

# **CARDIO-BOT – YOUR HEART HEALTH COMPANION**

*Submitted by*

**Parimala Dharshini M (RA2311008020132)**

**Raghul P (RA2311008020147)**

**Varshitha S (RA2311008020150)**

**Hayden Joseph D (RA2311008020151)**

Under the guidance of

**Dr. Deeptha R**

**Assistant Professor**

**Department of Information Technology**

*In partial fulfillment for the award of the degree*

**of**

**BACHELOR OF TECHNOLOGY**

**In**

**INFORMATION TECHNOLOGY**

**of**

**FACULTY OF ENGINEERING AND TECHNOLOGY**



**RAMAPURAM, CHENNAI-600089**

**APRIL 2025**

# **SRM INSTITUTE OF SCIENCE AND TECHNOLOGY**

(Deemed to be University Under Section 3 of UGC Act, 1956)

## **BONAFIDE CERTIFICATE**

Certified that the Mini Project titled “**Cardio-Bot - your heart health companion**” is the Bonafide certificate of **Parimala Dharshini M – RA2311008020132, Raghul P – RA2311008020147, Varshitha S – RA2311008020150, Hayden Joseph D – RA2311008020151** of II Year IT submitted for the course 21CSC206T – Artificial Intelligence for the Academic Year 2024 – 25 Even Semester.

### **SIGNATURE**

Dr. Deeptha R

Assistant Professor

Information Technology

SRM Institute of Science and Technology

Ramapuram, Chennai-78

## TABLE OF CONTENTS

<b>TITLE NO.</b>	<b>TITLE NAME</b>	<b>PAGE NO</b>
	<b>ABSTRACT</b>	<b>4</b>
<b>1</b>	<b>INTRODUCTION</b>	<b>5</b>
<b>2</b>	<b>SCOPE AND MOTIVATION</b>	<b>7</b>
<b>3</b>	<b>DESCRIPTION</b>	<b>9</b>
<b>4</b>	<b>ALGORITHM OR PSEUDOCODE</b>	<b>13</b>
<b>5</b>	<b>IMPLEMENTATION</b>	<b>18</b>
<b>6</b>	<b>OUTPUT</b>	<b>24</b>
<b>7</b>	<b>APPLICATIONS</b>	<b>27</b>
<b>8</b>	<b>CONCLUSION</b>	<b>28</b>

## ABSTRACT

This report outlines the design and implementation of **Cardio Bot**, an intelligent, document-driven conversational assistant tailored to facilitate understanding and engagement with cardiovascular health information. The primary goal of Cardio Bot is to bridge the gap between complex medical literature and end-users by transforming static health documents into interactive, accessible, and informative experiences. The application begins its workflow by ingesting PDF documents—such as medical reports, guidelines, or educational materials—using **PyMuPDF (fitz)**, a powerful library for extracting structured and unstructured text from PDF files. After the text is extracted, it undergoes a chunking process where the content is segmented into smaller, manageable units. This facilitates better context handling and improves performance during language model inference. These text chunks are encapsulated in Document objects, making them ready for further processing. While the application optionally supports vector-based semantic retrieval using **Sentence Transformer Embeddings** and **Chroma**, the core interaction layer is driven by advanced language models. Cardio Bot integrates **Cohere's command-xlarge-nightly** language model through its API to handle user queries with contextually relevant, natural language responses. This cloud-based solution ensures that responses are rich, human-like, and domain-aware. For environments where internet access is limited or where data privacy is a concern, the system also supports local inference through **LlamaCpp**, allowing users to load and utilize a pre-trained model such as **BioMistral-7B** directly on their machine. The bot includes utility features such as cache clearing for Hugging Face transformers and text saving, ensuring an optimized and clean execution environment. To support extended queries and avoid truncation in responses, the system implements a recursive strategy to retrieve longer answers from Cohere's API. Cardio Bot is a flexible and modular application that serves both educational and clinical purposes. It empowers users—patients, medical students, or healthcare professionals—to interact with technical content in a more intuitive and conversational manner. By combining robust text extraction, intelligent language modeling, and scalable architecture, Cardio Bot exemplifies the potential of AI-driven tools in the healthcare information space.

## INTRODUCTION

Cardiovascular diseases (CVDs) remain the leading cause of death globally, contributing to millions of fatalities each year. Despite the growing availability of medical knowledge and public health awareness campaigns, a significant gap persists between the abundance of health-related information and the general public's ability to access and comprehend it effectively. Much of the critical information regarding heart health is distributed through technical documents, research papers, clinical reports, and guidelines—often packaged as PDF files with dense medical terminology and complex formatting. These documents, although rich in insights, are not readily accessible or understandable to patients, caregivers, or even non-specialist healthcare professionals. This disconnect often leads to missed opportunities for early prevention, informed decision-making, and better patient outcomes.

The increasing capabilities of artificial intelligence, particularly in the field of natural language processing (NLP), offer promising solutions to bridge this information gap. The advent of large language models (LLMs) has revolutionized how machines understand, process, and generate human-like text, enabling more natural and context-aware interactions between users and machines. These advancements have paved the way for intelligent systems that can not only parse textual content but also engage users in meaningful dialogue. In this light, Cardio Bot has been developed as a conversational AI assistant specifically designed to interact with medical documents related to cardiovascular health. The goal of Cardio Bot is to make complex medical content more accessible and useful by allowing users to ask questions in natural language and receive accurate, contextually relevant responses.

