Fact Check Solution Using Ollama-based LLM Inference and FAISS Retrieval

Rahul Manikandan

April 13, 2025

1 Introduction

This document details the implementation and analysis of a fact-checking pipeline for verifying pharmaceutical marketing claims using supporting clinical literature. The system leverages document preprocessing, semantic retrieval using FAISS and SBERT embeddings, and inference via a local large language model (LLM) served through Ollama. The end-to-end architecture ensures claim verification is explainable, auditable, and memory-efficient — tailored for deployment even on limited hardware.

2 Data Preprocessing

The raw input comprises a set of clinical trial PDF documents. Each PDF is decomposed into three separate sources of information:

1. **Paragraph Text**: Extracted using PyMuPDF and stored as line-separated JSON.
2. **Tables**: Extracted using Camelot in "stream" mode, avoiding layout-sensitive failures and zero-area elements.
3. **Images and Figures**: Processed using PyMuPDF for image extraction and Tesseract OCR for text recognition.

Each piece of extracted content is tagged with metadata, such as page number, file name, and type (e.g., "paragraph", "table", or "figure"). The processed files are stored in a 'preprocessed/' directory as individual JSONL files.

3 Corpus and Vector Index Construction

To facilitate fast semantic search, the system converts extracted clinical content into a FAISS vector database. The following steps are followed:

3.1 Corpus Building

Using the ‘build$_c$*orpus*()‘*function*, *allrecordsfromthe*‘*preprocessed*‘*directoryareaggregatedintoaunifiedc*

3.2 Embedding Generation

Each textual unit (paragraph, table cell, or OCR result) is embedded using the `paraphrase-MiniLM-L3-v2` model from the SentenceTransformers library. Embeddings are normalized to support cosine similarity via dot product.

To avoid memory issues, embeddings are generated sequentially (one-by-one) and appended to a numpy array, which is later used to build the FAISS index.

3.3 FAISS Index Creation

FAISS is configured using an ‘IndexFlatIP‘, suitable for cosine similarity over normalized embeddings. The metadata (clinical records) is serialized using Python’s ‘pickle‘ module and saved alongside the index for future lookup.

4 Claim Retrieval and LLM-Based Inference

4.1 Retrieval Stage

Each claim from the input JSON file is encoded using the same SBERT model and compared to the FAISS index. Top-k candidate snippets are retrieved based on cosine similarity. These candidates form the context for downstream LLM inference.

4.2 LLM Inference with Ollama (Version 1)

The initial inference implementation uses the **Mistral** model served locally via Ollama. A single prompt is constructed per claim, containing:

- The claim text.
- Top-k candidate snippets with similarity scores and source information.

The prompt asks the model to return only those snippets that strongly support the claim. The selected evidence is saved in `ollama_results.json`, with each claim mapped to a list of ‘match_sources‘.

This version enables semantic filtering beyond similarity alone, letting the LLM decide what truly supports the claim.

4.3 LLM Inference with Reasoning (Version 2)

To improve transparency and auditability, a second inference method is introduced: `infer_matches_with_reason`. The updated prompt asks Mistral to evaluate *each candidate snippet individually*, returning:

- `document_name`
- `matching_text`
- `score`
- `supports` (Boolean)
- `reason` (LLM-generated explanation)

This structured reasoning allows users to filter results by logical correctness and inspect why certain snippets were accepted or rejected. The results are stored in `ollama2_results.json`.

5 Result Analysis

5.1 Version 1 (`ollama_results.json`)

This result set is concise and performant. It returns a list of supporting snippets selected by the LLM. However, it lacks:

- Explicit rejection of non-supporting evidence.
- Explanations or reasoning behind decisions.
- Any metadata for human auditing or score thresholding.

This makes Version 1 ideal for lightweight tasks but not well-suited for explainable pipelines or regulatory review.

5.2 Version 2 (`ollama2_results.json`)

This version significantly improves interpretability:

- Each snippet is evaluated with a `true/false` verdict and a textual justification.
- Reduces false positives: high similarity does not guarantee a “supports: true” flag.
- Enables structured downstream filtering, ranking, or UI rendering.

For instance, multiple snippets with high cosine scores were marked `supports: false` because they did not reference the claim’s specific topic (e.g., “HA antigen quantity per strain”).

Claims related to immune response, antigenic match, or recombinant platforms showed precise filtering with reasons pointing to terminology gaps or indirect references.

6 Limitations and Future Work: CoRAG and ToRAG

To further enhance reasoning, two iterative question-answering pipelines — **CoRAG** (Co-operative RAG) and **ToRAG** (Tree-of-Thought RAG) — are under development.

These methods generate follow-up questions to iteratively refine evidence collection and emulate human reasoning:

- **CoRAG**: Builds a chain of questions with early stopping once sufficient evidence is found.
- **ToRAG**: Branches into multiple sub-questions and selects the best response among them using an elimination strategy.

However, both methods require multiple LLM invocations per claim and per step. On local machines with limited memory (such as an M1 MacBook), the following issues were encountered:

- **Memory bottlenecks**: Mistral consumes significant RAM on each inference pass, leading to slowdowns or crashes after several iterations.
- **Cumulative load**: Tree-style iteration increases the number of LLM calls exponentially, exacerbating the memory pressure.

To resolve this, the plan is to port the entire inference pipeline (especially CoRAG/-ToRAG) to a cloud-based GPU runtime, possibly using Hugging Face endpoints or a hosted Ollama container. This will allow asynchronous batch inference and reduce latency during multi-turn evidence collection.