ÖZYEĞİN UNİVERSİTY
FACULTY OF ENGINEERING

SYMPTOM TO DISEASE – DISEASE TO GENE BOT

AHMET GENCO GÜVEN
S029392

i. Problem Statement

The specific problem being worked on during the internship at Acıbadem University was the development of a chatbot with two distinct functions: retrieving the relevant disease when a symptom is given and retrieving the effective genes and bacteria when a specific disease is provided by the user. In the current era, individuals are less inclined to seek medical attention for health concerns not perceived as critical. Factors such as inaccessibility, cost, and time constraints may contribute to this tendency. The first function of the chatbot is designed to guide users in determining whether to seek professional medical assistance. Initial information is provided when symptoms are not perceived as serious or when there is uncertainty about visiting a doctor. This chatbot aims to enhance user health awareness by equipping them with knowledge about common symptoms and associated diseases, potentially preventing more serious health issues through early diagnosis and treatment. Moreover, the chatbot's accessibility provides global health support, as it can be accessed anytime and from any location, serving a diverse user base. The second function of the chatbot offers significant benefits by providing information about the most effective genes and bacteria for specific diseases. This knowledge aids professionals in refreshing their memory and providing more robust feedback about medical tests. Additionally, genetic and microbial data serve as valuable resources for researchers, contributing to clinical trials and providing educational material for medical students and healthcare professionals.

Several constraints had to be addressed, such as the websites used for data extraction. Access to an organization’s database requires an 'API key,' which is often provided only to large companies. Consequently, data extraction had to rely on information available on websites rather than directly from the organization’s database. This approach limits access to detailed information and results in the retrieval of unnecessary data for large language models, with the process of filtering out irrelevant data presenting a significant challenge.

This is not the first time a chatbot is being developed at this research institute since the computer engineering team has been working on a way more comprehensive and well-equipped chatbot about bacteria and oil relations for about 2 months. This chatbot utilizes PubMed’s database for information extraction and includes features such as a reranking algorithm for retrieving relevant information, a feedback mechanism for fine-tuning answers, and a filtering algorithm to avoid unrelated information. The 20-day internship period was insufficient to add such features into the project, as the learning curve and implementation process require deeper knowledge and experience. Future projects will aim to contain these features, aligned with the enthusiasm and experience gained during the internship.

While there are examples of similar chatbots around the world, they are not fully customizable. Users cannot input their own queries and must choose from a limited set of options, restricting the chatbot’s versatility. "Gastrobot," developed by a Chinese company in 2024, is an example of a highly meticulous and efficient health bot but is limited to gastrointestinal diseases. Another example is "DxGPT," produced by an organization called "Foundation 29." This chatbot effectively lists diseases related to the entered symptom but responds in an informative rather than conversational manner, has a somewhat long response time, and has a minimum character entry limitation that reduces its usefulness.

ii. Tools and Techniques Used

A lot of software and techniques were utilized in this project. Some of them were key elements, while others were chosen for their advantages over alternatives. With the assistance of colleagues, research was conducted to find the most accurate software and techniques for the project.

Python was selected as the programming language due to its straightforwardness, adaptability, and extensive range of libraries and frameworks. Its readability made it an optimal choice for integrating various APIs. Additionally, strong community support and comprehensive documentation facilitated troubleshooting and skill acquisition, which was particularly beneficial given the lack of software experience at the project's start. However, Python's slower speed compared to C++ and Java, as an interpreted language, was noted but did not significantly impact the project.

The "Flask" framework was chosen for its simplicity and ability to facilitate rapid development. It provided flexibility and control over web application construction, which is often lacking in larger frameworks like Django, without the added complexity. Its micro-framework nature allowed for the creation of applications that were straightforward to understand and maintain, offering a simple yet effective solution for the project.

The Google Custom Search API was used to source relevant medical data from reputable sources such as Mayo Clinic and MedlinePlus. This API allows for targeted searching within specific domains, ensuring high-quality information retrieval. Its ability to search entries provided by users on selected websites was fundamental to the system's functionality. However, more efficient data retrieval could have been achieved with direct access to medical companies' databases via an API key. Due to security concerns, most companies do not provide an API key, necessitating data extraction from websites using the Google Custom Search API. Direct database access would have facilitated the collection of more detailed and organized data, which would have been more beneficial for the large language model's responses.