Cardio Bot operates by first extracting content from PDF documents using PyMuPDF (fitz), a robust library known for its high-precision text retrieval capabilities. The extracted content is then segmented into smaller, logical chunks to improve the performance of downstream language models and maintain contextual integrity. These chunks can be optionally converted into embeddings using SentenceTransformer models and stored in a Chroma vector database to facilitate semantic search and improve the relevance of query results. Once a user submits a question, the system retrieves the most relevant chunks and passes them to a language model to generate a response.

The response generation layer of Cardio Bot supports two modes of operation. For users with internet access, the system integrates with Cohere's command-xlarge-nightly model via API, offering fast and fluent answers backed by powerful cloud-based infrastructure. For users in offline environments or those requiring greater control over data privacy, Cardio Bot also supports local inference through LlamaCpp, enabling the execution of models like BioMistral-7B directly on a user's machine. This dual setup ensures flexibility and adaptability to various deployment contexts, whether for personal use, clinical settings, or educational institutions.

By combining intelligent document parsing, semantic chunking, vector search, and advanced language models, Cardio Bot offers a new way to interact with static medical documents. Instead of reading through lengthy text, users can now simply ask questions and receive tailored answers instantly. This significantly enhances comprehension, saves time, and democratizes access to expert-level content. Cardio Bot is particularly valuable in scenarios where users may lack formal medical training but still need to understand critical information about their health or the health of others.

The development of Cardio Bot underscores the broader potential of AI-powered tools in transforming digital healthcare. It showcases how machine learning and NLP can be used to simplify complex knowledge and bring it closer to the people who need it most. Although this version focuses on cardiovascular content, the architecture of Cardio Bot is modular and can be extended to other medical or technical domains with minimal reconfiguration. As such, Cardio Bot serves not only as a functional application but also as a demonstration of what is possible when technology is used to break down barriers between information and understanding.

## SCOPE AND MOTIVATION

In the rapidly evolving landscape of digital healthcare, one of the most persistent challenges is the accessibility and understandability of critical medical information. Although modern medicine has made enormous strides in diagnosing and treating cardiovascular diseases, the communication of this knowledge to patients, students, and even healthcare professionals remain fragmented and often inaccessible. Cardio Bot has been conceived as a response to this challenge—an intelligent system aimed at making complex medical documents more interactive, understandable, and contextually navigable.

The **motivation** behind developing Cardio Bot is rooted in both practical and societal observations. Most individuals are provided with health-related information in formats that are static, dense, and laden with specialized terminology. These documents—clinical reports, research publications, diagnostic summaries, and health education manuals—are often formatted as PDF files that are not designed for direct human interaction. Patients and non-specialists often find it difficult to interpret the meaning of such content, which may result in anxiety, misinformation, or even incorrect health-related decisions. Furthermore, healthcare professionals, despite their expertise, are frequently burdened by time constraints that prevent them from manually reviewing lengthy documents in full.

In parallel, the recent breakthroughs in **artificial intelligence**—particularly in natural language processing—have unlocked new ways of engaging with textual data. The emergence of large language models (LLMs), capable of understanding complex questions and generating fluent responses, offers a powerful toolset to bridge the gap between technical documentation and user-friendly dialogue. Cardio Bot leverages these capabilities, combining document parsing, semantic chunking, and LLM-driven inference to create a virtual assistant capable of answering user queries based on the content of a given medical PDF.

The **scope** of this project is both technically ambitious and practically relevant. Cardio Bot begins with the ingestion of medical documents in PDF format, using the PyMuPDF (fitz) library to accurately extract and preserve textual content. This stage ensures that the bot works well even with varied layouts, embedded tables, and complex formatting. The extracted text is then processed and divided into smaller chunks that maintain logical coherence. This chunking strategy is essential for enabling context-aware inference and efficient retrieval of relevant information during the response generation phase.

To enhance semantic understanding and context alignment, these chunks can optionally be embedded using pre-trained Sentence Transformer models and stored in a Chroma vector database. This setup supports vector-based semantic search, allowing the system to retrieve the most contextually appropriate content in response to user questions. This makes the system significantly more efficient and accurate compared to traditional keyword-based search mechanisms.

For answering queries, Cardio Bot supports two operational models. In cloud-based mode, the system uses the Cohere API with its command-xlarge-nightly language model to produce fluent and informative responses. This is ideal for users with internet access who need real-time, high-quality interactions. For environments that require offline functionality, heightened privacy, or data security—such as hospitals, clinics, or remote healthcare settings—the system can fall back on **local inference** using LlamaCpp. This allows it to load models like **BioMistral-7B**, ensuring robust performance even without cloud connectivity.

The system's dual-mode architecture reflects its versatility and scalability. It can be deployed in personal health apps, academic research platforms, clinical decision-support systems, and telemedicine interfaces. Moreover, although this implementation focuses on cardiovascular health, the underlying design is modular and adaptable. By simply changing the source documents and fine-tuning prompts or embeddings, Cardio Bot can be repurposed for other medical domains such as diabetes management, mental health, paediatrics, or infectious disease education.

