QA Chatbot with AWS Bedrock & Streamlit

This technical document presents the Regeneron Ethics QA Chatbot, a Retrieval-Augmented Generation (RAG) system designed for querying Regeneron's public ethics policy. Built with AWS Bedrock, Streamlit, LangChain, and FAISS, the chatbot loads a public PDF, generates embeddings, and utilizes Claude v2 for AI-generated answers. The following sections describe user workflow, architecture, key implementation steps, and offer actionable conclusions with resource links.

by Sandeep Patharkar

User Experience Demo

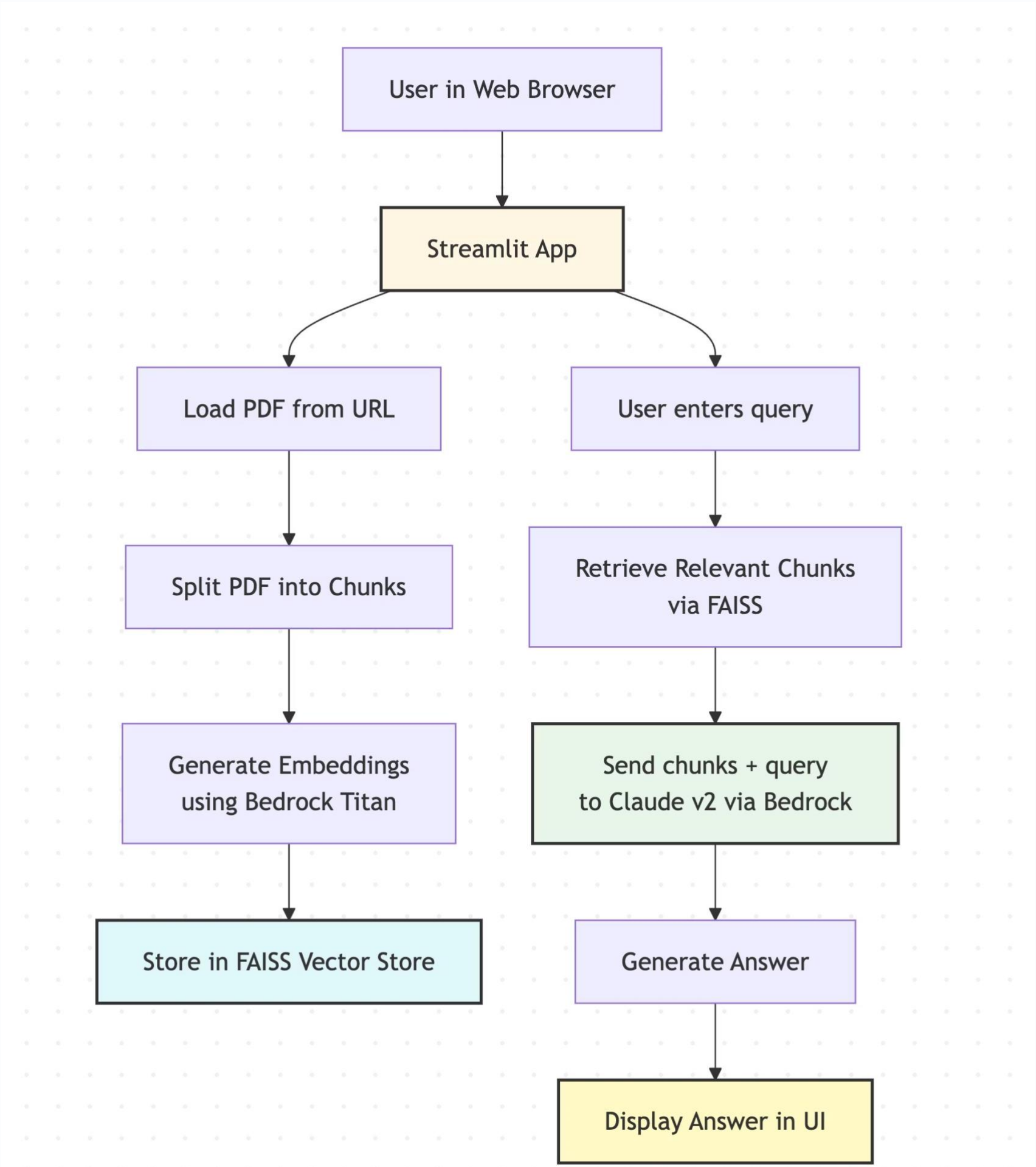
The chatbots interaction begins when a user accesses the Streamlit-based web interface. Users enter any question pertaining to Regenerons ethics policy. The application responds in real time, displaying an AI-generated answer specifically grounded in content from the official ethics PDF.

