SWIFTVISA AI-BASED VISA ELIGIBILITY SCREENING AGENT

CONTENTS

S. No	Title	Page No
1	PROJECT STATEMENT	1
2	PIPELINE MODEL	2
2.1	DATASET PREPARATION	3
2.2	TYPES OF EMBEDDING MODELS	10
2.3	EMBEDDING GENERATION	11
2.4	TRANSFORMER MODELS	13
2.5	FAISS VS CHROMA VECTOR DATABASE	14
2.6	RETRIEVER TYPE USED	15
2.7	LLM AUGMENTED WITH RETRIEVER	17
2.8	STREAMLIT APPLICATION	19
3	CONCLUSION	35

CHAPTER 1

PROJECT STATEMENT

The visa application process is more complex, requiring individuals to navigate through extensive documentation and varying eligibility criteria. Swift Visa is an AI-based Visa Eligibility Screening Agent designed to help users understand visa requirements, eligibility criteria, and application conditions for multiple countries such as the USA, UK, Canada, Ireland, and the Schengen region. The main problem addressed by this project is that visa information is scattered across lengthy documents, which are difficult for applicants to understand. Manual consultation can lead to errors, misunderstandings, and visa rejections. Swift Visa solves this problem by using Retrieval-Augmented Generation (RAG). Instead of relying on the language model's memory, the system retrieves relevant official visa policy documents and then generates answers strictly based on those documents. Its goal is to reduce confusion, save time, and improve accessibility in AI responses.

CHAPTER 2

PIPELINE MODEL

The Swift Visa system follows a Retrieval-Augmented Generation (RAG) pipeline. RAG combines two concepts: Information Retrieval and Large Language Models (LLMs). Retrieval-Augmented Generation (RAG) pipeline consists of data ingestion, chunking, embedding, vector storage, retrieval, and LLM-based response generation. The retrieval part works like a smart search engine—it looks through a database or collection of documents to find the most relevant facts. The large language model part then takes those facts and explains them in natural, easy-to-understand language. By joining these two abilities, RAG forces the AI to read documents first, then answer. Thus, Information Retrieval finds the right documents where as LLM generate a human like answer.

In this project, visa documents are collected first and processed into machine-readable formats. These documents are then converted into vector embeddings and stored in a vector database. When a user asks a question, the system retrieves the most relevant document chunks using semantic similarity search. Finally, the retrieved information is provided as context to a Large Language Model (LLM), which generates a grounded response. This pipeline ensures that answers are based on actual visa rules rather than assumptions. Finally, a GUI is built using streamlit app for smooth processing.

2.1 DATASET PREPARATION

Dataset preparation is a crucial step in the Swift Visa project because the accuracy of the system depends entirely on the quality of visa documents used. In this project, official visa requirement information for different regions such as the USA, UK, Canada, Ireland, and the Schengen area was collected from trusted sources. These documents were originally available in PDF or text format, which are suitable for human reading but not efficient for machine processing. Therefore, the collected were carefully cleaned and organized.

Here the dataset is prepared separately for each country to ensure clarity, accuracy, and easy retrieval of visa information. Each country's visa rules are stored in an individual JSON file, which acts as a structured and machine-readable data source. Each document contains detailed information such as visa types, eligibility conditions, required documents, fees, processing rules, and special notes. To make this data usable for the AI system, the unstructured text was converted into structured JSON files. The JSON format is used because it organizes information into key–value pairs and sections, making it easy for machines to process, search, and update. It makes easier for the system to identify and process different visa categories separately.

The various JSON files used for this project are described below.