From a broader perspective, Cardio Bot embodies the increasing shift towards **patient empowerment and digital health literacy**. By transforming rigid and static documents into interactive, conversational knowledge bases, the system helps individuals take a more active role in understanding their health. It also serves as a tool for medical students and early-career professionals who seek to quickly assimilate complex information from trusted sources.

In conclusion, Cardio Bot is motivated by the need to democratize access to medical knowledge and improve healthcare communication using the capabilities of modern AI. Its scope covers intelligent document parsing, semantic understanding, contextual response generation, and flexible deployment. As healthcare systems worldwide continue to digitize, solutions like Cardio Bot are poised to become integral components in shaping the future of personalized, AI-assisted medical education and care.



## DESCRIPTION

Cardio Bot is a state-of-the-art, AI-driven healthcare assistant designed to bridge the gap between complex cardiovascular health documents and users who may not have extensive medical knowledge. Medical texts, especially those related to cardiovascular health, can be daunting due to their dense technical language, complex terminologies, and highly structured formats. These documents are often used by healthcare professionals, researchers, or students and are typically stored in PDF format. However, PDFs, while being a standard format for distributing medical documents, pose several challenges for users who need to extract useful, understandable information from them. These challenges include the technical nature of the content, the structural complexity of the documents (such as multi-column layouts, embedded images, and detailed tables), and the volume of data that needs to be processed. Cardio Bot addresses all of these challenges by providing a solution that allows users to interact with complex cardiovascular health documents through natural language queries, making the information both accessible and understandable.

At the heart of Cardio Bot is its ability to seamlessly extract and process text from PDF documents, a process that is often more complicated than it seems. Medical PDFs are rarely simple text documents; they often contain embedded figures, charts, multi-column structures, and complex formatting that can interfere with straightforward text extraction. To handle these challenges, Cardio Bot utilizes the **PyMuPDF (fitz)** library, a powerful Python-based library that is specifically designed to process and extract text from PDFs, including those with complex layouts. Whether a document includes scientific charts, tables, or multiple columns, PyMuPDF ensures that the text is extracted correctly and is in a format that can be easily used for further processing. This is a crucial step because the accuracy of the text extraction directly impacts the quality of the responses generated by the system. By effectively handling a wide range of PDF formats, Cardio Bot ensures that even the most complex medical documents can be processed efficiently.

Once the text is extracted from the PDF, it undergoes a process known as **text chunking**, where the document is divided into smaller, more manageable sections. This is an important step because language models, like the ones used in Cardio Bot, perform better when processing shorter pieces of text. Typically, these chunks are designed to contain about 1000 characters, making them small enough to be processed efficiently while still retaining enough context to provide meaningful answers. By chunking the text, Cardio Bot ensures that each segment can

be independently queried while preserving its original meaning and logical structure. This approach enhances the overall efficiency of the system, enabling it to handle large volumes of data in a way that is both manageable and accurate. This chunking method also allows the system to optimize its memory usage, making it possible to process large medical documents without overwhelming the system's computational resources.

Additionally, Cardio Bot goes beyond traditional methods of text extraction by **embedding** the extracted text chunks into **high-dimensional vectors** using **SentenceTransformer**, a model designed to convert text into vector representations. These vector embeddings capture the semantic meaning of each chunk, which allows Cardio Bot to perform **semantic search**. Unlike traditional keyword-based search, which relies on matching specific words or phrases, semantic search focuses on understanding the meaning behind the words. This allows Cardio Bot to return more accurate and contextually relevant responses to user queries. For example, if a user asks about the symptoms of a particular heart disease, Cardio Bot will be able to search for the most relevant document chunks based on the overall meaning of the query rather than just matching individual words. The ability to perform semantic search significantly improves the accuracy of the system's responses, especially when dealing with complex medical content.

These vector embeddings are stored in a **Chroma vector database**, which is optimized for fast retrieval of semantically similar text. This database allows Cardio Bot to efficiently search for and retrieve relevant information when a user asks a question. The Chroma database works by indexing each embedded chunk, which enables the system to quickly identify and return the most relevant sections of text based on the query's semantic content. This process ensures that Cardio Bot can generate highly accurate responses, even when the user's question is complex or specific.

Once the most relevant chunks of text are retrieved, Cardio Bot uses **advanced language models** to generate human-like responses. Cardio Bot supports both **cloud-based** and **local inference**, allowing it to adapt to different environments and user needs. In the cloud-based mode, Cardio Bot connects to **Cohere's command-xlarge-nightly model**, a powerful large language model capable of generating natural, coherent responses. The cloud-based system offers the benefit of leveraging the computational power of cloud servers, enabling Cardio Bot to handle complex queries and generate detailed responses in real time. For example, if a user asks a question about the treatment options for a specific cardiovascular disease, the system can process the query, retrieve relevant information from the document, and generate a detailed,

context-aware response. The system can answer not only basic questions but also more intricate and complex queries, such as those related to clinical guidelines, medication dosages, or the pathophysiology of specific diseases.

