-goal disclaimer + config snapshot notes)

Validation commands:
- ruff check .
- ruff format --check .
- mypy ai_trading tests
- pytest -q
- (optional) run determinism seed matrix locally: SEED=1/42/1337.

Use placeholders for credentials/secrets. Never add real keys.
```

External references used to validate time-sensitive or standards-driven requirements

- Alpaca trading API throttling and common 429 behavior: 200 requests/min/account. [5]
- Python interpreter hash randomization and `PYTHONHASHSEED` semantics (must be set before interpreter start for repeatable hashes). [4]
- systemd `StandardOutput=journal` semantics (journal routing). [7]
- Institutional automated trading controls frameworks (pre-trade risk controls, kill switches, post-trade analysis). [3]
- Execution cost / implementation shortfall as core measurement framing (transaction cost measurement and risks). [8]

---

1 3 Guide to the Development
https://www.fia.org/sites/default/files/2019-09/FIA-Guide-to-the-Development-and-Operation-of-Automated-Trading-Systems_1.pdf?utm_source=chatgpt.com

2 5 Alpaca Support - Is there a usage limit for the number of API calls per second?
https://alpaca.markets/support/usage-limit-api-calls?utm_source=chatgpt.com

4 1. Command line and environment — Python 3.12.12 documentation
https://docs.python.org/3.12/using/cmdline.html?utm_source=chatgpt.com

6 alpaca-trade-api · PyPI
https://pypi.org/project/alpaca-trade-api/1.2.0/?utm_source=chatgpt.com

7 systemd.exec - CS50 Manual Pages
https://manual.cs50.io/5/systemd.exec?utm_source=chatgpt.com

8 Trading Costs and Electronic Markets | CFA Institute
https://www.cfainstitute.org/insights/professional-learning/refresher-readings/2025/trading-costs-and-electronic-markets?utm_source=chatgpt.com