- **canada.json**

The canada.json file contains structured information about Canada's visa and immigration system as managed by Immigration, Refugees and Citizenship Canada (IRCC). It covers both temporary visas such as visitor visas, study permits, work permits, and super visas, as well as permanent residence pathways like Express Entry, Provincial Nominee Programs, and family sponsorship. The file includes important details such as visa fees, biometric costs, proof of funds, required documents, and processing timelines. By organizing this information in a structured JSON format, the system can accurately retrieve Canada-specific visa rules and answer user queries related to Canadian visas. Sample, .Json file for Canada is shown below'


```
{
 "visa_type": "Canada",
 "last_updated": "November 2025",
 "overview": {
  "categories": [
   "Temporary resident visas (visitor, study, work)",
   "Permanent Residence options"
  ],
  "managing_authority": "IRCC (Immigration, Refugees and Citizenship Canada)"
 },
 "key_fees_2025": {
  "right_of_permanent_residence": "CA$575",
```

```
      "biometric_fee": "CA$85 per person"
    },
    "visa_permit_types": {
      "visitor_trv": {
        "purpose": ["Tourism", "Business visits", "Family visits", "Transit"],
        "fee": "CA$100",
        "entries": "Single or multiple entry",
        "required_documents": ["Passport", "Proof of financial means", "Travel plan", "Ties to home country"]
      },
      "eta": {
        "purpose": "For visa-exempt nationals traveling by air",
        "fee": "CA$7",
        "validity": "Up to 5 years or until passport expiry",
        "limitations": "Does not allow work or study"
      },
      "study_permit": {
        "fee": "CA$150",
        "financial_requirements": "Proof of sufficient funds for tuition + living costs",
        "required_documents": ["Acceptance letter from DLI", "Proof of funds", "Biometric info"]
      },
      "work_permits": {
        "types": ["Employer-specific", "Open Work Permit", "International Experience Canada"],
        "fees": {
```

```
            "employer_specific": "CA$155",
            "open_work_permit": "CA$100"
          },
          "processing_times": {
            "in_canada": "~120 days",
            "outside_canada": "~60 days (varies)"
          }
        },
        "super_visa": {
          "purpose": "For parents/grandparents of Canadian citizens/PRs",
          "fee": "CA$100",
          "stay_length": "Up to 2 years per visit",
          "requirements": ["Invitation from child/grandchild", "Medical insurance",
"Medical exam"]
        },
        "permanent_residence": {
          "streams": ["Economic (Express Entry)", "Provincial Nominee
Programs", "Family Sponsorship"],
          "fees": {
            "rprf": "CA$575",
            "dependent_child": "CA$175"
          }
        }
      }
```


- **usa.json**

The usa.json file stores detailed visa information for the United States, covering both non-immigrant (temporary) and immigrant (permanent) visa categories. It includes visitor visas (B1/B2), student visas (F and M), exchange visas (J), work visas (H, L, O, P, etc.), and immigrant visas such as family-based and employment-based green card options. The file also captures key rules like mandatory interviews, visa integrity fees, passport validity, required application forms, and compliance conditions. This structured data enables the system to provide precise answers for U.S. visa eligibility and procedures.

- **uk.json**

The uk.json file represents visa rules and requirements for the United Kingdom. It includes visitor visas, student visas, skilled worker visas, post-study (graduate) routes, family and settlement visas, and Indefinite Leave to Remain (ILR). The file also documents recent fee increases, Certificate of Sponsorship (CoS) requirements, Immigration Health Surcharge (IHS), and priority processing services. By storing this information in JSON format, the Swift Visa system can efficiently retrieve UK-specific visa details and ensure users receive accurate and updated responses.

- **ireland.json**

The ireland.json file contains visa information related to Ireland, covering short-stay visas (Type C), long-stay visas (Type D), student visas, work permits, and re-entry visas. It includes financial requirements, English language test conditions, biometric rules, visa fees, and post-arrival registration details such as Irish Residence Permit (IRP). This structured dataset allows the system to answer Ireland-specific visa queries clearly and based on official requirements.

- **schengen.json**

The schengen.json file stores visa rules applicable to the Schengen Area, which includes multiple European countries under a common visa policy. It covers short-stay Schengen visas (Type C) and long-stay national visas (Type D), along with rules

like the 90/180-day stay limit, mandatory travel insurance, biometric requirements, visa fees, and processing timelines. Because Schengen rules apply across many countries, this file helps the system provide consistent and reliable answers for Schengen visa-related questions.

Once the JSON files were created, these files were then passed through a chunking process, where large JSON content was split into smaller, manageable text chunks. Chunking is necessary because large documents cannot be processed efficiently by embedding models or language models at once. By dividing each country's visa information into logically sized chunks, the system ensures accurate embedding, fast retrieval, and precise responses. Overall, this dataset preparation process transforms complex, lengthy visa documents into clean, structured, and AI-ready data that enables reliable retrieval and question answering in the Swift Visa system.