- User opens the web app
- Inputs an ethics policy question
- AI provides a trustworthy answer based on the official document
- Interface is intuitive and accessible, optimized for clarity

System Architecture Overview

The application employs a modular RAG workflow to ensure accurate and contextually relevant answers:

- PDF is fetched from a trusted public source
- Document is split into overlapping, context-preserving chunks
- Each chunk is embedded using AWS Bedrock Titan
- Embeddings are managed in a FAISS vector database for rapid similarity search
- User queries are processed via Streamlit
- Relevant chunks are retrieved and sent to Claude v2 for response generation
- The resulting answer is displayed immediately to the user


User in Web Browser
Streamlit App
Load PDF from URL
User enters query
Split PDF into Chunks
Retrieve Relevant Chunks via FAISS
Generate Embeddings using Bedrock Titan
Send chunks + query to Claude v2 via Bedrock
Store in FAISS Vector Store
Generate Answer
Display Answer in UI

Step-by-Step Implementation Details

1. PDF Loading

```
loader = PyPDFLoader(pdf_url)
documents = loader.load()
```

Loads the Regeneron ethics policy from a public URL, supporting direct remote document access.

2. Chunking for Context

```
text_splitter = RecursiveCharacterTextSplitter(chunk_size=1000, chunk_overlap=100)
chunks = text_splitter.split_documents(documents)
```

Keeps contextual integrity by overlapping content; improves answer accuracy.

3. Embedding & Vector Storage

```
embeddings = BedrockEmbeddings(model_id="amazon.titan-embed-text-v1")
db = FAISS.from_documents(chunks, embeddings)
```

Transforms text into semantic vectors, enabling rapid similarity-based retrieval.

Retrieval, Querying, and User I/O

4. Retrieval Chain Construction

```
qa_chain = RetrievalQA.from_chain_type(
    llm=bedrock_llm, chain_type="stuff",
    retriever=db.as_retriever(search_kwargs={'k': 3}))
```

Leverages Claude v2 on AWS Bedrock through LangChain to answer user queries using top-scoring document chunks.

5. User Input and Answer Display

```
query = st.text_input("Enter your question:")
st.write("Answer:", qa_chain.run(query))
```

Integrates Streamlit for real-time user queries and immediate answer output, completing the RAG feedback loop.

Technology Stack

AWS Bedrock & Titan

- Highly scalable embedding and LLM service
- Supports multiple model endpoints including Claude v2
- Offers robust API and security features

LangChain

- Framework for chaining LLMs and retrievers
- Eases RAG pipeline orchestration and extension
- Integrates directly with FAISS, Bedrock, and Streamlit

Streamlit

- Rapid web frontend deployment for Python apps
- Simple syntax for text inputs and outputs
- User-friendly, low-latency experience

Benefits, Security, and Limitations

- **Benefits:** Delivers accurate, real-time answers directly from authoritative company documents, reducing manual policy referencing.
- **Security:** Using AWS-managed resources improves security, compliance, and scalability. FAISS storage is local to the application, minimizing external data risk. No user data is stored.
- **Limitations:** RAG performance depends on the quality and format of source PDFs; context windows and chunking strategy may limit thoroughness for ambiguous or complex queries.
- **Future Enhancements:** Consider multi-document support, role-based access for sensitive materials, and periodic update triggers for the document source.

Conclusions and Resources

The Regeneron Ethics QA Chatbot demonstrates a robust RAG-based workflow integrating AWS Bedrock, Streamlit, and LangChain for interactive document QA. This implementation provides technical audiences a template for scalable, secure, and document-grounded chatbots. Future improvements may build on current modularity for expanded document collections and enhanced access controls.

- **LangChain Documentation**
- **AWS Bedrock**
- **Streamlit**
- GitHub Repository