For users who require enhanced privacy or need the system to function in environments with limited or no internet connectivity, Cardio Bot also supports **local inference** using **LlamaCpp**, a framework that allows large language models to be run locally on a device. This mode is especially important for healthcare environments where data privacy and security are paramount, such as hospitals or clinics. By running models like **BioMistral-7B** locally, Cardio Bot ensures that sensitive patient data and queries are processed without being sent over the internet. This mode also provides the benefit of enabling offline functionality, making it suitable for areas with limited internet access, such as rural or remote regions.

To ensure that responses are always complete and thorough, Cardio Bot includes a feature for handling **long responses**. Medical queries often require detailed explanations, especially when dealing with complex conditions or treatment regimens. To avoid truncation or incomplete responses, the system monitors the length of the generated answers and automatically requests additional tokens if needed. This ensures that users receive comprehensive, fully-formed answers, no matter how long or detailed the query may be. For example, if a user asks a multi-part question about the treatment and prevention of heart disease, Cardio Bot will provide a complete, coherent response that addresses all aspects of the question, without cutting off critical information midway.

In addition to its core functionalities, Cardio Bot is designed with several **performance optimization features** to ensure that it operates smoothly, even when dealing with large datasets or complex queries. One such feature is its **cache-clearing mechanism**, which helps manage memory usage by clearing temporary data stored during the process of extracting and analyzing text. This ensures that the system remains responsive and efficient, even when processing large volumes of data. Additionally, once the system has processed a document and answered the user's queries, it performs cleanup operations to release any resources that were used, optimizing performance and preventing resource exhaustion. This makes Cardio Bot a highly reliable system that can handle long-term usage without compromising speed or accuracy.

The system's **extensibility** ensures that Cardio Bot can grow and evolve to meet the changing needs of users. While it is currently focused on cardiovascular health, its underlying

architecture is flexible enough to be adapted to other areas of medical research. The core components—text extraction, chunking, vector search, and response generation—can easily be reconfigured to support other medical domains, such as oncology, neurology, or endocrinology. This makes Cardio Bot a highly versatile tool that can be customized to meet a wide range of healthcare needs. For instance, Cardio Bot could be re-trained to process cancer-related research papers or provide information about diabetes management, thus expanding its utility beyond cardiovascular health.

Cardio Bot's versatility also allows it to be deployed across a wide range of settings. **Patients** can use it to better understand their conditions, treatment options, and preventive measures related to cardiovascular health. **Medical students** can benefit from its ability to quickly explain complex medical concepts and provide instant access to the latest research. **Healthcare professionals**, including doctors, nurses, and researchers, can use it as a decision support tool to quickly access guidelines, research, or patient-specific recommendations. Its ability to process large volumes of complex medical information and provide clear, contextually relevant responses makes it an invaluable resource for anyone working in the healthcare field. Furthermore, the system's ability to handle different types of medical documentation, whether clinical guidelines, research papers, or patient reports, ensures that it remains a comprehensive tool for all users.

In conclusion, Cardio Bot is a revolutionary tool that takes the vast and often impenetrable world of cardiovascular medical literature and transforms it into something accessible, understandable, and interactive. Through its use of advanced technologies such as AI-driven natural language processing, semantic search, and powerful local and cloud-based language models, Cardio Bot offers a transformative solution to the challenges posed by traditional medical literature. Its ability to handle complex documents, provide clear and accurate responses, and maintain privacy and security makes it an invaluable tool for patients, healthcare professionals, and researchers alike. With its potential to adapt to other medical domains and integrate with various healthcare platforms, Cardio Bot represents the future of digital healthcare assistance, empowering users to make informed decisions and better understand complex medical content.

## ALGORITHM OR PSEUDOCODE

### Algorithm

**1. Input: PDF document (medical content related to cardiovascular health)**

- Load the PDF file containing medical content.

**2. Text Extraction:**

- Use a PDF parsing library (like PyMuPDF) to extract text from the document.
- For each page, extract text while handling complex layouts (e.g., multi-columns, embedded images, tables).

**3. Text Chunking:**

- Split the extracted text into smaller chunks (1000-character segments) to improve processing efficiency.
- Store each chunk as a separate document for further analysis.

**4. Vectorization (Embedding):**

- Use the SentenceTransformer model to convert each chunk into vector embeddings.
- Store the resulting vector embeddings in a vector store (Chroma database).

**5. User Query Processing:**

- When a user submits a query, convert the query into a vector using the same SentenceTransformer model.
- Perform a **semantic search** on the Chroma vector store to find the most relevant chunks based on the user query's semantic meaning.

## 6. Response Generation:

- Retrieve the most relevant chunks based on the semantic search.
- Use a language model (such as Cohere's command-xlarge-nightly or LlamaCpp for local inference) to generate a human-like response using the relevant chunks.

## 7. Long Response Handling:

- If the response is too long or cut off, automatically request additional tokens to complete the response.

## 8. Output:

- Return the generated response to the user.

## 9. Clean up:

- Clear cache and temporary files to optimize memory usage.