Pseudocode for chunking

```
import os
import json
def chunk_json_files():
    # Use the actual folder name where your JSON files are
    input_folder = "C:\\Users\\akhil\\Downloads\\SWIFT VISA"
      # Create output folder
    output_folder = os.path.join(input_folder, "chunked_output")
    os.makedirs(output_folder, exist_ok=True)
     # Process each JSON file
    for filename in os.listdir(input_folder):
       if filename.endswith(".json"):
```

```
        input_path = os.path.join(input_folder, filename)
        output_path = os.path.join(output_folder, filename.replace(".json", "_chunks.txt"))
         # Read JSON file
        with open(input_path, 'r') as f:
            data = json.load(f)
        # Convert to string and split into chunks
        json_str = json.dumps(data, indent=2)
        chunks = []
        current_chunk = ""
        for line in json_str.split('\n'):
            if len(current_chunk + line) > 500:  # ~500 chars per chunk
                chunks.append(current_chunk)
                current_chunk = line + '\n'
            else:
                current_chunk += line + '\n'
        if current_chunk:
            chunks.append(current_chunk)
         # Save chunks to file
        with open(output_path, 'w') as f:
            for i, chunk in enumerate(chunks, 1):
                f.write(f"--- CHUNK {i} ---\n")
                f.write(chunk)
                f.write("\n\n")
            print(f"Created {output_path} with {len(chunks)} chunks")
```

```
# Run the function
chunk_json_files()
```

2.2 TYPES OF EMBEDDING MODELS

Embedding models convert text into numerical vectors so that machines can understand semantic meaning. Different types of embedding models are used based on the level of context and accuracy required.Word Embeddings represent individual words as vectors. Examples include Word2Vec and GloVe. These embeddings capture basic word meaning but fail to understand context, as the same word always has the same vector.

Sentence Embeddings represent entire sentences or paragraphs as a single vector. They capture the overall meaning of text and are more suitable for semantic search and retrieval tasks. Sentence Transformers are commonly used for this purpose.

Document Embeddings extend sentence embeddings to longer documents. They are useful when entire documents need to be compared instead of small text chunks.

Contextual Embeddings are generated using transformer-based models such as BERT. These embeddings understand context, meaning that the same word can have different embeddings depending on its usage in a sentence.

In the Swift Visa project, sentence-level contextual embeddings are used because visa rules are long and complex. The chosen model, all-MiniLM-L6-v2,

provides efficient and accurate semantic embeddings, making it suitable for Retrieval-Augmented Generation (RAG) systems.

2.3 EMBEDDING GENERATION

Embedding is the process of converting text into numerical vectors so that a computer can understand the meaning of the text instead of just matching keywords. There are different types of embeddings used in AI systems, depending on the kind of data being processed. Word embeddings convert individual words into vectors, but they do not understand full sentence meaning or context very well. Sentence embeddings represent entire sentences or paragraphs as vectors, capturing overall meaning rather than just individual words. Document embeddings extend this idea further by representing long texts or documents as vectors. There are also contextual embeddings produced by transformer models (like BERT-based models), which understand context and meaning more accurately because the same word can have different meanings in different sentences. In Retrieval-Augmented Generation (RAG) systems, sentence or document embeddings are preferred because they allow semantic comparison between user queries and document content.

In the Swift Visa project, the Sentence Transformer model all-MiniLM-L6-v2 is used for generating embeddings. The all-MiniLM-L6-v2 model was chosen because it is fast, lightweight, and accurate for semantic search tasks. Compared to larger models like BERT or paid API-based embeddings, this model works efficiently on local systems and provides good performance with less computation. Therefore, it is well suited for retrieval-based systems like the RAG pipeline used in this project.

The embedding.py file is responsible for converting chunked visa document text into numerical embeddings that can be used for semantic search. The script loads the pre-trained Sentence Transformer model all-MiniLM-L6-v2 and reads all chunked text files from the chunked output directory. For each file, it extracts individual text chunks, removes empty lines, and passes the chunks to the embedding model to generate vector representations. These vectors capture the semantic meaning of the visa information. The script then saves both the original text chunks and their corresponding embeddings into a structured JSON file inside the embeddings folder. This output serves as the input for the FAISS vector database, enabling fast and accurate retrieval of visa-related information during user queries.

Pseudocode

```
import os, json
from sentence_transformers import SentenceTransformer
model = SentenceTransformer("all-MiniLM-L6-v2")
os.makedirs("embeddings", exist_ok=True)
for fname in os.listdir("chunked_output"):
   if fname.endswith(".txt"):
      with open(os.path.join("chunked_output", fname), "r", encoding="utf-8") as f:
         chunks = [line.strip() for line in f if line.strip()]
      embedding = model.encode(chunks).tolist()
      with open(os.path.join("embeddings", f"{os.path.splitext(fname)[0]}_embedding.json"), "w", encoding="utf-8") as f:
```

```
        json.dump({"chunks": chunks, "embedding": embedding}, f,
indent=2)
```

2.4 TRANSFORMER MODELS

Transformer models are a class of deep learning models designed to understand relationships between words in a sentence using an attention mechanism. Unlike traditional models that process text sequentially, transformers process all words at the same time, making them faster and more accurate.

