ChatGPT

InquisitorNet project – status review & Phase-2/3 planning (Sep 24 2025)

1 Current repository status

The `InquisitorNet` repository has been partially modularised. It now contains a **Phase-1** scraper/detector pipeline as well as a **Phase-2** scaffolding for policy gate, labelling and metrics. The original monolithic script (`inquisitor_net.py`) is still present but has not been refactored. Key observations:

area	current implementation	evidence
Phase 1 data pipeline	A command-line interface (`phase1/cli.py`) orchestrates a scraper and regex-based detector. Configuration is loaded from YAML (`config/subreddits.yml`, `scraper_rules.yml`, `detector_rules.yml`). The scraper iterates over fixture data, applies inclusion/exclusion regexes and stores hits into an SQLite DB; the detector compiles rule patterns, scores each hit and inserts marks/acquittals into separate tables [1] [2].	Implementation in `phase1` modules and configuration files.
Phase 2 policy gate	`phase2/gate_cli.py` implements a **dry-run** policy gate. It reads regex checks from `config/policy_gate.yml`, processes draft posts from a JSONL file and writes allow/deny decisions into the `policy_checks` table [3] [4]. Decision logic uses simple regex matching and a stubbed LLM reasoner; the gate currently only logs flags and does not integrate with posting.	`gate_cli.py` and `config/policy_gate.yml`.
Phase 2 labelling	`phase2/label_cli.py` provides a small labelling tool. It samples a few recent detector marks from the DB, and in "auto-skip" mode writes placeholder TN labels [5]. Interactive labelling is not yet user friendly and there is no near-threshold sampling.	`label_cli.py`.
Phase 2 metrics	`phase2/metrics_job.py` aggregates counts of TP/FP/TN/FN labels over the last *N* days to compute precision/recall/F1 and writes a daily row into `metrics_detector_daily` [6]. It is intended to run as a scheduled job, but there is no scheduler integration yet.	`metrics_job.py`.

area	current implementation	evidence
Database migrations	The Phase 2 migration (`migrations/002_phase2.sql`) defines tables for policy gate results, labels and daily metrics [7]. Phase-1 migration exists but is not versioned.	SQL files.
Original monolithic bot	`inquisitor_net.py` still contains the original "Inquisitor" bot logic – personalities, encryption, database routines, posting and replying via PRAW, and simple scheduling [8]. This file has not been reconciled with the modular Phase-1/2 pipeline and there is duplication of database functionality.	`inquisitor_net.py`.

2 Comparison with high-level outlines

Your earlier outline (attached separately) emphasised **modularisation**, **unit tests**, and a staged rollout of features. Comparing that outline to the current code:

1. **Phase 1** – The scraper/detector modules exist and largely follow the outline: fixture-based scraping, regex filtering, threshold-based marking and acquittal, and storing results in an SQLite DB. However, there are still TODOs such as API-mode scraping and sentiment-based features (disabled in the detector config). Tests are missing.
2. **Phase 2** – The outline described a policy gate that uses both regex rules and lightweight LLM reasoning, an interactive labelling workflow for human calibration, and metrics generation. The current repository contains CLI tools for these tasks, but they remain **dry-run**: the gate writes decisions to the DB but is not wired into any automated posting pipeline; the LLM reasoner is stubbed; labelling only writes placeholder TN labels; and metrics are calculated but not scheduled or visualised. The outline also suggested a more generic rule engine and proper unit tests, both of which are missing.
3. **Refactoring** – The outline recommended breaking up the monolithic `inquisitor_net.py` into modules and integrating it with the new phases. This refactoring has not been done; the old file still exists and duplicates functionality (database manager, bot behaviours) while Phase-1/2 code uses a separate DB schema.

In summary, the current repository matches the early part of the outline for Phase 1, but Phase 2 is only scaffolded and lacks the integration, configurability and testing described. Phase 3 has not been started.

3 Plan to complete Phase 2

To finish Phase 2 we need to move beyond the dry-run scaffolding and make the policy gate, labelling and metrics pipeline production-ready. The following tasks are grouped by area.

3.1 Policy gate

- **Parameterise & test rules:**

- Expand `policy_gate.yml` to cover more policy scenarios (e.g. IP infringement, personal data leaks, safety/harassment) and allow rule weights/actions. Add unit tests to ensure patterns are compiled and matched correctly.
- **LLM integration:**
- Replace the `llm_reason_stub` with a real call to a lightweight model (e.g. OpenAI `gpt-3.5-turbo`) when allowed. Wrap the call behind a provider interface so it can be stubbed in tests. The reasoner should summarise which checks triggered and justify allow/flag/block decisions [9].
- **Real-time wiring:**
- Convert `gate_cli.py` into a reusable module `phase2/gate.py` exposing a `check_draft` function that accepts text and returns a decision/flags/reason. Update the CLI to call this module.
- Integrate the gate into the posting pipeline (currently in `inquisitor_net.py` or a future Phase-3 poster): before posting publicly, run candidate content through the gate and drop/flag posts accordingly.
- **Testing:**
- Write unit tests for the rule compilation, decision logic and LLM stub. Use fixture drafts to verify allow/block outcomes.

