Technical Report: EB-1A RFE Risk Analyzer

Author: Anjolaiya Kabiawu **Date:** August 26, 2025

1. Problem Framing and Approach

The process of applying for an EB-1A "Extraordinary Ability" visa is fraught with subjectivity. A key pain point for legal practitioners and applicants is the risk of receiving a Request for Evidence (RFE), which can cause significant delays and uncertainty. Our approach was to leverage a Large Language Model (LLM) to build a tool that can "think like an adjudicator" to preemptively identify these weaknesses. This project evolved from a simple prompt-based system to a sophisticated, data-driven **Retrieval-Augmented Generation (RAG)** pipeline to ensure the feedback is not only intelligent but also grounded in real-world legal precedent.

2. Architecture and Tool Design

The system is designed with a two-phase architecture: an offline data preprocessing phase to build a knowledge base, and an online analysis phase to evaluate petitions.

Phase 1: Offline Knowledge Base Construction (`build_knowledge_base.py`)

The foundation of the system is a rich knowledge base derived from real-world legal outcomes. This phase is handled by a dedicated script that performs the following steps:

1. **Data Scraping:** It reads a list of URLs from `Master_file.txt` pointing to thousands of USCIS AAO decision documents.
2. **Text Extraction:** For each URL, it downloads the PDF in memory, extracts the full text using `PyMuPDF`, and creates a LangChain `Document` object, storing the source URL in the metadata.
3. **Chunking & Embedding:** The extracted documents are split into smaller, semantically coherent chunks. Each chunk is then converted into a numerical vector (embedding) using a sentence-transformer model (`all-MiniLM-L6-v2`), which runs locally.
4. **Vector Store Creation:** These embeddings are indexed and stored in a FAISS (Facebook AI Similarity Search) vector store, which is then saved to the local disk in a `faiss_index` directory. This offline process creates a highly efficient, searchable database of legal text.

Phase 2: Online RFE Analysis (`main.py`)

When a user wants to analyze a petition, the main application executes the following RAG-enhanced pipeline:

1. **Knowledge Base Loading:** The application starts by loading the pre-built `faiss_index` from disk into memory.
2. **Petition Ingestion:** The user's draft petition (*.docx, .pdf, or .txt*) is parsed and segmented into its constituent parts (e.g., Personal Statement, criterion-specific evidence).
3. **Initial Analysis (LLM Call #1):** Each petition segment is sent to the GPT-4o API with a prompt that instructs it to identify weaknesses and format them in a table. This step identifies the core issues.
4. **Retrieval-Augmentation (RAG):** For each weakness identified, the description of the weakness is used as a query to the FAISS vector store. The system retrieves the most relevant text chunks from the thousands of real AAO decisions.
5. **Enhanced Generation (LLM Call #2):** The original weakness, along with the retrieved legal context, is passed to the GPT-4o API with a new prompt. This prompt instructs the model to act as an expert legal assistant and generate a new, evidence-based suggestion that is grounded in the provided real-world examples.
6. **Report Generation**: The final, enhanced analysis is compiled into a professional `.docx` report.

3. Edge Cases or Limitations

While the prototype is effective, it has several known limitations:

- **Criterion Misidentification:** The initial analysis model can still occasionally misclassify a section's criterion due to semantic overlap.
- **Complex Document Formatting:** The current text extraction and segmentation logic assumes a simple, linear document structure. It would likely fail on documents with complex layouts like multiple columns, embedded tables, or extensive footnotes.
- **Quality of Scraped Data:** The effectiveness of the RAG system is entirely dependent on the quality and relevance of the documents in `Master_file.txt`. The file contains non-EB-1A cases which can introduce noise.
- **API Dependency & Cost:** The system relies on the OpenAI API for its reasoning capabilities, which involves operational costs, particularly during the initial knowledge base construction.

4. Future Extension Opportunities

This prototype serves as a strong foundation for a more robust and feature-rich tool. Key future extensions could include:

- **Knowledge Grounding with a Vector Database:** The scraping script could be enhanced with a classification model to automatically filter out irrelevant (non-EB-1A) cases before they are added to the knowledge base.
- **Fine-Tuning a Specialized Model:** The scraped data provides a perfect foundation for fine-tuning a specialized open-source model. This could reduce reliance on the GPT-4o API and create a more cost-effective and expert system.
- **Interactive Web Interface:** The tool could be embedded in a user-friendly web application (using Streamlit or Flask). An attorney could upload a document, and the identified weaknesses could be highlighted directly on the text in an interactive side-by-side view, allowing for real-time editing and re-analysis.
- **Interactive "Cited Sources”:** The final report could be enhanced to include direct quotes and links to the specific AAO decisions that the RAG system used to generate its suggestions, providing ultimate explainability.