The key component of transformers is self-attention, which allows the model to focus on important words in a sentence while understanding context. This helps the model capture long-range dependencies and complex language patterns.

Popular transformer models include BERT, GPT, T5, and MiniLM. These models are widely used for tasks such as question answering, text summarization, translation, and semantic search.

In the Swift Visa project, transformer-based models are used in two places. First, a transformer-based sentence embedding model (all-MiniLM-L6-v2) converts visa documents into embeddings. Second, a transformer-based Large Language Model (Gemini) generates human-readable answers based on retrieved visa information.
Transformers are chosen because they provide high accuracy, contextual understanding, and scalability, making them ideal for real-world AI applications like visa eligibility screening.

2.5 FAISS VS CHROMA VECTOR DATABASE

Vector databases are used to store embeddings and perform similarity search in Retrieval-Augmented Generation systems. Two commonly used vector databases are FAISS and Chroma.

FAISS (Facebook AI Similarity Search) is a high-performance vector search library developed by Meta. It is optimized for speed and efficiency and is suitable for large-scale similarity search. FAISS works well for local deployments and provides fast nearest-neighbor search using mathematical distance metrics such as L2 distance or cosine similarity.

Chroma is a modern vector database designed specifically for LLM-based applications. It provides built-in support for metadata storage, document tracking, and easy integration with frameworks like LangChain. Chroma is more user-friendly but slightly slower compared to FAISS for very large datasets.

In the Swift Visa project, FAISS is chosen because it offers faster retrieval, lower memory usage, and better control over indexing. Since the project focuses on local document retrieval and high-speed semantic search, FAISS is more suitable than Chroma.

2.6 RETRIEVER TYPE USED

A retriever is a component in an AI system that is responsible for finding and selecting relevant information from a large collection of documents. Instead of generating answers directly, the retriever first searches the stored data and brings back the most relevant pieces of information related to a user's question. Retrievers are used to anchor responses in real, current data from sources like PDFs. Common types include dense retrievers, which use vector embeddings for semantic matching and excel at understanding natural language; sparse retrievers like BM25 for fast keyword searches; and hybrid ones that combine both for optimal speed and accuracy.

A retriever in RAG systems like SWIFT VISA is a search component that scans a database of visa documents and pulls the most relevant chunks such as eligibility rules or requirement based on the user's question, supplying precise context to the AI model for generating trustworthy answers. This project uses a vector similarity retriever. All visa document chunks are converted into numerical embeddings and stored in a FAISS vector database. When a user asks a question, the question is also converted into an embedding and compared with the stored document embeddings using similarity search (such as cosine similarity or inner product). FAISS then retrieves the top-k most similar document chunks based on meaning, not just keywords. This type of retriever is chosen because it provides fast, accurate, and semantic retrieval, which is essential for the RAG pipeline used in the project. faiss_indexer file is used to create a FAISS vector database from the embeddings. It first reads all embedding JSON files from the embeddings folder and

extracts the numerical embedding vectors from them. These vectors are converted into a NumPy array and saved as a .npy file for reuse. Then, a FAISS index is created using these vectors, which allows fast similarity search based on vector distance. The FAISS index stores all visa document embeddings in an optimized form so that when a user asks a question, the system can quickly retrieve the most relevant visa information based on meaning. This file therefore prepares the backend retrieval system that enables semantic search in the Swift Visa project.

```
import os, json, numpy as np, faiss
BASE = r"C:\Users\Akhil\Downloads\SWIFT VISA"
EMB = os.path.join(BASE, "embeddings")
OUT = os.path.join(BASE, "faiss_output")
IDX = os.path.join(OUT, "faiss.index")
NPY = os.path.join(OUT, "vectors.npy")
os.makedirs(EMB, exist_ok=True)
os.makedirs(OUT, exist_ok=True)
vecs = []
for f in os.listdir(EMB):
    if f.endswith(".json"):
        with open(os.path.join(EMB, f), "r", encoding="utf-8") as fp:
            data = json.load(fp)
        if isinstance(data, dict):
            data = [data]
        for item in data:
            if "embedding" in item:
                v = np.array(item["embedding"], dtype="float32")
                vecs.append(v)
```

```
if not vecs:
    print(" No embeddings found.")
    exit()
arr = np.vstack(vecs)
np.save(NPY, arr)
index = faiss.IndexFlatL2(arr.shape[1])
index.add(arr)
faiss.write_index(index, IDX)
print(" FAISS index created successfully")
print(" Vectors saved at:", NPY)
print(" Index saved at:", IDX)
```

