ChatGPT

Repo-wide Audit of dmorazzini23/ai-trading-bot

Executive summary

Enabled connectors: **github**.

Your repo already contains many “institutional-grade” building blocks: a formal institutional acceptance matrix with concrete code anchors and tests (e.g., runtime contract fail-fast, retry/idempotency rules, pretrade controls, kill-switch behavior, replay determinism, leakage checks, TCA/execution reporting, portfolio limits, learning bounds). [1]

However, there are several high-risk correctness and “institutional hardening” gaps that can lead to silent misbehavior, non-deterministic test outcomes, stuck reconciliation loops, and security degradations that are unacceptable in production:

- **Determinism is partially undermined by design choices that cannot be “fixed by setting env vars at runtime.”** Python hash randomization is enabled by default and `hash()` results are not predictable across interpreter runs unless controlled at interpreter start; setting `PYTHONHASHSEED` after startup does not retroactively make `hash()` deterministic. [2]
- **OMS terminal-status taxonomy looks incomplete relative to how execution reconciliation uses it** (e.g., statuses like `FAILED` or broker-native terminal statuses such as `expired` can become “non-terminal,” causing repeated “open intents” forever and repeated reconciliation work). This is visible in the shared terminal-status sets used by the OMS intent store and migration scripts.
- **Security “fallbacks” appear to fail open.** In `ai_trading/security.py`, cryptography absence creates a no-op `Fernet`, and encryption can silently degrade; master key generation can also become ephemeral unless properly forced for production.
- **Multiple simulation/execution paths exist, with inconsistent determinism guarantees.** You have a seeded deterministic simulated broker, but also a slippage/fill simulator that uses global `random` without a required seed, which can defeat replay parity goals if used unintentionally.
- **CI is already strong, but can be made “institutional-grade” by adding security and supply-chain gates (SAST/SCA/SBOM/Scorecard) and by making determinism/replay/idempotency gates non-optional.** Your repo already has institutional gates and smoke patterns in `Makefile` and gate scripts.

A critical security note from your shared context (outside the repo): the `.env` excerpt you pasted includes a **non-redacted database credential in** `DATABASE_URL`. Treat it as compromised: rotate the DB password/user immediately, revoke any leaked connection strings, and enforce secret scanning + secret store migration.

Information needs and scan actions

What I must learn to audit well

- The **true runtime entrypoints** and their call graph (live loop cadence, failure modes, reconciliation cadence, and whether multiple "mains" exist).
- How configuration is **validated, normalized, deprecated, and overridden** (and whether any "relaxed" fallbacks exist in production paths).
- The OMS's **durability and idempotency contract** end-to-end: intent creation → submit → fill/partial fill → reconciliation after crash/restart.
- The **concurrency and rate limiting model**, including host-level caps and retry policy (and whether retries are truly restricted to idempotent reads). [1]
- The **test/CI gating reality**: what is required vs optional, and how determinism is enforced (pytest plugin autoload off, deterministic replays, etc.).
- How security controls behave when dependencies are missing (fail closed vs fail open), and how secrets are prevented from leaking into logs/artifacts.

Audit information needs

- Dependency graph + pinning strategy, including constraints files and whether heavy ML deps are mandatory at runtime.
- Python version and toolchain assumptions (formatter, linter, type checker settings).
- OMS persistence: SQLAlchemy vs SQLite, schema evolution / migrations for intent store.
- Execution engine lifecycle: "active orders" model, reconciliation triggers, polling frequency, and backoff behavior.
- Cost model usage: how bounded costs influence gating (min edge bps, expected cost buffers, fallback penalties).
- Replay harness determinism guarantees and what inputs are required for parity tests. [1]
- Governance/promotion and artifact verification policy: what "verify" means (checksum only vs signature), what fails closed. [1]
- Observability: SLO targets, rate-limit dashboards, and alert criteria for degraded data/provider failover.

Immediate GitHub scan actions performed

Using the **github** connector, I sampled the repo's institutional gates and acceptance matrix, packaging and dependencies, core execution/OMS components, determinism/simulation components, and CI workflows (baseline for a repo-wide audit). [1]

Architecture and high-risk surfaces

Your repo documents an "institutional platform" architecture spanning core, database, risk, execution, strategy, monitoring, and governance components.

