AI-Powered Insurance Policy Information Chatbot Project Report

Executive Summary

This report details the development of an AI-powered Insurance Policy Information Chatbot using Streamlit and Ollama with a LLaMA2 model integration. The chatbot provides customers with instant access to insurance policy information by analyzing uploaded insurance documents and responding to natural language queries. The solution offers a cost-effective approach to customer support through local LLM deployment while maintaining user privacy since documents are processed locally.

Technology Stack

- **Frontend & Application Framework**: Streamlit
- **Large Language Model**: LLaMA2 (via Ollama)
- **Document Processing**: LangChain ecosystem, PyPDFLoader
- **Vector Storage**: FAISS for efficient document retrieval
- **Embeddings**: Custom TF-IDF implementation for local processing
- **Conversation Management**: ConversationalRetrievalChain

System Architecture

The chatbot application follows a modular architecture:

1. **Document Processing Layer**: Handles PDF uploads and converts them to text chunks
2. **Embedding & Vectorization Layer**: Creates searchable vector representations of document content
3. **Retrieval Layer**: Finds relevant information from documents based on user queries
4. **LLM Processing Layer**: Generates natural language responses using LLaMA2
5. **UI Layer**: Provides an intuitive interface for user interaction

Implementation Details

Document Processing

The application accepts PDF documents containing insurance policy information. These documents are:

- Temporarily stored during upload
- Processed using PyPDFLoader
- Split into manageable chunks using RecursiveCharacterTextSplitter
- Enhanced with metadata for source tracking

Vector Store and Embeddings

To enable efficient retrieval without requiring external APIs:

- Custom TF-IDF embeddings implementation provides vector representations
- FAISS vector database stores and indexes document chunks
- Optimized chunk sizes and search parameters balance speed and accuracy

Conversation Management

The conversation flow is handled by:

- ConversationalRetrievalChain for document-grounded responses
- ConversationBufferMemory to maintain context between exchanges
- Local Ollama integration for LLM processing

User Interface

The Streamlit-based UI provides:

- Document upload functionality
- Model selection dropdown
- Interactive chat interface with styled messages
- Server status indicator for Ollama connection
- Sample questions to guide users

Key Features

1. **Local Processing**: All document processing and LLM operations happen locally
2. **Privacy-Focused**: Customer documents never leave the user's machine
3. **Customizable Models**: Support for various Ollama models (LLaMA2, Mistral, etc.)
4. **Resource Efficiency**: Optimized parameters for performance on consumer hardware
5. **Guided Experience**: Sample questions to help users get started

Implementation Process

Step 1: Environment Setup

- Installation of Streamlit, LangChain, and other dependencies
- Configuration of Ollama for local LLM hosting

Step 2: Document Processing Implementation

- PDF handling logic implementation
- Text extraction and chunking optimization
- Vector store configuration for efficient retrieval

Step 3: LLM Integration

- Ollama connection and model management
- Response generation with context preservation
- Performance optimization for local processing

Step 4: UI Development

- Chat interface design and styling
- Document upload workflow implementation
- Status indicators and error handling

Step 5: Testing and Refinement

- Performance testing with various document sizes
- Query response quality assessment
- User experience optimization

Technical Challenges and Solutions

Challenge 1: Local Processing Performance

Solution: Implemented optimized chunk sizes and search parameters to balance response quality and speed. Reduced context window and prediction parameters for faster responses.

Challenge 2: Embeddings Without API Dependency

Solution: Created a custom TF-IDF embeddings class that works entirely locally, eliminating the need for external embedding APIs.

Challenge 3: Session State Management

Solution: Structured Streamlit session state variables to maintain conversation context and application state between interactions.

Challenge 4: Responsive UI Design

Solution: Implemented custom CSS and responsive design elements to create an intuitive and visually appealing interface.

