ALGORITHM

**TECHSTACK USED:**

Programming language – python

Langchain - Generative AI framework

Frontend – Flask

LLM – meta Llama 2

Vector BB – Pinecone

**STEPS:**

Step-1: Create and Activate Environment

Open the project repository.

Run the following commands:

conda create -n medibot python=3.10 -y

conda activate medibot

Step-2: Install Requirements and Configure Environment

Installing dependencies

Create an environment file in the root directory and add Pinecone and OpenAI credentials

Store embeddings in Pinecone

Start the app

AWS Setup for Deployment

Login to AWS Console.

Create IAM User for deployment with permissions for:

EC2 (virtual machine setup)

ECR (Docker image storage)

Deployment Process

Build Docker Image and push to ECR.

Launch an EC2 Instance:

Pull the Docker image from ECR.

Launch the Docker image.

AWS IAM Policies Needed

AmazonEC2ContainerRegistryFullAccess

AmazonEC2FullAccess

Setup Steps for ECR and EC2

Create ECR Repository and save the URI (e.g., 970547337635.dkr.ecr.ap-south-1.amazonaws.com/medicalchatbot).

Create EC2 Instance (Ubuntu)

==================================================

Step-by-Step Algorithm for Medical Knowledge Retrieval System

Back-End Side

Data Preparation

Step1: Load The Gale Encyclopedia of Medicine PDF file.

Use pypdf to extract text from each page of the PDF, turning it into a continuous text structure for processing.

Step2: Preprocess the text by removing headers, footers, and special characters to prepare clean data for chunking.

Chunking the Text Data

Step1: Define chunk size (e.g., 100, 200, or 300 tokens) and overlap size (e.g., 20 or 30 tokens) to maintain context across chunks.

Step2: Use LangChain’s recursiveCharacterTextSplitter to split the cleaned text into chunks based on the predefined chunk size and overlap.

Why Chunking? Models like Llama2 have a maximum token limit (e.g., 4096 tokens), so the data needs to be broken down into smaller, manageable parts.

Overlap Handling: Specify a small overlap (e.g., 20 tokens) so each chunk carries a portion of the previous one, preserving continuity of information across chunks.

Embedding Generation

Step1: Load the free embedding model all-MiniLM-L6-v2 from Hugging Face via the sentence-transformers==2.2.2 library.

Step2: Convert each text chunk into a vector representation (embedding) using this model. These embeddings capture the semantic content of each chunk, making it easier to search and compare in later stages.

Output: A collection of embeddings representing each chunk of the medical text.

Creating a Semantic Index

Step1: Use Pinecone, a vector database, to store and organize the embeddings.

Step2: Build a semantic index by clustering the embeddings based on similarity, using distance metrics like cosine similarity.

Clustering: Similar embeddings are grouped, creating clusters within Pinecone’s semantic index. This facilitates efficient retrieval based on similarity.

Output: A structured semantic index that organizes the embeddings for rapid, similarity-based searches.

Building the Knowledge Base

Step1: Insert all embeddings and associated metadata (chunk ID, chunk size, overlap details) into Pinecone to create the knowledge base.

Step2: Confirm the knowledge base’s search capabilities to ensure it can retrieve relevant vectors based on query embeddings.

Alternative: If necessary, consider using ChromaDB as an alternative to Pinecone for storing the embeddings.

Output: A comprehensive, organized knowledge base that holds all vectorized data and is optimized for query-based searches.

User Side

User Query Processing

Step1: Accept the user’s query via the Flask web application interface.

Step2: Preprocess the query (e.g., remove punctuation, convert to lowercase) to standardize it before embedding.

Query Embedding Generation

Step1: Convert the user’s query into an embedding using the same model (all-MiniLM-L6-v2 from sentence-transformers==2.2.2) used for creating text chunk embeddings.

Output: A query embedding that represents the meaning of the user’s question, enabling meaningful comparisons with the knowledge base.

Searching the Knowledge Base

Step1: Send the query embedding to the Pinecone knowledge base.

Step2: Perform a similarity search to retrieve the top n most relevant chunks, based on how close each chunk's embedding is to the query embedding.

Output: A ranked list of text chunks from the knowledge base, ordered by relevance to the user’s query.

Response Generation with Llama2

Step1: Provide the ranked list of text chunks, along with the user’s original query, to Llama2 for response generation.

Step2: Llama2 processes both the query and the content of the retrieved chunks, synthesizing the most relevant information into a coherent response.

Filtering: Llama2 helps filter out extraneous information and generates a concise, accurate response based on the content and user’s query.

Output: A well-structured answer that directly addresses the user’s question.

Returning the Answer to the User

Step1: Display the generated response in the Flask web interface for easy user access.

Optional: Allow users to provide feedback on the answer’s relevance, which can be used to refine the knowledge base or improve the response-generation process.

Technologies Used

Programming Language: Python

GenAI Framework: LangChain (LlamaIndex as an alternative)

Web Framework: Flask (for user interface and query handling)

Large Language Model: Meta Llama2 (for response generation)

Vector Database: Pinecone (or ChromaDB as an alternative, if required)

Important Libraries

ctransformer==0.2.5: To load and utilize the quantized Llama2 model on CPU.

sentence-transformers==2.2.2: Embedding model from Hugging Face (all-MiniLM-L6-v2) for generating vector embeddings.

pinecone-client: For connecting to and interacting with the Pinecone vector database.

langchain==0.0.225: Provides tools for seamless integration across all components.

flask: Web framework for handling user interaction.

pypdf: For extracting text from PDF files.

Implementation Instructions

Prepare a requirements.txt File

Include all the necessary libraries specified above.

Install them by running:

bash

Copy code

pip install -r requirements.txt

Integrate Components Using LangChain

Use LangChain to link text extraction, chunking, embedding generation, and knowledge base retrieval, forming a streamlined workflow for the entire process.

Set Up Flask Web Interface

Develop a simple Flask interface to handle user queries, display responses, and optionally collect feedback for ongoing system improvement.

This detailed, technology-specific algorithm provides clear instructions for setting up a robust medical question-answering system. Each step is aligned with your specified tech stack, ensuring practical implementation and alignment with research objectives.