A practical "audit lens" architecture for correctness/performance is:

```
flowchart LR
  A[Market data providers] --> B[Feature + signal pipeline]
  B --> C[Risk engine + portfolio limits]
  C --> D[Pretrade gate: size/collars/dup-intent]
  D --> E[Idempotency + OMS intent store]
  E --> F[Broker adapter submit]
  F --> G[Order monitoring + reconciliation loop]
  G --> H[TCA + execution reports + decision records]
  H --> I[Governance: model artifacts + promotion]
  G -->|degraded| J[Provider failover + circuit breakers]
```

Key "institutional-grade" positives (what you're doing right):

- You explicitly codified institutional requirements into an acceptance matrix with **tests and deploy checks**, which is a strong sign of operational maturity. [1]
- You have an OMS intent store and idempotency layer, and you explicitly test restart reconciliation semantics (at least in integration).
- You invested in "institutional gates" in build tooling (`Makefile` targets for gate scripts, secret scan, exception audits).
- You maintain explicit dependency constraints and careful pinning for known ecosystem conflicts (e.g., `urllib3`/alpaca libs, `yfinance` websockets constraints), which is often necessary in production trading systems.

Highest-risk correctness/performance surfaces (prioritized):

- **Execution engine reconciliation loop**: risk of excessive broker polling, repeated reconciliation for intents that never become terminal, and cumulative latency/CPU waste.
- **Terminal status taxonomy mismatch** across OMS + engine + migration scripts: can create stuck "open intents" sets, repeated cancel/lookup, and eventual drift between broker truth and local truth.
- **Determinism leakage**: mixing seeded deterministic simulators with unseeded ones + using runtime-derived IDs compromises reproducibility and makes "replay parity" unreliable.
- **Security fail-open fallbacks**: cryptography absence or missing master keys should not silently degrade encryption posture in production.
- **Dependency footprint**: heavy ML deps in the default runtime environment can hinder deployment speed, memory, cold start latency, and increase CVE surface area.

Audit checklist for static and dynamic analysis

This checklist is designed to be systematic and reproducible, with quick "static" wins first and then "dynamic" determinism / race / idempotency probes.

Static analysis gate sequence

- **Formatting + lint** with Ruff:
- `ruff check .`

- `ruff format --check .` [3]

- **Type checking** with mypy:

- `mypy ai_trading` (then iterate into `--strict` by module as debt is paid down). [4]

- **Security lint** with Bandit:

- `bandit -r ai_trading -q` (upgrade to SARIF output in CI once stable). [5]

- **SAST** with Semgrep:

- `semgrep scan --config p/python --error --strict` (or `semgrep ci` for CI-aware mode). [6]

- **SCA** with pip-audit:

- `pip-audit -r requirements.txt` (and separately `pip-audit -r requirements-dev.txt`), then decide whether to use OSV mode and whether to fail CI on "unfixed" vulns. [7]

- **Secret scanning**:

- Keep (and enforce) your existing secret scan target.
- Add **pre-commit secret hooks** and a CI "hard fail" gate (see CI recommendations below).

Dynamic analysis gate sequence

- **Deterministic test runs + coverage**:
- `pytest -q` for fast paths (as you already do via deterministic smoke patterns).
- `pytest --cov=ai_trading --cov-report=term-missing --cov-fail-under=85` (ratchet upward). [8]

- For a lower-level baseline: `coverage run -m pytest && coverage report -m`. [9]

- **Replay determinism**:

- Run the repo's replay determinism tests and add a "determinism fingerprint" artifact (hash of decision log + TCA outputs) per seed/window. [1]

- **Race-condition probes**:

- Stress `IntentStore` concurrent updates (simulated multi-thread fills + cancellation) and assert state invariants.

- Run under `PYTHONHASHSEED=0` and `PYTHONHASHSEED=1` in CI matrix to catch accidental `hash()` determinism assumptions. Python hashes are salted and not predictable across runs unless controlled at startup. [2]

- **Idempotency + "exactly once"**:

- Crash/restart integration test: submit intent, crash post-submit pre-fill, restart, ensure no duplicate submit and intents go terminal.
- Add "broker status mapping" tests: `expired`, `replaced`, `stopped` mapped to terminal or handled explicitly.

- **Performance microbenchmarks**:

- Use your existing perf-check workflow as a base, extend with benchmark tests for indicator computation, data fetch concurrency, and reconciliation cadence.

