InquisitorNet Phase Modules Guide

Overview
InquisitorNet is organized into three explicit phases that build on one another:

- **Phase 1 (Foundations)** ingests data and runs the first-pass detector.
- **Phase 2 (Calibration + Policy Gate)** adds labeling, metrics, and an auditable policy gate for drafts.
- **Phase 3 (Inquisitor Actions)** plans actions (posts, dossiers, logs) based on marks, and reuses the policy gate to keep outputs compliant.

Each phase is implemented as a Python package (‘phase1/‘, ‘phase2/‘, ‘phase3/‘) with one or more CLI entry points. The phases share configuration and the SQLite database schema stored under ‘migrations/‘.

Phase 1: Foundations (Ingestion + Detection)

Purpose: Gather candidate content, persist it in SQLite, and run a deterministic rule-based detector to mark or acquit items.

‘phase1/cli.py‘
- The orchestration entry point for Phase 1.
- Loads settings from ‘config/‘ via ‘phase1.config.Settings‘.
- Applies database migrations (‘migrations/001_init.sql‘ and ‘migrations/004_offline_fixtures.sql‘).
- Runs the scraper (‘phase1.scraper.run_scraper_to_db‘) and detector (‘phase1.detector.run_detector_to_db‘).
- Outputs a summary of how many items were kept, marked, or acquitted.

‘phase1/config.py‘
- Loads YAML configuration for all Phase 1 components.
- Sources:
  - ‘config/subreddits.yml‘ (mode selection, fixtures path, allow list)
  - ‘config/scraper_rules.yml‘ (scraper keyword rules)
  - ‘config/detector_rules.yml‘ (detector patterns and thresholds)
- Stores the resolved database path (defaults to ‘inquisitor_net_phase1.db‘).

‘phase1/db.py‘
- Thin helpers to connect to SQLite and execute migrations.
- ‘get_conn()‘ ensures the parent directory exists and returns a ‘sqlite3.Connection‘.
- ‘migrate()‘ executes a given SQL script and commits.

‘phase1/scraper.py‘
- Implements the ingestion step that yields and stores candidate items.

- Inputs depend on mode:
  - **fixtures:** reads JSONL from ‘fixtures/reddit_sample.jsonl‘.
  - **offline:** reads local DB table rows for deterministic tests.
  - **api:** streams real Reddit comments via ‘core.reddit_client.RedditClient‘.
- Applies inclusion/exclusion regex rules from ‘config/scraper_rules.yml‘.
- Uses a simple ‘discard_if‘ mechanism (e.g., ‘len(body) < N‘) to skip short content.
- Writes kept rows to ‘scrape_hits‘ with redacted author tokens and stored regex hits.

‘phase1/detector.py‘
- Runs a rule-based detector over ‘scrape_hits‘.
- Compiles regex rules from ‘config/detector_rules.yml‘ into weighted patterns with optional exculpatory matches.
- Scores each item and compares it against thresholds:
  - **mark** â inserts into ‘detector_marks‘.
  - **acquit** â inserts into ‘detector_acquittals‘.
  - **hold** â leaves the item only in ‘scrape_hits‘ for later review.
- Generates rationale strings with ‘explain_noop()‘ to document decisions.

Related components
- ‘core/reddit_client.py‘ is the thin wrapper over ‘praw‘ used by the scraper in API mode.
- ‘migrations/001_init.sql‘ defines foundational tables: ‘scrape_hits‘, ‘detector_marks‘, ‘detector_acquittals‘.

Phase 2: Calibration + Policy Gate

Purpose: Improve detector quality via human labeling and metrics while adding an auditable policy gate for outgoing drafts.

‘phase2/gate.py‘
- Core logic for policy gating.
- Loads regex policy rules from ‘config/policy_gate.yml‘ into ‘GateRule‘ objects.
- Evaluates text and returns a ‘GateDecision‘ with:
  - **decision:** ‘allow‘, ‘flag‘, or ‘block‘.
  - **reasons:** matched rule metadata and snippets.
  - **llm_reason:** optional explanation from a stub LLM provider.
- Default policy: any ‘block‘ hit â block; else if ‘flag‘ score â¥ 1 â flag; otherwise allow.

‘phase2/gate_cli.py‘
- CLI wrapper that reads a JSONL file of drafts (‘{"text": ...}‘) and writes policy decisions.
- Invokes ‘phase2.gate.check_draft()‘ for each item.
- Produces an auditable JSONL output with decision metadata and reasons.

‘phase2/label_cli.py‘
- CLI tool for human labeling of detector outcomes.
- Samples items from ‘detector_marks‘ and ‘detector_acquittals‘.
- Prompts for TP/FP/TN/FN labels and stores them in the ‘labels‘ table.

‘phase2/metrics_job.py‘
- Aggregates labels into detector metrics (precision, recall, F1).
- Outputs both CSV and Markdown reports under ‘reports/metrics/‘.

Related components
- ‘config/policy_gate.yml‘ contains regex rules and actions (note/flag/block).
- ‘migrations/002_phase2.sql‘ adds Phase 2 tables such as ‘labels‘ and ‘policy_checks‘.

Phase 3: Inquisitor Actions

Purpose: Translate detector marks into concrete planned actions (posts, dossiers, logs) while reusing the policy gate to prevent unsafe outputs.

‘phase3/inquisitor_cli.py‘
- CLI pipeline for Phase 3.
- Reads a JSONL file of marks (‘item_id‘, ‘score‘, ‘rationale‘).
- Uses ‘phase3.bots.BaseBot‘ to decide an action for each mark.
- If the action is a **post**, it is passed through the Phase 2 policy gate; blocked or flagged posts are downgraded to dossiers.
- Inserts planned actions into ‘planned_actions‘ and generated dossiers into ‘dossiers‘.

‘phase3/bots/base.py‘
- Defines the simplest decision policy via ‘BaseBot‘:
  - score â¥ 0.8 â create a **post** draft.
  - score â¥ 0.5 â create a **dossier**.
  - otherwise â **log** only.
- ‘InquisitorPersonality‘ provides a minimal persona configuration (name, style, traits) for future expansion.

Related components
- Phase 3 stores outputs in SQLite tables created by ‘ensure_phase3_tables()‘ inside the CLI.
- It reuses ‘phase2.gate.check_draft()‘ for the same policy standards established in Phase 2.

End-to-End Flow (Quick Summary)
1. **Phase 1** ingests content and stores ‘scrape_hits‘, then marks or acquits via regex

scoring.
2. **Phase 2** lets humans label detector results and generates daily metrics; it also adds the policy gate for outgoing drafts.
3. **Phase 3** consumes marks to plan actions and routes any public-facing draft through the Phase 2 gate.

Suggested Starting Points for Newcomers
- Start with ‘phase1/cli.py‘ to see the entry point and control flow.
- Follow the data flow in ‘scraper.py‘ â ‘detector.py‘ â SQLite tables.
- Explore ‘phase2/gate.py‘ to understand the policy logic used in later phases.
- Review ‘phase3/inquisitor_cli.py‘ and ‘phase3/bots/base.py‘ for the action planning layer.