## Pseudocode

Pseudo # Step 1: Input PDF document

```
def load_pdf(pdf_file_path):
```

```
    # Open PDF document
```

```
    pdf_document = fitz.open(pdf_file_path)
```

```
    return pdf_document
```

# Step 2: Extract Text from PDF

```
def extract_text_from_pdf(pdf_document):
```

```
    pdf_text = ""
```

```
    num_pages = pdf_document.page_count
```

```
    for page_num in range(num_pages):
```

```
        page = pdf_document.load_page(page_num)
```

```

    pdf_text += page.get_text()

return pdf_text

# Step 3: Text Chunking

def chunk_text(text, chunk_size=1000):

    chunks = [text[i:i + chunk_size] for i in range(0, len(text), chunk_size)]

    return chunks

# Step 4: Convert text chunks to vector embeddings

def convert_text_to_embeddings(chunks):

    embeddings = []

    for chunk in chunks:

        embedding = SentenceTransformer().encode(chunk)

        embeddings.append(embedding)

    return embeddings

# Step 5: Semantic Search for User Query

def semantic_search(query, embeddings, vector_store):

    query_embedding = SentenceTransformer().encode(query)

    # Retrieve the most similar chunk based on the query embedding

    similar_chunks = vector_store.search(query_embedding)

    return similar_chunks

# Step 6: Generate Response from Language Model (Cohere or LlamaCpp)

def generate_response(query, similar_chunks):

    prompt = prepare_prompt(query, similar_chunks)

    response = generate_with_llm(prompt) # Using Cohere or LlamaCpp

    return response

```

```

# Helper function for response generation with Cohere

def generate_with_llm(prompt):

    cohere_client = cohere.Client(COHERE_API_KEY)

    response = cohere_client.generate(

        model="command-xlarge-nightly",

        prompt=prompt,

        max_tokens=1000

    )

    return response.generations[0].text.strip()

# Step 7: Handle Long Responses (if truncated)

def handle_long_responses(response):

    if len(response.split()) < 1000: # Threshold for incomplete response

        additional_response = generate_with_llm(response) # Request more tokens

        return response + additional_response

    return response

# Step 8: Return Response to User

def return_response_to_user(response):

    print(response) # Or send via chatbot interface

# Step 9: Cleanup

def cleanup():

    # Clear cache, temporary files, and perform cleanup operations

    clear_cache()

# Main function

```



```

def main(pdf_file_path, query):

    pdf_document = load_pdf(pdf_file_path)

    extracted_text = extract_text_from_pdf(pdf_document)

    # Chunk the text into manageable sections

    text_chunks = chunk_text(extracted_text)

    # Convert chunks to embeddings and store in vector store

    embeddings = convert_text_to_embeddings(text_chunks)

    vector_store = store_embeddings_in_chroma(embeddings)

    # Process the query and perform semantic search

    similar_chunks = semantic_search(query, embeddings, vector_store)

    # Generate response from language model based on retrieved chunks

    response = generate_response(query, similar_chunks)

    # Handle long responses if needed

    full_response = handle_long_responses(response)

    # Return the generated response to the user

    return_response_to_user(full_response)

    # Cleanup resources

    cleanup()

# Example usage

query = "What are the symptoms of heart failure?"

pdf_file_path = "path_to_pdf_file"

main(pdf_file_path, query)code

```

## Detailed Explanation of the Pseudocode:

1. **Input PDF Document:** The system starts by loading a PDF file, which contains cardiovascular health-related information.
2. **Extract Text:** It then extracts text from the document page by page, handling multi-column and complex layouts.
3. **Text Chunking:** The extracted text is split into smaller, manageable chunks (each with around 1000 characters).
4. **Embedding Conversion:** Each chunk is converted into a vector embedding, which captures the semantic meaning of the text. These vectors are stored in a vector store for efficient retrieval.
5. **Semantic Search:** The user's query is also converted into a vector embedding, and a semantic search is performed against the vector store to retrieve the most relevant document chunks.
6. **Generate Response:** Based on the retrieved chunks, a language model (like Cohere or LlamaCpp) is used to generate a detailed response.
7. **Handle Long Responses:** If the generated response is cut off or too short, the system requests more tokens to complete the response.
8. **Return Response:** The system returns the complete response to the user, ensuring it answers the query thoroughly.
9. **Cleanup:** Finally, the system clears any temporary data and memory resources.

## IMPLEMENTATION

```
import os

import re

from PyPDF2 import PdfReader

from langchain_community.embeddings import SentenceTransformerEmbeddings

from langchain_community.vectorstores import Chroma

from langchain.prompts import PromptTemplate

from langchain.schema import Document

from langchain_community.llms import LlamaCpp
```

```

import warnings

import fitz # PyMuPDF

import requests

from dotenv import load_dotenv

import cohere

# Load environment variables from .env file

load_dotenv()

# Define the Cohere API key directly in the code (replace with your actual API key)

COHERE_API_KEY = "ppjDiRL0osIzqc0iFHZ53OkVNkx1QvexXIAUciPL"

# Clear Hugging Face cache function

def clear_cache():

    cache_dir = os.path.expanduser("~/cache/huggingface/transformers")

    if os.path.exists(cache_dir):

        for root, dirs, files in os.walk(cache_dir, topdown=False):

            for name in files:

                os.remove(os.path.join(root, name))

            for name in dirs:

                os.rmdir(os.path.join(root, name))

    clear_cache()

print("Cache cleared.")

# Path to your repaired PDF file

pdf_file_path = r"C:\\Users\\parim\\Downloads\\healthyheart (1)_repaired.pdf"

# Open the PDF file

pdf_document = fitz.open(pdf_file_path)