Codex prompt requirement mapping to repo anchors

The attached Codex prompt emphasizes "institutional-grade" hardening: durable OMS, deterministic sim + cost modeling, governance, data provenance, migrations, alerts/safety/kill switch, and "no silent fallbacks." Your repo maps strongly to many of these via the acceptance matrix, but has a few mismatches and "fail-open" patterns.

Mapping table (audit view)

Codex requirement (interpreted)	Primary repo anchor(s)	Existing tests / gates	Audit notes / likely gaps
Durable OMS with restart-safe idempotency	`ai_trading/oms/intent_store.py`, `ai_trading/execution/idempotency.py`	Restart reconciliation integration test exists ; institutional matrix entries AC-05/06/13 [1]	Terminal status set likely incomplete (`FAILED`/`EXPIRED` risk); ensure "failed intents" do not stay "open" indefinitely
Deterministic replay harness and deterministic broker	`ai_trading/execution/simulated_broker.py`, replay modules referenced in acceptance matrix [1]	AC-13 replay deterministic gate [1]	Also a non-seeded simulator exists (`execution/simulator.py`); enforce a single deterministic sim path
Bounded cost model (no explosive costs)	`ai_trading/execution/cost_model.py`	Cost-floor / TCA references in matrix [1]	`CostModel.load()` returns defaults on failure—acceptable only if explicitly "fail closed" is intended and monitored
"No stubs / no silent fallbacks" runtime contract	Acceptance matrix AC-01 [1], config management `get_env` shim	Gate script `ci/scripts/institutional_gates.sh`	Some modules implement fail-open or "compat shims"; decide which are permitted in prod and enforce via runtime contract

Codex requirement (interpreted)	Primary repo anchor(s)	Existing tests / gates	Audit notes / likely gaps
Pretrade risk controls: size caps, collars, duplicate intent window	AC-05 in acceptance matrix [1]	Multiple pretrade tests referenced by matrix [1]	Ensure pretrade is invoked on every path (including "fallback order" / degraded modes)
Kill switch cancels orders and blocks submits	AC-06 [1]	Kill switch test and runbook referenced [1]	Add: "kill during submission" + "kill after partial fill" invariants
Governance: artifact verification + fail-closed policy	AC-22 [1] and `ai_trading/models/artifacts.py`	Artifact tests referenced by matrix [1]	Artifact verification appears checksum-based; if the Codex prompt expects signatures, add signing + key management
Promotion workflow and shadow/prod separation	`ai_trading/governance/promotion.py` (exists; governance in matrix) [1]	Promotion gates in matrix (shadow/prod via governance status) [1]	Ensure promotion cannot use `verify_dataset_hash=False` in true prod promotions unless explicitly justified
Data provenance included in manifests	`ai_trading/registry/manifest.py`	Manifest validation unit tests exist	Validate "data sources" are not empty and are propagated into decision snapshots
Database migrations / schema versioning	`scripts/migrate_oms_intent_store.py`	Not clearly tied to Alembic migrations	If you truly need institutional DB evolution, add real schema migrations + version markers
Security & secret hygiene	`ai_trading/security.py` and secret-scan target	Secret scan target exists	Change "crypto missing" from fail-open to fail-closed; enforce secret scanning in CI
CI institutional gates and smoke determinism	`.github/workflows/ci.yml` , `Makefile`	Gate scripts present	Add SAST/SCA/SBOM/Scorecard; add deterministic seed matrix

Likely bug classes to search for and top tests to add

Bug classes to systematically search for

This is the “search list” I recommend driving through with grep + Semgrep + focused code review:

- Off-by-one and boundary logic in sizing/collars/price steps (especially when quantity is rounded or limited post-signal).
- Unseeded randomness and time-derived IDs (non-reproducible replay; nondeterministic tests) and unintended use of `hash()`. Python hashing is salted across runs by default. [2]
- Improper exception handling (broad excepts that silently downgrade correctness; “warn and continue” on critical invariants). Your repo already has an exception audit target, which is good.
- Silent fallbacks (data-provider failover, crypto absence, “default cost model” loads) without explicit “degraded mode” state and alerts.
- Duplicated/aliased config keys where precedence is unclear or differs by module (`AI_TRADING_*` vs legacy keys).
- SQL transaction misuse (missing atomicity for fill + status updates; race between cancel and fill; isolation level assumptions).
- Time-zone bugs (naive datetimes, local vs UTC drift, day-boundary logic in NYC vs server TZ).
- Improper resource cleanup (thread pools, sessions, file handles).
- Unsafe secret handling (accidental logging of env vars; persistence of secrets in artifacts).
- “Stuck open intent” states due to status mapping gaps, leading to repeated reconciliation and broker API load.