2.7 LLM AUGMENTED WITH RETRIEVER

An LLM (Large Language Model) is an artificial intelligence model that can understand and generate human-like text. It is trained on a very large amount of text data, such as books, articles, and websites. Because of this training, an LLM can answer questions, explain topics, summarize information, and generate meaningful responses. In the Swift Visa project, the LLM takes the user's question along with retrieved visa document information and produces a clear and readable answer. An LLM is used in the Swift Visa project to generate clear and human-readable answers to visa-related questions. Visa rules are complex and written in long official documents, which are difficult for users to understand. The LLM helps by converting the retrieved visa information into simple, well-structured responses. However, the LLM does not work alone; it is combined with a retriever so that it uses only official visa documents as input. This ensures that the answers are accurate, reliable, and

easy to understand. Therefore, the LLM is used to improve answer quality, readability, and user experience in the Swift Visa system.

Pseudocode

```
import os, json, numpy as np, faiss
BASE = r"C:\Users\Akhil\Downloads\SWIFT VISA"
EMB = os.path.join(BASE, "embeddings")
OUT = os.path.join(BASE, "faiss_output")
IDX = os.path.join(OUT, "faiss.index")
NPY = os.path.join(OUT, "vectors.npy")
os.makedirs(EMB, exist_ok=True)
os.makedirs(OUT, exist_ok=True)
vecs = []
for f in os.listdir(EMB):
    if f.endswith(".json"):
        with open(os.path.join(EMB, f), "r", encoding="utf-8") as fp:
            data = json.load(fp)
        if isinstance(data, dict):
            data = [data]
        for item in data:
            if "embedding" in item:
                v = np.array(item["embedding"], dtype="float32")
                vecs.append(v)
if not vecs:
    print(" No embeddings found.")
    exit()
arr = np.vstack(vecs)
```

```
np.save(NPY, arr)
index = faiss.IndexFlatL2(arr.shape[1])
index.add(arr)
faiss.write_index(index, IDX)
print(" FAISS index created successfully")
print(" Vectors saved at:", NPY)
print(" Index saved at:", IDX)
```

2.8 STREAMLIT APPLICATION

Streamlit is an open-source Python library used to build interactive web applications with minimal Python code. It quickly turns Python scripts into shareable, visually appealing apps without needing to know web technologies like HTML, CSS, or JavaScript.

Streamlit used in SWIFT VISA, handles user inputs like country selection and eligibility forms seamlessly, displays AI-generated visa answers effortlessly, and deploys easily for demos—ideal for this RAG-based app without needing web dev skills. The app2.py file is the main application file of the SWIFT VISA project. It creates a Streamlit-based web application that allows users to ask visa-related questions and receive intelligent answers using a Retrieval-Augmented Generation (RAG) approach. This file connects all parts of the project such as the dataset, retriever, embeddings, and the large language model into one working system. When the application starts, it first configures the Gemini large language model using an API key and sets up required file paths such as retrieved visa chunks, stored documents, background image, and a log file. The Streamlit interface is then created,

where the user selects a destination country and enters a visa-related question. Based on the question, the system detects the visa type (student, work, tourist, business) and checks whether eligibility details are required. If needed, the application displays a form to collect applicant details like education, work experience, funds, and travel history. After the user submits the question (and details, if applicable), the application loads relevant visa information from previously retrieved document chunks or from all stored visa documents. This acts as the retrieved context in the RAG pipeline. To improve accuracy and freshness, the system may also fetch a short summary from Wikipedia. All this information is combined and passed to the Gemini language model along with a carefully designed prompt. The language model then generates a clear, personalized visa answer or eligibility assessment based only on the provided context. Finally, the generated answer is displayed on the Streamlit interface and also saved in a log file for future reference. In this way, app2.py file deals with handling user interaction, retrieval of visa data, LLM-based answer generation, and result presentation, making the entire SWIFT VISA system functional and user-friendly.

