Architecture Roadmap: The Unified Engine Migration

Version: 2.0.4 **Date:** 2025-12-11 **Status:** Active

---

1. The Strategic Vision

Goal: Create a single analysis engine that powers both a Command-Line Interface (CLI) and a Web Interface (Django). **Core Constraint:** "Write Once, Render Everywhere." Logic must not be duplicated. **Deployment Model: Stateless & Portable.**

- No User Accounts. No User Database. Atomic Sessions.
- Containerized (Docker) to ensure "Laptop == Server".

2. The "Golden Path" Workflow

This defines the specific user experience and data flow for the Web Interface.

1. **The Three-Slot Model:** The analyzer presents **three generic input slots** (Log A, Log B, Log C).
2. **Identity Agnostic:** There is no concept of "My Log" vs. "Competitor."
3. **Source Agnostic:** Each slot can be filled by either:
   - **Direct Upload:** User uploads a local `.log` file.
   - **Public Fetch:** User selects a contest/year/callsign (Phase 4).
4. **Zero Persistence:**
   - Files (uploaded or fetched) exist only for the duration of the request processing.
   - Once the report is delivered to the browser, all source logs are purged.

3. Architectural Decision Records (ADRs)

ADR-001: Data Abstraction Layer (DAL)

- **Decision:** All business logic exists in `data_aggregators/`. Returns **Pure Python Primitives**.

ADR-002: Unified Visualization (Client-Side Rendering)

- **Decision:** Replace Matplotlib with **Plotly**.
- **Animation:** Replace server-side MP4 generation with **Plotly HTML Animations**.

ADR-007: Shared Presentation Layer

- **Decision:** The Web App (Django) must point to the existing `contest_tools/templates`.
- **Reason:** Ensures CLI HTML reports and Web Views are bit-for-bit identical.

ADR-009: The Ephemeral Fetcher Pattern

- **Decision:** Public Logs are **not** stored permanently.
- **Mechanism:** The system fetches specific logs on-demand into a temporary session directory.

---

4. Master Transition Timeline

Phase 1: Data Decoupling (COMPLETE)

- **Status:** Complete. DAL is operational.

Phase 2: Visualization Standardization (COMPLETE)

- **Status:** Complete. Static charts migrated to Plotly.

Phase 2.5: Animation Modernization (ACTIVE)

- **Goal:** Eliminate Matplotlib/FFmpeg dependencies for animations.
- **Constraint:** Establish the `.html` file as the definitive artifact for regression testing.
- **Tasks:**
  - ☒ Implement `plot_interactive_animation.py` (Plotly).
  - ☐ Deprecate `plot_hourly_animation.py` (Matplotlib). *(Plan Delivered)*
  - ☐ Update `run_regression_test.py`. *(Plan Delivered)*

Phase 3: The Web Pathfinder (CQ WW & Uploads)

- **Goal:** A fully functional, stateless web application (MVP) running locally and deployable.
- **The Pathfinder:** We use **CQ WW** as the primary test case to validate the UI/UX.
- **Tasks:**
  - ☐ **Containerization:** Create `Dockerfile` and `docker-compose.yml` (Python only, no FFmpeg).
  - ☐ **Django Bootstrap:** Initialize the project skeleton.
  - ☐ **UI Implementation:**
    - Build the **Three-Slot Upload Form**.

    - Build the **Report Dashboard View** (Rendering DAL outputs via Jinja2).
  - ☐ **Validation (The Pathfinder Run):** Verify full "Upload -> Process -> View" loop using complex **CQ WW** logs.

Phase 4: The Data Layer (Public Log Access)

- **Goal:** Enable "Select from Public Source" in the UI.
- **Tasks:**
  - ☐ **Scrapers:** Create lightweight Python adapters to find/fetch logs.
  - ☐ **UI Integration:** Add "Fetch URL" tabs to the Input slots.
  - ☐ **Async Workers:** Introduce Celery/Redis *only if* fetching/processing exceeds HTTP timeouts.