Top high-risk tests to add

Below are **12 concrete tests** to add that directly target the most likely high-severity failure modes. They are intentionally “reproducible” and should run in CI without broker credentials (via mocks or the simulated broker).

> Notes: These snippets are examples; adjust imports to your test layout (you already have `tests/unit` and `tests/integration` patterns). [1]

Terminal status completeness and reconciliation sanity

```python
# tests/unit/test_intent_store_terminal_statuses.py
from ai_trading.oms.intent_store import IntentStore

def test_intent_store_terminal_statuses_include_failed_and_expired(tmp_path):
    store = IntentStore(url=f"sqlite:///{tmp_path}/test.db")

    intent, _ = store.create_intent(
        intent_id="i1", idempotency_key="k1", symbol="AAPL", side="buy",
        quantity=1, decision_ts="2026-01-01T00:00:00Z",
    )
    store.close_intent(intent.intent_id, final_status="FAILED", last_error="x")
```

```
    # Expect FAILED to be treated as terminal (no longer returned as open).
    open_intents = store.get_open_intents()
    assert all(i.intent_id != "i1" for i in open_intents)
```

Why: prevents infinite reconciliation loops when `FAILED` intents never become terminal.

```
# tests/unit/test_broker_status_mapping_terminal.py
def test_expired_status_mapped_to_terminal(monkeypatch, tmp_path):
    # When broker reports "expired", intent should become terminal and stop
retrying.
    # Implement mapping in engine/adapter, then assert it.
    assert True  # replace with engine mapping assertion
```

Determinism and “no accidental hash()”

```
# tests/unit/test_no_builtin_hash_in_deterministic_paths.py
import inspect
import ai_trading.execution.engine as engine

def test_execution_engine_does_not_use_builtin_hash_for_determinism():
    src = inspect.getsource(engine)
    assert "hash(" not in src,
"Avoid builtin hash() in deterministic simulation paths"
```

Motivation: `hash()` is salted across runs unless controlled at interpreter start, and can’t be made deterministic by setting env var after startup. [2]

```
# tests/unit/test_simulated_broker_reproducible.py
from ai_trading.execution.simulated_broker import SimulatedBroker

def test_simulated_broker_is_reproducible():
    b1 = SimulatedBroker(seed=42)
    b2 = SimulatedBroker(seed=42)
    o = {"symbol": "AAPL", "qty": 10, "side": "buy", "type": "limit",
"limit_price": 100}

    r1 = b1.submit_order(o, timestamp="2026-01-01T00:00:00Z")
    r2 = b2.submit_order(o, timestamp="2026-01-01T00:00:00Z")

    assert r1["id"] == r2["id"]
```

Why: hard requirement for replay parity.

Simulation path hygiene

```python
# tests/unit/test_no_unseeded_global_random_simulator_usage.py
import ai_trading.execution.simulator as sim

def test_simulator_requires_explicit_seed(monkeypatch):
    # Example: enforce construction requires seed or uses injected Random
    assert hasattr(sim, "FillSimulator")
```

Why: `execution/simulator.py` uses global `random`; enforce explicit seeding/injection or ensure it is never used in deterministic replay mode.

Idempotency and “exactly once submit” across restart

```python
# tests/integration/test_exactly_once_submit_across_restart.py
from ai_trading.oms.intent_store import IntentStore

def test_exactly_once_submit_across_restart(tmp_path):
    url = f"sqlite:///{tmp_path}/oms.db"
    store1 = IntentStore(url=url)

    intent, created = store1.create_intent(
        intent_id="i1", idempotency_key="key", symbol="AAPL", side="buy",
        quantity=1, decision_ts="2026-01-01T00:00:00Z",
    )
    assert created

    store1.mark_submitted(intent.intent_id, broker_order_id="broker-1")

    # "Restart"
    store2 = IntentStore(url=url)
    intent2 = store2.get_intent("i1")
    assert intent2 is not None
    assert intent2.broker_order_id == "broker-1"
```

Why: institutional OMS property.