Pseudocode

```
import os
import base64
import streamlit as st
import google.generativeai as genai
import wikipedia

# ---------- Gemini config ----------
```

```
genai.configure(api_key=os.getenv("GEMINI_API_KEY"))
MODEL = genai.GenerativeModel("models/gemini-2.5-flash")  # [web:9]


# ---------- Paths ----------
BASE = r"C:\Users\akhil\Downloads\SWIFT VISA"
RETR = os.path.join(BASE, "retrieved_chunks.txt")
EMBED_DIR = os.path.join(BASE, "chunked_output")
BG_IMAGE_PATH = os.path.join(BASE, "seven_wonders.jpg")    # background image
LOG_FILE = os.path.join(BASE, "qa_log.txt")              # Q&A log file
# ---------- Background helpers ----------
def set_background(image_path: str):
    """Set full-page background using a local image (Seven Wonders collage)."""  # [web:13][web:17]
    if not os.path.exists(image_path):
        return
    with open(image_path, "rb") as img:
        encoded = base64.b64encode(img.read()).decode()
    css = f"""
    <style>
    [data-testid="stAppViewContainer"] {{
        background-image: url("{encoded}");
        background-size: cover;
        background-position: center;
        background-repeat: no-repeat;
```

```
}}
[data-testid="stHeader"] {{
    background: rgba(0,0,0,0.4);
    backdrop-filter: blur(6px);
}}
.block-container {{
    background: rgba(255,255,255,0.92);
    padding: 2rem;
    border-radius: 1.5rem;
    box-shadow: 0 8px 32px rgba(0,0,0,0.4);
}}
.stButton>button {{
    background: linear-gradient(90deg, #667eea 0%, #764ba2 100%);
    color: white;
    border: none;
    border-radius: 0.5rem;
    padding: 0.5rem 2rem;
    font-weight: 600;
}}
.stButton>button:hover {{
    transform: translateY(-2px);
    box-shadow: 0 4px 12px rgba(102, 126, 234, 0.4);
}}
</style>
```

```
    """
    st.markdown(css, unsafe_allow_html=True)


# ---------- RAG context loaders ----------
def load_retrieved():
    if os.path.exists(RETR):
        return open(RETR, encoding="utf-8").read().strip()
    return ""


def load_all_chunks():
    text = ""
    if not os.path.isdir(EMBED_DIR):
        return ""
    for folder in os.listdir(EMBED_DIR):
        fpath = os.path.join(EMBED_DIR, folder)
        if not os.path.isdir(fpath):
            continue
        for fname in os.listdir(fpath):
            if fname.endswith(".txt"):
                text += open(os.path.join(fpath, fname), encoding="utf-8").read() + "\n"
    return text.strip()
def search_external(question, country):
```

```
    """Short, up-to-date summary from Wikipedia to complement local chunks."""
    try:
        query = f"{country} visa {question} latest"
        summary = wikipedia.summary(query, sentences=4)  # [web:11]
        return summary
    except Exception:
        return None
# ---------- Intent & visa type ----------
def detect_visa_type(question: str):
    q = question.lower()
    if any(w in q for w in ["student", "study", "university", "college", "education",
"ms", "mba", "phd"]):
        return "student"
    if any(w in q for w in ["work", "job", "employment", "h1b", "skilled", "worker"]):
        return "work"
    if any(w in q for w in ["business", "entrepreneur", "invest", "startup"]):
        return "business"
    if any(w in q for w in ["tourist", "visit", "travel", "tourism", "vacation"]):
        return "tourist"
    return None
def is_eligibility_question(q: str) -> bool:
    q = q.lower()
    phrases = [
        "am i eligible",
        "am i eligable",
```

```
        "eligible for",
        "eligibility for",
        "can i get this visa",
        "will i get this visa",
        "do i qualify",
    ]
    return any(p in q for p in phrases)
# ---------- LLM call ----------
def ask_model(question, context, country, user_details=None):
    details_text = ""
    if user_details:
        details_lines = [f"- {k}: {v}" for k, v in user_details.items() if str(v).strip() !=
""]
        if details_lines:
            details_text = "\n\nApplicant Details:\n" + "\n".join(details_lines)
    prompt = f"""
You are an expert visa consultant. Use the context and, only when needed, general
up-to-date visa rules.
Always answer the user's exact question. Do not change the topic.
Country: {country}
Context (visa rules, fees, eligibility, etc. from internal visa docs and public sources):
\"\"\"{context}\"\"\"
{details_text}
Question:
\"\"\"{question}\"\"\"
```

```
Instructions:
- Answer clearly and concisely.
- If the exact information (for example, fee or document list) is not in the context or may be outdated,
  say that it can vary and advise the user to confirm on the official visa website, then give a general explanation.
- If applicant details are provided and the user is asking about eligibility, you must:
