Visa Eligibility Screening Agent using Retrieval-Augmented Generation

Author Name

January 2, 2026

Abstract

Visa eligibility assessment involves interpreting complex immigration policies that vary across countries, visa types, and applicant profiles. Manual interpretation is time-consuming and error-prone, while purely rule-based systems lack flexibility.

This project presents a Retrieval-Augmented Generation (RAG) based visa screening agent that grounds large language model outputs in official policy documents. The system uses semantic embeddings and vector similarity search to retrieve relevant policy sections and constrains the LLM response using retrieved evidence. A Streamlit-based interface enables structured input, country-aware UI theming, and explainable eligibility decisions.

1 Project Statement

Visa application rejections are frequently caused by misunderstanding eligibility criteria, incomplete documentation, and reliance on unofficial advice. Immigration rules are often expressed in legal language spanning hundreds of pages, making them inaccessible to average applicants.

Existing automation approaches typically rely on static decision trees or keyword-based search, which fail to capture semantic meaning and contextual dependencies. Large language models, while powerful, are unsuitable for direct use due to hallucination risks in legal domains.

Problem Definition

Design a visa screening assistant that provides:

- Accurate, policy-grounded eligibility assessment
- Explainable decisions
- Support for multiple countries and visa types
- A scalable architecture without retraining

Proposed Solution

The solution uses Retrieval-Augmented Generation (RAG), combining semantic document retrieval with controlled LLM reasoning.

2 System Architecture

The system is divided into three major components:

1. Document Embedding Pipeline
2. Retrieval-Augmented Generation Pipeline
3. Streamlit User Interface

User → Streamlit UI → RAG Pipeline → FAISS Vector Store → LLM → Eligibility Decision

This separation ensures modularity, maintainability, and scalability.

3 Document Embedding Pipeline

The embedding pipeline (`embedder.py`) prepares visa policy documents for semantic retrieval.

3.1 PDF Text Extraction

```
from pypdf import PdfReader

def extract_text(pdf_path):
    reader = PdfReader(pdf_path)
    text = ""
    for page in reader.pages:
        text += page.extract_text()
    return text
```

This function loads official policy PDFs and extracts raw text for downstream processing.

3.2 Text Chunking Strategy

```
def chunk_text(text, chunk_size=500, overlap=100):
    chunks = []
    for i in range(0, len(text), chunk_size - overlap):
        chunks.append(text[i:i+chunk_size])
    return chunks
```

Chunking enables fine-grained retrieval, while overlap preserves contextual continuity.

3.3 Embedding Generation

```
from sentence_transformers import SentenceTransformer

model = SentenceTransformer("all-MiniLM-L6-v2")
embeddings = model.encode(chunks)
```

MiniLM embeddings balance semantic accuracy and computational efficiency.

3.4 FAISS Index Creation

```
import faiss
import numpy as np
```

```
index = faiss.IndexFlatL2(embeddings.shape[1])
index.add(np.array(embeddings))
```

FAISS enables fast similarity search across large document collections.

4 Retrieval-Augmented Generation Pipeline

The RAG pipeline (rag$_p$*ipeline.py)dynamicallyretrievesrelevantpolicycontentandconstrainsLLMres*

4.1 Query Embedding and Retrieval

```python
def retrieve_chunks(query, index, model, chunks, k=5):
    query_embedding = model.encode([query])
    distances, indices = index.search(query_embedding, k)
    return [chunks[i] for i in indices[0]]
```

The user query is embedded and matched against the vector store to retrieve relevant policy sections.

4.2 Prompt Construction

```python
def build_prompt(profile, question, context):
    return f"""
You are a visa eligibility expert.

Context:
{context}

Applicant Profile:
{profile}

Question:
{question}

Provide:
- Eligibility decision
- Explanation
- Confidence score
"""
```

The prompt explicitly restricts the LLM to retrieved policy context.

4.3 LLM Invocation

```python
import google.generativeai as genai

def generate_answer(prompt):
    response = genai.generate_content(prompt)
    return response.text
```

Gemini is used to generate grounded responses, with fallback logic if needed.

4.4 Pipeline Orchestration

```
def run_rag_pipeline(profile_json, question):
    retrieved = retrieve_chunks(question, index, model, chunks)
    prompt = build_prompt(profile_json, question, retrieved)
    answer = generate_answer(prompt)
    return answer, retrieved, "gemini"
```

This function orchestrates retrieval and generation into a single callable pipeline.

5 Streamlit User Interface

The Streamlit application (app.py) serves as the front-end interface.

5.1 User Input Collection

```
dest = st.selectbox("Destination country", countries)
visa_type = st.selectbox("Visa category", visa_types)
question = st.text_area("Ask a visa-related question")
```

Structured inputs ensure consistent backend processing.

5.2 Profile Construction

```
profile_payload = {
    "age": age,
    "nationality": nationality,
    "destination": dest,
    "visa_type": visa_type
}
profile_json = json.dumps(profile_payload)
```

Applicant data is serialized into JSON for pipeline compatibility.

5.3 Pipeline Invocation

```
if st.button("Run Eligibility Check"):
    answer, retrieved, used_api = run_rag_pipeline(
        profile_json, question
    )
```

The UI triggers the backend RAG pipeline.

5.4 Results Visualization

```
st.markdown(
    f"<div class='card'>{answer}</div>",
    unsafe_allow_html=True
)
```

The output is displayed using animated, country-themed UI cards.

6 Design Decisions and Justification

- `RAG over Fine-Tuning:  Enables rapid policy updates`
- `FAISS: High-performance vector retrieval`
- `MiniLM: Lightweight yet accurate embeddings`
- `Streamlit:  Rapid UI development with flexibility`

7 Limitations and Future Enhancements

- OCR support for scanned documents
- Multilingual policy handling
- Hybrid rule-based validation
- Deployment with authentication and logging

8 Conclusion

This project successfully demonstrates the design and implementation of an intelligent Visa Eligibility Assistant using a Retrieval-Augmented Generation (RAG) architecture. The system integrates structured policy documents with semantic search and large language models to provide accurate, explainable, and context-aware visa eligibility guidance.

By combining FAISS-based vector retrieval, sentence-transformer embeddings, and a modern Streamlit-based user interface, the solution ensures both high-quality reasoning and an intuitive user experience. The separation of frontend presentation (app.py) from backend intelligence (rag$_p$*ipeline.py*).

The application enhances transparency by presenting retrieved policy evidence alongside AI-generated explanations and confidence indicators. This approach reduces hallucination risk and increases user trust, making the system suitable for decision-support use cases rather than black-box recommendations.

Overall, the project highlights the practical application of RAG systems in real-world policy domains, demonstrating how AI can assist users in complex eligibility evaluations while remaining grounded in authoritative sources. With further enhancements such as real-time policy updates, multilingual support, and expanded jurisdiction coverage, the system can evolve into a robust and production-ready visa advisory platform.