Security fail-closed checks

```python
# tests/unit/test_security_crypto_required_in_production.py
import os
import ai_trading.security as sec

def test_crypto_missing_fails_closed(monkeypatch):
    monkeypatch.delenv("MASTER_ENCRYPTION_KEY", raising=False)
```

```
    # In production, you should require MASTER_ENCRYPTION_KEY and crypto
availability.
    # Replace this with your new fail-closed behavior.
    assert True
```

Why: crypto fallbacks must not silently become “no encryption.”

Host-level concurrency cap and resource cleanup

```
# tests/unit/test_host_slot_limits_concurrency.py
import threading
from ai_trading.utils.http import host_limit_snapshot, host_slot

def test_host_slot_enforces_limit(monkeypatch):
    # Set env var to force limit = 1, then assert peak inflight never exceeds 1.
    monkeypatch.setenv("HTTP_MAX_CONNS_PER_HOST", "1")

    def worker():
        with host_slot("example.com"):
            pass

    threads = [threading.Thread(target=worker) for _ in range(10)]
    for t in threads: t.start()
    for t in threads: t.join()

    snap = host_limit_snapshot("example.com")
    assert snap["peak"] <= 1
```

Why: prevents runaway concurrency and vendor throttling issues.

Cost model calibration and bounds invariants

```
# tests/unit/test_cost_model_bounds.py
from ai_trading.execution.cost_model import CostModel

def test_cost_model_output_bounded():
    cm = CostModel()
    bps = cm.estimate_cost_bps(spread_bps=10_000, volatility_pct=10.0,
participation_rate=1.0, tca_cost_bps=10_000)
    assert 0 <= bps <= 10_000  # tighten to expected max_bps once frozen
```

Why: “bounded outputs” requirement.

Migration correctness for terminal statuses

```
# tests/unit/test_migration_terminal_statuses.py
from scripts.migrate_oms_intent_store import _TERMINAL_STATUSES

def test_terminal_statuses_include_failed():
    assert "FAILED" in _TERMINAL_STATUSES
```

Why: mismatches here can permanently poison the “open intents” set post-migration.

Coverage regression guard

```
# tests/unit/test_coverage_floor.py
def test_repo_coverage_floor_is_enforced():
    # This is enforced by CI flags, but keep a placeholder doc-test to avoid
slow drift.
    assert True
```

Rationale: CI-level `--cov-fail-under` is where the enforcement lives. [8]

Remediation roadmap, CI hardening, monitoring, and security actions

Prioritized remediation steps with effort and risk

Highest priority (blockers for “institutional-grade”)

- Fix OMS terminal status taxonomy end-to-end (IntentStore + reconciliation + migration script): add `FAILED`, `EXPIRED`, and any broker-native terminal statuses you observe; add strict mapping and tests so “open intents” cannot include terminal failures.
  Effort: 0.5–1 day. Risk: high (can prevent runaway reconciliation and repeated orders).

- Remove any reliance on `hash()` or time-derived IDs for determinism in replay/simulation paths; replace with stable hashes (`sha256`) and explicit seed injection. Python hashing is salted across runs by default.
  Effort: 1–2 days. Risk: high (replay parity + reproducible CI). [2]

- Convert security fallbacks to fail-closed in production: require cryptography availability and require `MASTER_ENCRYPTION_KEY` (or an equivalent secret) for “production mode,” rather than silently disabling encryption.
  Effort: 0.5–1 day. Risk: high (secret exposure and false sense of security).

Medium priority (material performance + operational risk)

- Consolidate simulation modules: ensure only the deterministic simulator is used for deterministic replay, and the non-deterministic simulator cannot run unless explicitly requested.
  Effort: 1 day. Risk: medium.

- Tighten "no silent fallback" policy: treat missing optional dependencies as explicit "feature unavailable" errors, except in dev. You already codified "runtime contract" in acceptance matrix—enforce it uniformly.
  Effort: 1–3 days. Risk: medium. [1]

- Reduce default dependency footprint for production: split heavy ML deps (`torch`, `transformers`) into an extra, and keep runtime lean unless features are enabled.
  Effort: 0.5–1 day. Risk: medium (deployment speed, memory, attack surface).

Lower priority (quality-of-life, long-term debt)

- Remove or unify overlapping Ruff config locations (`pyproject.toml` vs `.ruff.toml`) so CI/local runs are consistent and developers aren't confused. [10]
  Effort: 0.25–0.5 day. Risk: low.

CI pipeline changes to catch regressions

You already have CI workflows and institutional gates.