  1) State if the person is Likely eligible / Possibly eligible / Unlikely to be eligible.
  2) Give 2–3 short reasons.
  3) End the answer with a line: "Confidence: XX%" where XX is a number from 0 to 100.
Return only the final answer for the user.
"""
    resp = MODEL.generate_content(prompt)
    return resp.text.strip()
# ---------- Logging helper ----------
def save_qa_to_file(question, answer):
    """
    Append question, answer, and confidence line (if present) to qa_log.txt. [web:29]
    """
    confidence_line = ""
    for line in reversed(answer.splitlines()):
        if line.strip().lower().startswith("confidence:"):
            confidence_line = line.strip()
            break
```

```
    with open(LOG_FILE, "a", encoding="utf-8") as f:
        f.write("----\n")
        f.write(f"Q: {question}\n\n")
        f.write("A:\n")
        f.write(answer.strip() + "\n")
        if confidence_line:
            f.write(confidence_line + "\n")
        f.write("----\n\n")
# ---------- Streamlit config ----------
st.set_page_config(page_title="SWIFT VISA Assistant", layout="wide")
set_background(BG_IMAGE_PATH)
# Session state
if "show_form" not in st.session_state:
    st.session_state.show_form = False
if "submitted" not in st.session_state:
    st.session_state.submitted = False
if "user_details" not in st.session_state:
    st.session_state.user_details = {}
if "visa_type" not in st.session_state:
    st.session_state.visa_type = None

# ---------- UI ----------
st.title("🌍 SWIFT VISA — Smart Visa Assistant")
st.markdown("### Get instant, personalized answers to your visa questions")
```

```
countries = ["United States", "United Kingdom", "Canada", "Ireland", "Schengen / Europe"]
country = st.selectbox("🌐 Select destination country:", countries)
question = st.text_input(
    "💬 Ask your visa question:",
    placeholder="e.g., What are the requirements for studying MS in USA?",
)
if question:
    visa_type = detect_visa_type(question)
    st.session_state.visa_type = visa_type
    needs_form = is_eligibility_question(question) or (visa_type is not None)
    if needs_form:
        st.session_state.show_form = True
# ---------- Details form ----------
user_details = {}
if st.session_state.show_form and question:
    visa_type = st.session_state.visa_type or "general"
    st.info("📋 Please fill your details so that eligibility and risk can be assessed more accurately.")
    st.markdown("---")
    st.subheader(f"📝 {visa_type.upper()} Visa Applicant Details")
    # Common fields
    col1, col2 = st.columns(2)
```

```
    with col1:
        user_details["Full Name"] = st.text_input("Full Name")
        user_details["Date of Birth (DD/MM/YYYY)"] = st.text_input("Date of Birth")
        user_details["Passport Validity (months remaining)"] = st.text_input("Passport Validity (months)")
        user_details["Country of Citizenship"] = st.text_input("Country of Citizenship")
    with col2:
        user_details["Current City"] = st.text_input("Current City")
        user_details["Financial Capacity (approx. funds in USD)"] = st.text_input("Available Funds (USD)")
        user_details["Previous International Travel"] = st.selectbox(
            "Have you traveled abroad before?", ["Select", "Yes", "No"]
        )
        user_details["Any Visa Refusals Before"] = st.selectbox(
            "Any previous visa refusals?", ["Select", "Yes", "No"]
        )
    # Visa-type specific
    if visa_type == "student":
        st.markdown("#### Education details")
        c1, c2 = st.columns(2)
        with c1:
            user_details["Previous University"] = st.text_input("Previous University/College")
```

```
        user_details["Highest Qualification"] = st.text_input("Highest Qualification")
        user_details["Intended Degree"] = st.selectbox(
            "Intended Degree", ["Select", "Bachelor's", "Master's", "PhD", "Diploma"]
        )
    with c2:
        user_details["Overall CGPA/Percentage"] = st.text_input("Overall CGPA or Percentage")
        user_details["English Test Score (IELTS/TOEFL/etc)"] = st.text_input("English Test Score")
        user_details["Can Show Financial Proof"] = st.selectbox(
            "Can you show sufficient financial proof?", ["Select", "Yes", "No", "Partial"]
        )

elif visa_type == "work":
    st.markdown("#### Work details")
    c1, c2 = st.columns(2)
    with c1:
        user_details["Current/Previous Job Title"] = st.text_input("Current/Previous Job Title")
        user_details["Years of Work Experience"] = st.number_input(
            "Years of Work Experience", min_value=0, max_value=50, step=1
        )
    with c2:
```

```