```

```

# Get the number of pages in the PDF

num_pages = pdf_document.page_count

print(f"Number of pages in the PDF: {num_pages}")

# Initialize an empty string to hold the extracted text

pdf_text = ""

# Extract text from all pages

for page_num in range(num_pages):

    page = pdf_document.load_page(page_num) # Get each page

    pdf_text += page.get_text() # Extract text from the page

# Optionally, save the extracted text to a .txt file with UTF-8 encoding
with open("extracted_text.txt", "w", encoding="utf-8") as file:

    file.write(pdf_text)

print("Text extraction completed successfully.")

# Function to chunk text into smaller pieces

def chunk_text(text, chunk_size):

    return [text[i:i + chunk_size] for i in range(0, len(text), chunk_size)]

# Initialize an empty list to store text from each page

text = [page.get_text() for page in pdf_document]

chunks = [chunk for page in text for chunk in chunk_text(page, chunk_size=1000) if
chunk.strip()]

documents = [Document(page_content=chunk) for chunk in chunks]

# Create the vector store (optional step for context retrieval)

# If you still want to use vector search, you can continue using it as per your design

# Otherwise, skip this part and focus on using Cohere for responses

```

```

# Set up the Llama model correctly (if you're using local models)

try:

    # Path to the Llama model (update this path as needed)

    llm_model_path = r'C:\\Users\\parim\\Documents\\cardio-bot\\BioMistral-
7B.Q4_K_M.gguf'

    # Load the LLM model

    llm = LlamaCpp(

        model_path=llm_model_path,

        temperature=0.2,

        max_tokens=2048,

        top_p=0.95,

        context_length=32768 # Include the context length directly here

    )

    print("LLM model loaded successfully!")

except Exception as e:

    print(f"Error loading LLM model: {e}")

    llm = None # Set llm to None or a mock value

# Define the get_response function to use Cohere API

def get_response(query):

    try:

        # Initialize Cohere client

        co = cohere.Client(COHERE_API_KEY)

        # Generate a response using Cohere's generate endpoint

        response = co.generate(

            model="command-xlarge-nightly",

```

```

    prompt=query,

    max_tokens=1000 # Increased max_tokens for longer responses
)

# Extract and return the generated response

return response.generations[0].text.strip()

except Exception as e:

    return f"Error connecting to the Cohere API: {e}"

# Define a function to handle longer responses in case of truncation
def get_long_response(query):

    try:

        co = cohere.Client(COHERE_API_KEY)

        response = co.generate(

            model="command-xlarge-nightly",

            prompt=query,

            max_tokens=1000 # Adjust this value

        )

        # Check if the response is cut off and needs more tokens

        full_response = response.generations[0].text.strip()

        # If it's cut off, request more tokens

        while len(full_response.split()) < 1000: # Adjust this threshold as needed

            response = co.generate(

                model="command-xlarge-nightly",

                prompt=query,

                max_tokens=1000

```

```

    )

    full_response += response.generations[0].text.strip()

    return full_response

except Exception as e:

    return f"Error connecting to the Cohere API: {e}"

# Example usage of the get_response function

if __name__ == "__main__":

    query = "What are the risk factors for heart health?"

    response = get_long_response(query)

    print(response)

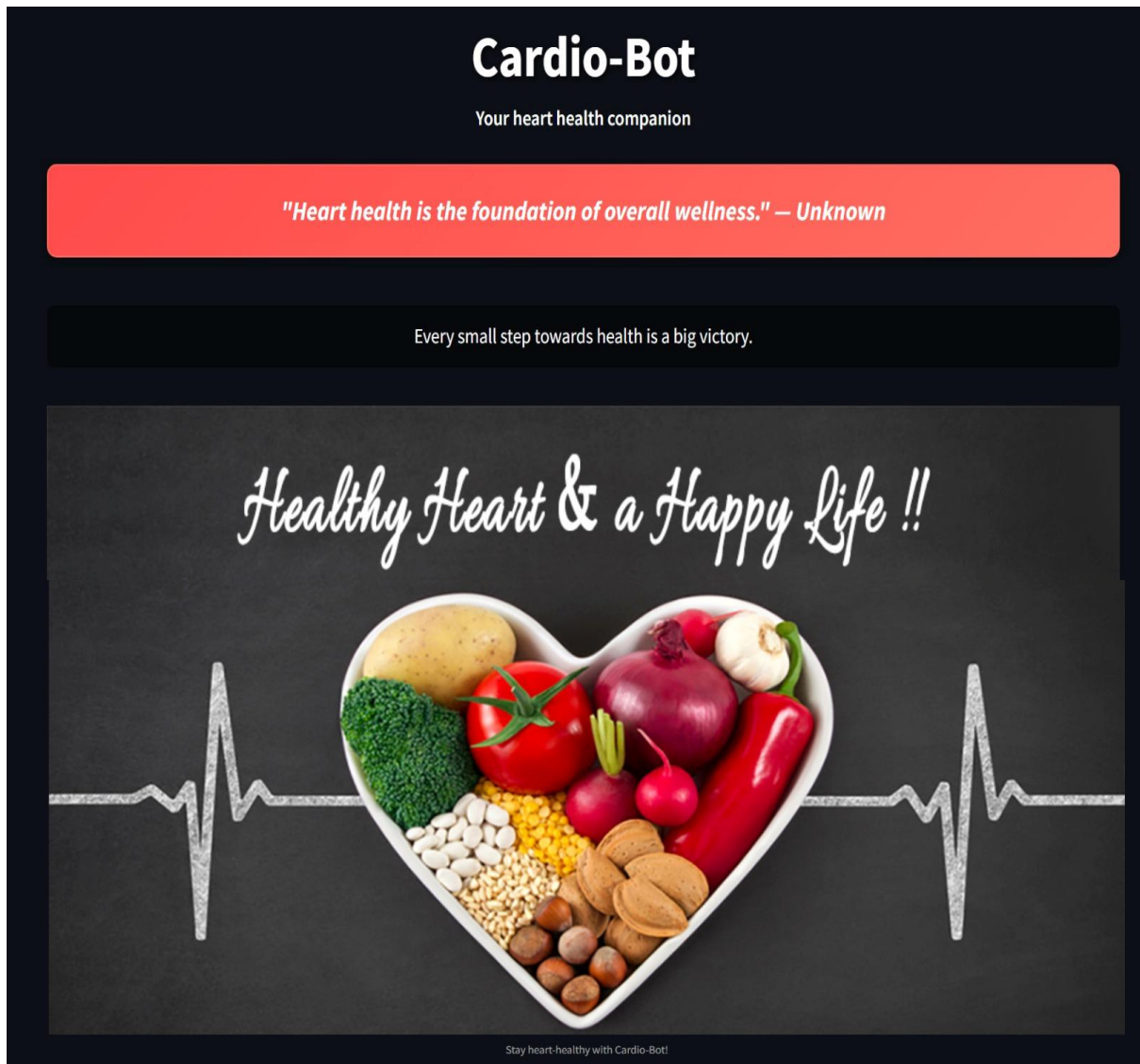
# Clean up

pdf_document.close()

print("Resources cleaned up successfully!")