Add these "institutional-grade" security/supply-chain gates:

- **CodeQL** for code scanning (SAST) via the official action. [11]
- **Semgrep** job for Python rulesets, configured to fail on new high-severity findings. [12]
- **Bandit** job for Python security lint. [5]
- **pip-audit** for requirements + dev requirements scanning; decide policy for "unfixed" CVEs. [7]
- **OpenSSF Scorecard** workflow to enforce secure repo practices and publish SARIF where possible. [13]
- **SBOM publication** (Syft) on release tags (CycloneDX JSON and SPDX JSON). [14]

Coverage and determinism gates:

- Enforce **coverage floor** with `pytest-cov` and `--cov-fail-under`. [8]
- Add a CI matrix for determinism: run core tests twice with different `PYTHONHASHSEED` (e.g., 0 and 1) to catch accidental determinism assumptions. Hash randomization is enabled by default. [2]

Monitoring and alerting rules

You already track SLO and rate limit docs/modules; tune them to explicitly detect "silent degradation."

Institutional-grade alert rules to add:

- **OMS health**: count of open intents by status; alert if any intent remains non-terminal beyond threshold (e.g., > 30 minutes) or if "FAILED but still open" appears.
- **Idempotency collisions**: count and rate of duplicate-intent blocks; alert on spikes.
- **Broker API error budget**: 429/5xx rates; alert if sustained above SLO.
- **Provider failover / degraded mode**: alert if running on fallback feeds beyond a bounded window.
- **Reconciliation churn**: if reconciliation runs > N times for same intent/order without state change.
- **TCA staleness**: if TCA sample count stale beyond configured window (since other systems depend on it).

Security actions you should take now

- **Immediately rotate the exposed DB credential** in your environment (and any secrets in pasted logs). Treat as compromised.
- Migrate secrets to a real secret store (for example, DigitalOcean [15] App Platform/Secrets or vault-like tooling) and ensure `.env` files never contain production secrets in plaintext.
- Enforce secret scanning:
- Pre-commit hooks (you already have a pre-commit config).
- CI secret scan gate (you already have `secret-scan` target).
- Ensure encryption is **fail-closed** in production (`cryptography` required, master key required).

Assumptions and explicit unknowns

Assumptions made for this audit:

- CI is running on GitHub Actions, and Python target is 3.12 (as implied by Ruff config and CI).
- "Institutional acceptance matrix" is intended to be normative (i.e., tests listed there are the "definition of done" for production readiness). [1]

Unspecified / not evidenced in the sampled materials:

- Whether production deployment uses containers, systemd, or a platform like DigitalOcean [15] App Platform (deployment docs exist but were not exhaustively audited here).
- Exact broker adapters and status mappings for every broker terminal status (needs a targeted review of adapters and reconciliation code paths).

---

[1] docs/acceptance_matrix.md
https://github.com/dmorazzini23/ai-trading-bot/blob/7fd45b9b3be83872278b27cd1807fa1166b449e6/docs/acceptance_matrix.md

[2] https://docs.python.org/3.11/using/cmdline.html
https://docs.python.org/3.11/using/cmdline.html

[3] [10] https://github.com/astral-sh/ruff
https://github.com/astral-sh/ruff

[4] https://github.com/python/mypy
https://github.com/python/mypy

[5] https://bandit.readthedocs.io/
https://bandit.readthedocs.io/

[6] https://github.com/semgrep/semgrep
https://github.com/semgrep/semgrep

[7] https://github.com/pypa/pip-audit
https://github.com/pypa/pip-audit

[8] https://github.com/pytest-dev/pytest-cov
https://github.com/pytest-dev/pytest-cov

[9] https://coverage.readthedocs.io/
https://coverage.readthedocs.io/

[11] https://github.com/github/codeql-action
https://github.com/github/codeql-action

[12] https://semgrep.dev/docs/semgrep-ci/sample-ci-configs
https://semgrep.dev/docs/semgrep-ci/sample-ci-configs

[13] https://scorecard.dev/
https://scorecard.dev/

[14] https://oss.anchore.com/docs/guides/sbom/formats/
https://oss.anchore.com/docs/guides/sbom/formats/

[15] ai_trading/oms/intent_store.py
https://github.com/dmorazzini23/ai-trading-bot/blob/7fd45b9b3be83872278b27cd1807fa1166b449e6/ai_trading/oms/intent_store.py