The requests library was utilized as a crucial component for making HTTP requests to Google Custom Search and retrieving necessary data for user responses. It streamlined the process of sending and receiving web data, aiding the integration of external APIs into the project.

During the implementation of the large language model, the Hugging Face Hub was employed. It provided a comprehensive range of pre-trained models for various tasks, reducing training and fine-tuning time. The Inference API offered easy access to these models, making it an optimal choice for generating chatbot responses with natural language processing. The "Zephyr-7b-beta" model was used for its capability to generate comprehensive and natural language responses that closely mirrored a medical professional's style and tone. This model handled complex medical prompts and was adaptable for identifying symptoms, diseases, genes, and bacteria. Its manageable size allowed for local execution and testing.

API integration techniques were employed throughout the project to retrieve and process data from online sources, including Google and Hugging Face. This approach was instrumental in building the project, enabling access to real-time, reliable data and enhancing the application's modularity and maintainability.

To enhance the chatbot's functionality, Natural Language Processing (NLP) techniques were leveraged to generate text responses based on user input, fostering engaging and scalable interactions. During this stage, the prompt of the large language model was refined to ensure that responses reflected a professional tone, thereby providing a more reliable and straightforward user experience.

iii. Detailed Explanation

During the initial phase of the internship, a series of introductory and intermediate Python exercises were assigned to gain a fundamental understanding of coding and the industry. Upon satisfactory completion of these exercises, the chatbot project was assigned. A comprehensive account of the methodology and tools employed in bringing this project to life will now be provided.

The objective of the chatbot should first be briefly recalled. The initial phase of the chatbot serves as a symptom-based directory of potential diseases. Its purpose is to provide individuals with convenient access to health information when seeking medical advice is not possible for various reasons. Additionally, it assists individuals in taking necessary precautions and alleviating their concerns. The second part of the chatbot, which lists bacteria and genes related to the entered disease, is designed for use by healthcare professionals and health students. Both parts are fast, organized, and not overly detailed, making the chatbot practical and user-friendly.

The code file starts with the following lines:

```
from flask import Flask, render_template, request
import requests
from huggingface_hub import InferenceClient
```

This section is dedicated to the importing of the necessary libraries and modules.

- The "Flask" library is used to import "Flask" for the development of web applications, "render_template" for rendering HTML files, and "request" for receiving form data from users.
- The "requests" library is imported to facilitate the sending of HTTP requests via the Google Custom Search API.

- The “InferenceClient” class is imported from the “huggingface_hub” library, enabling the execution of the language model through the Hugging Face API.

```
hf_token = "FBJcG427fh4zefbf45JIcd7h3cRb6algMf89"
google_api_key = "Pc7b8JuIOd4g5hj3feK98C32ydfr65acabr"
google_cse_id = "jGıSS42NfKLI4FDse4t"
```

This section defines the application programming interface (API) keys.

- The "hf_token" is defined as the access key required to utilize the large language model on the Hugging Face platform.
- The "google_api_key" is defined as the API key required to search Google using the Google Custom Search API.
- The "google_cse_id", the Google Custom Search Engine ID, is defined to enable custom searches on specific websites.

```
def search_google(query):
    print(f"Searching Google for: {query}")
    base_url = "https://www.googleapis.com/customsearch/v1"
    medical_query = f"{query} site:medlineplus.gov OR
site:mayoclinic.org OR site:nih.gov OR site:webmd.com OR
site:healthline.com OR site:genecards.org OR site:www.ncbi.nlm.nih.gov
OR site:nhs.uk"
    params = {
        "key": google_api_key,
        "cx": google_cse_id,
        "q": medical_query
    }
    response = requests.get(base_url, params=params)
    if response.status_code == 200:
        data = response.json()
        if "items" in data:
            return [item["snippet"] for item in data["items"]]
        else:
            print("No search results found.")
            return []
    else:
        print(f"Error: {response.status_code}")
        return []
```

- “search_google”: This function is used for searching a specific symptom or disease across a large network of health websites. The "query" represents the keywords to search for.
- “base_url”: The base URL of the Google Custom Search API.
- “medical_query”: A query specifying which sites to search for the entered symptom or disease. The most well-equipped and trusted health websites have been carefully selected to provide the best possible resources.