3.2 Labelling workflow

- **Sample near-threshold cases:**
- Extend `sample_item_ids` to include items whose detector scores lie between the mark and acquit thresholds, in addition to confirmed marks. This will expose ambiguous cases for human review.
- **Interactive UI:**
- Provide a more user-friendly labelling mode (e.g. a simple console interface or web UI) that displays the post body and allows quick TP/FP/TN/FN selection. Persist labels to the `labels` table.
- **Auto-skip removal:**
- Remove the auto-skip placeholder logic once the label UI is available. The CLI should require explicit labels or skip.
- **Testing & docs:**
- Add unit tests for sampling and labelling functions. Document how to run the labelling CLI and how many items should be labelled daily to maintain metrics quality.

3.3 Metrics & reporting

- **Schedule metrics job:**
- Use `APScheduler` or an external cron to run `metrics_job.py` daily. Parameterise the look-back window and database path via configuration.
- **Integrate with Phase-1 DB:**
- Currently the metrics job reads from `inquisitor_net_phase1.db`. Ensure all Phase-1/2 modules write to the same DB or provide migration scripts to unify schemas.
- **Visualisation:**
- Design simple reporting (e.g. generate a CSV or Markdown report) summarising daily precision/recall/F1 scores. Later, integrate with a dashboard.
- **Testing:**
- Add tests that insert known counts into the labels table and verify the metrics computation [6].

3.4 Documentation & refactoring

- **Refactor monolith:**
- Extract reusable components (database manager, bot personality definitions, encryption helpers) from `inquisitor_net.py` into a `core` package. Delete unused sections once replaced.
- **Update README:**
- Describe the Phase-1 and Phase-2 pipelines clearly, including how to run scrapers, detectors, gates, labelling, and metrics jobs. Note any environment variables required.
- **Add tests & CI:**
- Introduce a test suite (e.g. using `pytest`) and a GitHub Actions workflow to run tests on pushes. Start with tests for Phase-1/2 modules.

Completing the tasks above will transform Phase 2 from a dry scaffold into a functional policy gating and feedback loop, ready for real content.

4 Early planning for Phase 3

Phase 3 will connect the detection/gating pipeline to real Reddit interactions and add sophisticated bot behaviour. Based on the existing monolithic bot code, the following high-level plan is proposed:

1. **Modular bot framework**
2. **Core modules:** Create a `bots` package with classes such as `InquisitorPersonality`, `BotMemory`, `EncryptionModule` and a `BaseBot` with common logic (rate-limiting, logging, memory storage, posting and replying). The current implementations exist in `inquisitor_net.py` [8] and should be migrated into separate files for reusability.
3. **Adapters:** Implement adapters for Reddit (`praw`) and other potential platforms. Each adapter should expose methods for fetching new posts/comments, submitting posts, and replying.
4. **Content generation & scheduling**
5. **Prompt generation:** Build a prompt engine that uses personality traits, speech patterns, and conversation context to produce prompts for the LLM [10]. Make prompts configurable per personality.
6. **LLM integration:** Define a service layer for generating responses using OpenAI models (or local models where privacy is a concern). Include error handling and fallback messages [11].
7. **Scheduler:** Use `APScheduler` to schedule scraping, detection, policy gating and posting tasks. The scheduler should manage multiple bots, ensuring they respect cooldowns and daily post limits [12].
8. **End-to-end pipeline**
9. **Streamed scraping:** Replace the fixture-based scraper with Reddit stream listeners to collect new posts and comments in near-real time. Feed them through the Phase-1 detector and Phase-2 policy gate.
10. **Decision logic:** Only content that passes the policy gate should be handed off to the bot for response generation. Blocked or flagged content should be logged for review or human escalation.

11. **Investigation workflow:** Incorporate the `investigations` table from `inquisitor_net.py` to allow moderators to track serious offences. Provide CLI commands for opening, closing and assigning investigations.

12. **Additional Phase-3 features**

13. **Personality management:** Store personalities and traits in a configuration file or database rather than hard-coding them. Allow easy onboarding of new Inquisitors.
14. **Encryption & secrecy:** Use the `EncryptionModule` to encrypt sensitive messages automatically when certain keywords are detected [13].
15. **Metrics & feedback:** Extend the metrics system to monitor bot engagement (posts, replies, upvotes/downvotes) and the impact of policy gating decisions. Use these metrics to refine detector thresholds and gate rules.
16. **Testing & simulation:** Build a test harness that simulates Reddit interactions using fixtures so that bot behaviour can be tested without hitting the live API.

By implementing the above plan, Phase 3 will transform InquisitorNet from an offline detection pipeline into an active, policy-compliant bot network capable of interacting with Reddit communities in an in-character Warhammer 40K style.

---

[1] GitHub
https://github.com/johnson-liu-code/InquisitorNet/blob/main/phase1/scraper.py

[2] GitHub
https://github.com/johnson-liu-code/InquisitorNet/blob/main/phase1/detector.py

[3] [4] [9] GitHub
https://github.com/johnson-liu-code/InquisitorNet/blob/main/phase2/gate_cli.py

[5] GitHub
https://github.com/johnson-liu-code/InquisitorNet/blob/main/phase2/label_cli.py

[6] GitHub
https://github.com/johnson-liu-code/InquisitorNet/blob/main/phase2/metrics_job.py

[7] GitHub
https://github.com/johnson-liu-code/InquisitorNet/blob/main/migrations/002_phase2.sql

[8] [10] [11] [12] [13] GitHub
https://github.com/johnson-liu-code/InquisitorNet/blob/main/inquisitor_net.py