```

## OUTPUT





## Chat with Cardio-Bot

Can you suggest a heart-healthy diet plan?

Sure! Here is a suggested heart-healthy diet plan:

### Heart-Healthy Diet Plan

#### Overview

A heart-healthy diet is essential for maintaining cardiovascular health and reducing the risk of heart disease. This diet plan emphasizes certain foods and nutrients that have been linked to a lower risk of heart disease and promotes overall health.

#### Guidelines

- **Fruits and Vegetables:** Aim for a variety of colorful fruits and vegetables, as they are rich in antioxidants, fiber, and nutrients that support heart health. Try to fill half your plate with produce at each meal.
- **Whole Grains:** Choose whole grain bread, pasta, rice, and cereals. Whole grains provide fiber and nutrients that can help lower cholesterol and improve heart health.
- **Lean Protein:** Include lean sources of protein such as fish, poultry, beans, and nuts. These provide essential amino acids without the high saturated fat content of fatty meats.
- **Healthy Fats:** Incorporate healthy fats like olive oil, avocado, nuts, and seeds. These contain monounsaturated and polyunsaturated fats, which can help raise HDL (good) cholesterol and lower LDL (bad) cholesterol.
- **Limit Unhealthy Fats:** Reduce your intake of saturated and trans fats, which are found in fatty meats, full-fat dairy products, and processed foods. These can raise LDL cholesterol and increase the risk of heart disease.
- **Reduce Sodium Intake:** Limit your sodium

Thank you for reaching out! Stay heart-healthy and keep smiling! 😊

## Chat with Cardio-Bot

does acidity affect heart ?

Yes, acidity can affect the heart. High levels of acid in the body, often due to dietary choices or certain medical conditions, can have both direct and indirect effects on cardiovascular health. Here's how acidity can impact the heart:

#### 1. Direct Effect on Cardiomyocytes (Heart Cells):

- Acidity, or a high acid load in the body, can directly influence the function of cardiomyocytes. Research suggests that an acidic environment can impair the contractile function of these heart cells, reducing their ability to contract effectively. This can lead to decreased cardiac output and potentially contribute to heart failure over time.

#### 2. Indirect Effects through Electrolyte Imbalances:

- Acidity can disrupt the balance of electrolytes in the body, particularly affecting potassium and magnesium levels. These electrolytes are crucial for maintaining normal heart rhythm. Imbalances can increase the risk of arrhythmias (irregular heartbeats), which can be dangerous and even life-threatening.

#### 3. Impact on Blood Vessels:

- High acidity can contribute to vascular inflammation and endothelial dysfunction, which is the impairment of the inner lining of blood vessels. This can lead to a reduction in vascular flexibility and an increase in peripheral vascular resistance, making it harder for the heart to pump blood efficiently.

#### 4. Contribution to Metabolic Disorders:

- Acidity is often associated with metabolic disorders such as obesity, insulin resistance, and type 2 diabetes. These disorders are independent risk factors

Thank you for reaching out! Stay heart-healthy and keep smiling! 😊

## Chat with Cardio-Bot

What role does sodium play in heart health?