        user_details["Highest Education"] = st.selectbox(
            "Highest Education", ["Select", "High School", "Bachelor's", "Master's", "PhD"]
        )
        user_details["Job Offer in Destination Country"] = st.selectbox(
            "Do you already have a job offer?", ["Select", "Yes", "No"]
        )
    elif visa_type == "business":
        st.markdown("#### Business / investment details")
        c1, c2 = st.columns(2)
        with c1:
            user_details["Type of Business"] = st.text_input("Type of Business")
            user_details["Planned Investment (USD)"] = st.text_input("Planned Investment (USD)")
        with c2:
            user_details["Business Experience (years)"] = st.number_input(
                "Years in Business", min_value=0, max_value=50, step=1
            )
            user_details["Business Plan Ready"] = st.selectbox(
                "Do you have a structured business plan?", ["Select", "Yes", "No"]
            )

    elif visa_type == "tourist":
        st.markdown("#### Trip details")
        c1, c2 = st.columns(2)
```

```
        with c1:
            user_details["Purpose of Visit"] = st.text_input("Purpose of Visit (tourism/family/etc.)")
            user_details["Intended Stay Duration (days)"] = st.text_input("Intended Stay Duration (days)")
        with c2:
            user_details["Employment Status"] = st.selectbox(
                "Current Employment Status",
                ["Select", "Employed", "Self-Employed", "Retired", "Student", "Unemployed"],
            )
            user_details["Return Ticket Booked"] = st.selectbox(
                "Return ticket booked?", ["Select", "Yes", "No", "Not yet"]
            )
    st.markdown("")
    c1, c2, c3 = st.columns([1, 1, 1])
    with c2:
        if st.button("🚀 Submit & Get Assessment", use_container_width=True):
            if user_details.get("Full Name", "").strip():
                st.session_state.user_details = user_details
                st.session_state.submitted = True
                st.rerun()
            else:
                st.error("Please enter at least your Full Name before submitting.")
```

```
# ---------- Assessment branch ----------
if st.session_state.submitted and question:
    with st.spinner("🔍 Analyzing your application and generating response..."):
        ctx = load_retrieved()
        if not ctx:
            ctx = load_all_chunks()
        external_info = search_external(question, country)
        if external_info:
            ctx = (ctx + "\n\n" + external_info) if ctx else external_info
        ctx = ctx[:50000]

        answer = ask_model(question, ctx, country, st.session_state.user_details)
        save_qa_to_file(question, answer)

        st.markdown("---")
        st.subheader("✅ Assessment Result")
        st.write("Here is a generated response using the available visa documents and current public information.")
        st.write(answer)

        if st.button("🔄 Start New Query"):
            st.session_state.clear()
            st.rerun()
```

```
# ---------- Simple Q&A branch ----------
elif question and not st.session_state.show_form:
    if st.button("Get Answer"):
        with st.spinner("🔍 Generating response..."):
            ctx = load_retrieved()
            if not ctx:
                ctx = load_all_chunks()
            external_info = search_external(question, country)
            if external_info:
                ctx = (ctx + "\n\n" + external_info) if ctx else external_info
            ctx = ctx[:50000]

            answer = ask_model(question, ctx, country)
            save_qa_to_file(question, answer)
            st.markdown("---")
            st.subheader("📌 Answer")
            st.write("Here is a generated response using the available visa documents and current public information.")
            st.write(answer)
```

CONCLUSION

The Swift Visa project successfully demonstrates the use of artificial intelligence to provide accurate and reliable visa eligibility guidance. By integrating document preprocessing, embedding generation, vector database retrieval using FAISS, and a Large Language Model augmented with a retriever, the system ensures that answers are generated strictly from official visa documents using the Retrieval-Augmented Generation (RAG) approach. The inclusion of a Streamlit-based web application makes the system interactive, user-friendly, and easily accessible, allowing users to ask visa-related questions and receive real-time responses without technical complexity. Overall, the project highlights an effective, scalable, and practical AI-driven solution for visa assistance and showcases how modern AI tools can be applied to real-world information systems.