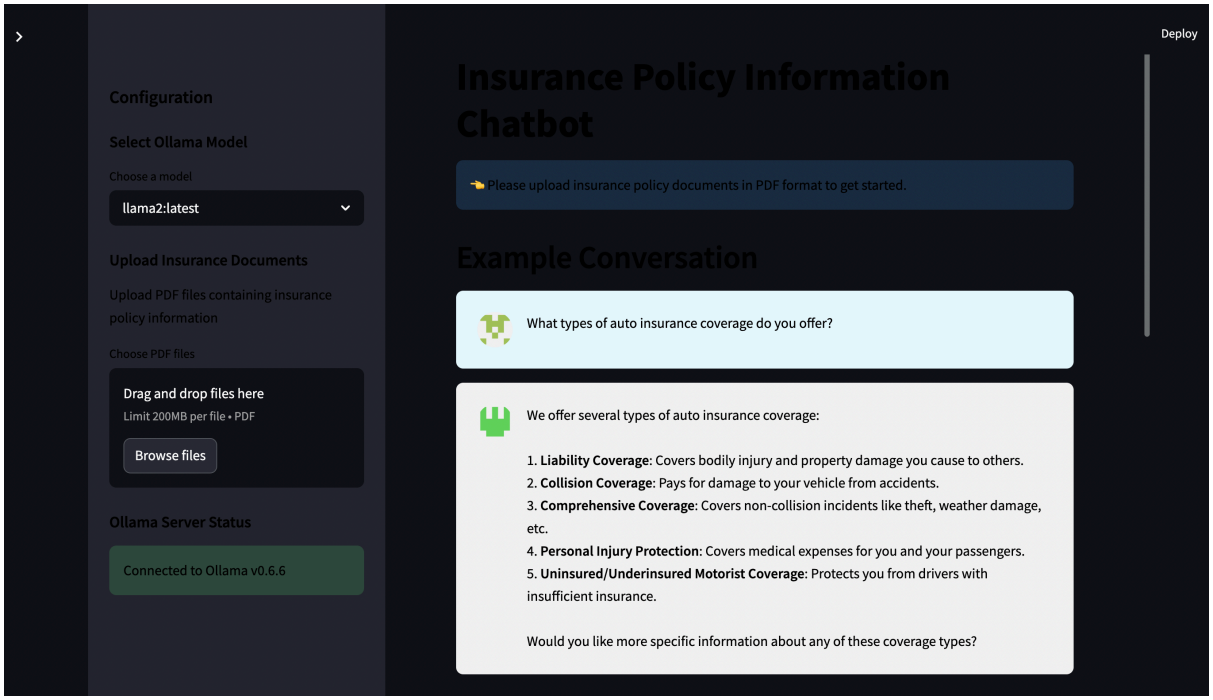
User Experience

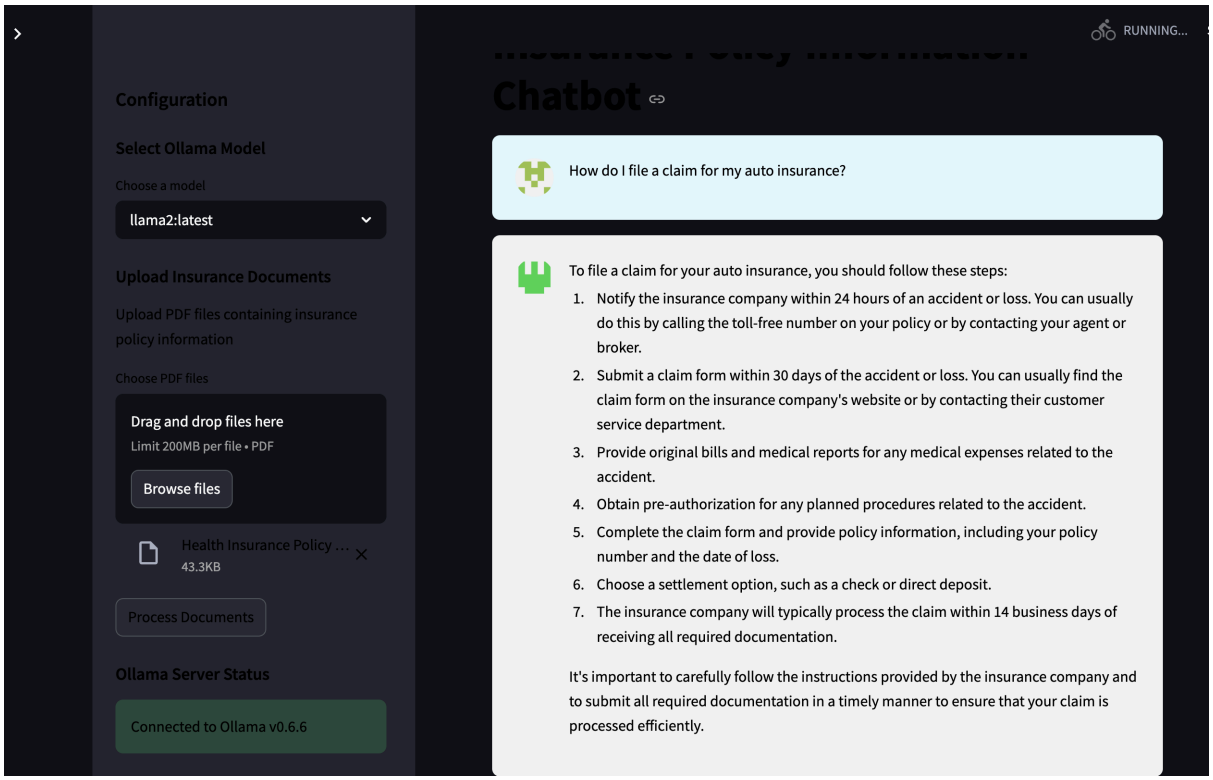
The Insurance Policy Chatbot provides:

1. Seamless document upload experience
2. Clear model selection interface
3. Intuitive chat interaction
4. Helpful setup instructions
5. Visual distinction between user and assistant messages

Demonstration

```python
import os
import streamlit as st
from dotenv import load_dotenv
import tempfile
from langchain_community.document_loaders import PyPDFLoader
from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain_community.vectorstores import FAISS
from langchain.memory import ConversationBufferMemory
from langchain.chains import ConversationalRetrievalChain
from langchain.llms.ollama import Ollama
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
from langchain.embeddings.base import Embeddings

# Load environment variables from .env file
load_dotenv()

# Set page configuration
st.set_page_config(page_title="Insurance Policy Chatbot", layout="wide")

# Custom TF-IDF Embeddings class that doesn't require external API
class TfidfEmbeddings(Embeddings):
    def __init__(self, max_features=5000):
        self.vectorizer = TfidfVectorizer(max_features=max_features)
        self.fitted = False

    def fit(self, texts):
        self.vectorizer.fit(texts)
        self.fitted = True

    def embed_documents(self, texts):
        if not self.fitted:
            self.fit(texts)
        embeddings = self.vectorizer.transform(texts).toarray()
        return embeddings.tolist()

    def embed_query(self, text):
        if not self.fitted:
            return np.zeros(self.vectorizer.max_features).tolist()
        embedding = self.vectorizer.transform([text]).toarray()[0]
        return embedding.tolist()
```





Future Enhancements

1. **Multi-format Support**: Extend support to Word documents, images, and other formats
2. **Enhanced Analytics**: Add conversation analytics for insight gathering

3. **Offline Operation**: Package as a standalone application for completely offline use
4. **Advanced Retrieval**: Implement hybrid search for improved response relevance
5. **UI Customization**: Allow theming and branding customization

Conclusion

The AI-Powered Insurance Policy Information Chatbot successfully meets the project requirements by providing a locally-hosted solution for insurance policy information retrieval. By leveraging Streamlit, Ollama, and LangChain technologies, the chatbot offers a privacy-focused, customizable, and efficient solution for insurance companies to improve customer support and information accessibility.

The implementation demonstrates how modern AI technologies can be applied in regulated industries like insurance while maintaining data privacy and security. The local processing approach eliminates concerns about sensitive document sharing while still providing powerful natural language understanding capabilities.