PoliRag

PoliformaT Retrieval-Augmented Generation

0xbiel

February 4, 2026

Contents

1 Introduction

PoliRag (PoliformaT Retrieval-Augmented Generation) is a comprehensive personal knowledge base application designed for students of the Universitat Politècnica de València (UPV). It bridges the gap between the university's Learning Management System (PoliformaT) and modern Large Language Models (LLMs). By automating the ingestion of course materials and providing a local, semantic search interface, PoliRag allows users to interact with their curriculum in natural language.

The design philosophy emphasizes **local-first privacy**, **efficiency**, and **robustness**. No user data leaves the machine during the retrieval phase, and the system is designed to handle network instability and varied content formats.

2 System Architecture

The system is built in Rust, leveraging its memory safety and concurrency features. It follows a modular architecture where the core domain logic (RAG) is decoupled from the user interface (TUI) and data acquisition (Scraper).

2.1 High-Level Component Diagram

- **User Interface Layer**: Textual User Interface (TUI) built with `ratatui`.
- **Application Layer**: Orchestrates the sync pipeline, query handling, and state management.
- **Domain Layer**:
  - **RAG Service**: Embeddings, Vector Search, Document Management.
  - **Scraper Service**: UPV PoliformaT Automation.
- **Infrastructure Layer**:
  - **HNSW Index**: On-disk vector storage.
  - **LLM Client**: HTTP client for OpenRouter/LM Studio.
  - **Config Store**: JSON based persistence with encryption.

3 Detailed Component Analysis

3.1 Scraper Module (`src/scrapper`)

The scraper automation is powered by `headless_chrome`, which controls a local Chrome instance via the DevTools Protocol (CDP).

3.1.1 Authentication & Session Management

PoliformaT requires UPV Single Sign-On (SSO). PoliRag creates a robust session flow:

1. **Credential Resolution**: Checks (1) Cached Config, (2) Environment Variables (`POLIFORMAT_USER`).
2. **Headless Login**:
   - Navigates to the login portal.
   - Detects input fields dynamically (handling variable IDs like `#username` vs `input[name='dni']`).
   - Submits credentials and waits for the `#toolMenu` element to confirm success.
3. **Cookie Export**: Upon successful login, the scraper exports the browser cookies. These are shared with a `reqwest::Client` instance, enabling high-performance concurrent HTTP requests for subsequent operations that don't require JavaScript rendering.

3.1.2 Parallel Execution Strategy

While navigating subjects is sequential (to manage the browser's download behavior context safely), content extraction within a subject utilizes hybrid blocking/async tasks.

- **Subject Discovery**: Executes a JS snippet in the browser context to parse the DOM and extract all subject URLs, deduplicating them.
- **Download Management**: The scraper uses `Page.setDownloadBehavior` to redirect downloads to a per-subject directory. It monitors file system events (presence of `.crdownload` files) to ensure completion.
- **Enhanced Content Extraction**: Beyond PDF files, the scraper now automatically navigates to the "Guia Docent" and "Syllabus" pages to extract rich text descriptions and professor information, providing a more comprehensive context for the RAG system.

3.2 RAG Engine (`src/rag`)

The core of PoliRag is its vector search capabilities.

3.2.1 Embedding Model

PoliRag utilizes a dynamic embedding system powered by a custom `embed_model!` macro.

- **Dynamic Loading**: The macro allows for switching between different models (e.g., `google/embedding-gemma` or `nomic-embed-text`) based on configuration, without recompiling the entire logic.
- **Model**: Defaults to quantized GGUF models (e.g., `Q4_0` precision).
- **Engine**: `llama.cpp` bindings.
- **Hardware Acceleration**: On macOS, the system offloads all layers to Metal (GPU) for near-instant inference.
- **Chunking Strategy**: Uses overlapping semantic chunks (max 512 tokens) to preserve contextual coherence across document fragments.

3.2.2 Vector Storage (HNSW)

Instead of a linear scan, PoliRag uses Hierarchical Navigable Small World (HNSW) graphs via the `hnsw_rs` crate.

- **Algorithm**: Constructs a multi-layer graph where higher layers act as expressways for search, descending to lower layers for precision.
- **Parameters**: M=24 (max links per node), ef_construction=10000 (candidates during build).
- **Persistence**: The index is serialized to two files:
  - `.hnsw.graph`: The graph structure.
  - `.data`: The document content and metadata (managed via `bincode`).

3.3 TUI & Event Loop (`src/tui`)

The interface follows a reactor pattern.

3.3.1 Async Architecture

The main thread runs the UI render loop. Heavy operations (LLM generation, Sync) are offloaded to `tokio` tasks which communicate results back via `mpsc` channels.

- `tx_llm / rx_llm`: Stream tokens from the LLM.
- `tx_sync / rx_sync`: Stream log messages during the scraping process.

3.3.2 Rendering Pipeline

- **Markdown Caching**: Re-parsing markdown every frame is CPU-intensive. PoliRag implements a cell-level cache for rendered lines, enabling fluid 60FPS scrolling and near real-time text streaming even with complex formatting.
- **Throbber & Progress Status**: Animated indicators provide feedback for background processes like scraping or "thinking" states.
- **Thinking Blocks**: Native support for DeepSeek-style reasoning models. The UI detects `<think>` tags and allows users to toggle the visibility of the chain-of-thought, keeping the interface clean.
- **Intelligent Filename Detection**: The chat interface matches keywords against the local document index. If a user asks for a specific file (e.g., "summarize 'p1.pdf'"), the system retrieves the full content rather than just snippets.

4 Data Structures

4.1 Document

The foundational unit of the knowledge base.

```
#[derive(Serialize, Deserialize, Clone, Debug)]
pub struct Document {
    pub id: String,                  // Unique URI (e.g., subject_id/file_path)
    pub content: String,             // The raw text content
    pub embedding: Vec<f32>,         // High-dimensional vector
    pub metadata: HashMap<String, String>, // Key-values: type, user_id,
    timestamp
    pub user_id: String,             // Multi-user support field
}
```

4.2 Configuration

Stored in standard OS locations (e.g., `~/Library/Application Support/polirag/config.json` on macOS).

```
pub struct Config {
    pub last_model: Option<String>,
    pub cached_credentials: Option<EncryptedCredentials>, // Simple XOR +
    Base64
    pub llm_provider: LlmProvider, // LmStudio | OpenRouter
    // ...
}
```

5 Error Handling & Security

- **Credential Security**: Credentials are not stored in plain text. A simple XOR encryption with a hardcoded key obfuscates them on disk, preventing casual snooping (though not robust against determined attackers with binary access).
- **Graceful Degradation**: If the HNSW index is corrupted or missing, the system attempts to rebuild it or start fresh rather than crashing.
- **Network Resilience**: The scraper includes retry logic for finding DOM elements, handling the slow loading times of the university portal.