Sodium is an essential mineral that plays a crucial role in maintaining heart health. Here are some key points explaining the role of sodium in heart health:

- **Fluid Balance:** Sodium helps regulate fluid balance in the body. It attracts and retains water, influencing the distribution of fluids inside and outside the cells. This fluid balance is critical for maintaining normal blood volume and blood pressure.
- **Blood Pressure Regulation:** Sodium intake directly impacts blood pressure regulation. A high-sodium diet can lead to increased blood pressure, known as hypertension. Hypertension is a significant risk factor for cardiovascular diseases, including heart attacks, strokes, and heart failure. Reducing sodium intake can help lower blood pressure and reduce these risks.
- **Electrolyte Function:** Sodium is an electrolyte, which means it carries an electric charge when dissolved in body fluids. Electrolytes, including sodium, are vital for nerve conduction and muscle contraction, including the heart muscle. They help generate electrical impulses that keep the heart beating rhythmically.
- **Cell Function:** Sodium is necessary for maintaining the balance of other electrolytes, such as potassium, inside and outside cells. This balance is crucial for proper cell function, including the cells of the heart. It also plays a role in nutrient absorption and the transport of certain substances across cell membranes.
- **Volume Expansion:** In cases of low blood volume or dehydration, intravenous sodium solutions (saline) are often used to expand blood volume and improve circulation. This is particularly

**Thank you for reaching out! Stay heart-healthy and keep smiling! 😊**

## Chat with Cardio-Bot

Are there specific exercises beneficial for cardiovascular health?

Yes, there are many exercises that can be beneficial for cardiovascular health. Here are some examples:

1. **Aerobic exercise:** Aerobic exercise, also known as cardio, is any type of physical activity that increases the heart and lung capacity and improves the efficiency of oxygen delivery to the working muscles. Examples include walking, jogging, running, swimming, cycling, and rowing. Aim for at least 150 minutes of moderate-intensity aerobic exercise or 75 minutes of vigorous-intensity aerobic exercise per week.
2. **High-intensity interval training (HIIT):** HIIT involves alternating periods of intense exercise with periods of rest or low-intensity exercise. This type of training can improve cardiovascular fitness, increase calorie burn, and provide time-efficient workouts. Examples include sprint intervals, Tabata protocols, or circuit training.
3. **Strength training:** Strength training helps build and maintain muscle mass, which is important for overall health and can also improve cardiovascular function. Try bodyweight exercises like push-ups, lunges, and squats, or use resistance bands, free weights, or weight machines. Aim for 2-3 sessions per week.
4. **Swimming:** Swimming is a full-body aerobic exercise that increases heart and lung capacity while putting minimal stress on the joints. It can improve cardiovascular health and endurance.
5. **Cycling:** Cycling is another low-impact aerobic exercise that can improve cardiovascular fitness and endurance. It can be done outdoors or on a stationary bike.

**Thank you for reaching out! Stay heart-healthy and keep smiling! 😊**

## APPLICATIONS

The Cardio Bot project has a wide range of real-world applications in both the healthcare industry and patient support systems. One of its primary uses is in **medical education**, where students, practitioners, and healthcare professionals can use it to understand complex cardiovascular concepts extracted directly from medical literature, research papers, and diagnostic reports. It also serves as an **intelligent assistant for patients**, enabling them to ask natural language questions about heart health, symptoms, treatment options, medications, and lifestyle changes. This can help reduce misinformation and provide consistent, reliable information based on real documents. In **clinical settings**, Cardio Bot can be used by doctors and nurses as a reference tool that quickly summarizes or interprets large medical PDFs or patient reports, reducing the time needed to manually read through dense documents. The system can also be integrated into **telemedicine platforms** or mobile health apps to deliver on-demand, context-aware insights into cardiovascular conditions. Moreover, it can aid in **research environments** by helping researchers retrieve and summarize relevant information across large corpora of cardiovascular literature, accelerating the pace of innovation. With further development, Cardio Bot could be incorporated into **public health initiatives** to spread awareness about heart health through interactive, AI-powered platforms. Overall, this tool demonstrates how artificial intelligence and natural language processing can revolutionize access to critical healthcare information, making medical knowledge more approachable, understandable, and actionable for everyone.

## CONCLUSION

The development of Cardio Bot marks a significant step forward in the integration of artificial intelligence with healthcare knowledge systems. By leveraging advanced natural language processing techniques, PDF parsing tools, semantic search through vector embeddings, and language generation models like Cohere and LLaMA, Cardio Bot transforms complex and technical cardiovascular literature into an interactive, user-friendly experience. It bridges the gap between vast medical information and the everyday user or healthcare professional by offering accurate, context-aware responses drawn directly from reliable sources such as academic papers or clinical documents. The bot's ability to extract, chunk, and intelligently interpret content from PDFs allows it to serve as a robust support system in medical education, patient self-care, and professional diagnostics. Its real-time answering capability empowers users with quick access to vital knowledge, thereby enhancing awareness, diagnosis, and treatment guidance in heart health. Furthermore, its modular and scalable design opens avenues for future enhancements, including multilingual support, voice integration, and incorporation of real-time data analytics. In conclusion, Cardio Bot is more than just a question-answering tool; it is a powerful, AI-driven assistant designed to democratize healthcare information and support better decision-making in the field of cardiovascular care.