- “params”: Parameters sent to the API. The "key" represents the API key, "cx" is the Google Custom Search Engine ID, and "q" is the search query.
- “requests.get()”: An HTTP GET request is sent to the Google API, and the response is assigned to the "response" variable. The status code of the HTTP request is checked with "response.status_code". If successful (status code 200), the JSON format response is translated into a Python dictionary, and a short summary of the results is returned. If no results are found, a “No search results found” message is printed. If the HTTP request is unsuccessful, an error message with the status code is returned.

```
def search_mayoclinic(query, site="mayoclinic.org/diseases-
conditions/"):
    print(f"Searching Mayo Clinic for: {query}")
    base_url = "https://www.googleapis.com/customsearch/v1"
    params = {
        "key": google_api_key,
        "cx": google_cse_id,
        "q": f"{query} site:{site}"
    }
    response = requests.get(base_url, params=params)
    if response.status_code == 200:
        data = response.json()
        if "items" in data:
            return [item["snippet"] for item in data["items"]]
        else:
            print("No results found.")
            return []
    else:
        print(f"Error: {response.status_code}")
        return []
```

This code is similar to the previous one, but it searches for symptoms and diseases only at "mayoclinic.org/diseases-conditions/”. The rationale for this approach will be discussed later.

```
def search_disease(symptoms):
    mayoclinic_results = search_mayoclinic(symptoms)
    google_results = search_google(symptoms)

    mayoclinic_length = sum(len(item) for item in mayoclinic_results)
    if mayoclinic_length < 500:
        return "No diseases are found for this symptom."
    return generate_chatbot_response(" ".join(google_results),
symptoms)
```

- “search_disease”: This function searches for diseases based on symptoms.

- The “mayoclinic_results” variable contains the results of a search conducted on the Mayo Clinic website, while the “google_results” variable contains the results from searches on selected websites.
- “sum()”: This function sums the character length of the Mayo Clinic results. If this length is less than 500, it indicates insufficient information and returns the message “No disease found for this symptom.” When the Google API searches a keyword on a website, it returns relevant words. However, on most websites, even if the search is unrelated to medicine, users are often directed to medical-related terms with messages like "Did you mean?". This situation does not occur on the Mayo Clinic website, where the absence of relevant words is explicitly indicated.
- Although a more stable and reliable filtering mechanism could have been developed for the project, the internship duration was insufficient to learn and implement such mechanisms. Consequently, this filtering mechanism was developed.

```
def generate_chatbot_response(combined_abstracts, symptom):
    try:
        client = InferenceClient(model="HuggingFaceH4/zephyr-7b-beta",
token=hf_token)
        prompt = (
            f"Using the following abstracts, provide a detailed
explanation about the top 5 most frequent diseases "
            f"that this symptom '{symptom}' can cause without stating
that they are the top 5 in frequency. Don't talk about what caused this
symptom '{symptom}'. Explain how '{symptom}' can be a symptom of these
diseases "
            f"its possible impact on the patient's quality of life.
Briefly provide guidance or advice on what they might consider or do
next. "
            f"Do not mention the abstracts or articles themselves.
Focus on giving information about the top 5 most frequent diseases in a
conversational manner without stating that you will discuss these
diseases.\n\n"
            f"{combined_abstracts}\n\n"
            "Summarize the key findings and provide a patient-centered
response that connects the symptom to the diseases described without
stating that they are a a patient-centered response "
            "Avoid mentioning the articles or abstracts. Focus on the
diseases and provide useful information in a friendly and approachable
tone."
        )

        response = client.text_generation(prompt=prompt,
max_new_tokens=1000)
        return response['generated_text'] if 'generated_text' in
response else response
    except Exception as e:
        return f"Error: {e}"
```

This part generates the chatbot response based on the symptoms entered via the Hugging Face API.

- "InferenceClient()": This is used for interacting with the Hugging Face language model. In this example, the "Zephyr-7B-beta" model is used for text generation. The function generates text based on the specified prompt.
- The prompt is designed to generate a user-friendly response that describes the five diseases associated with the entered symptom, explains their impact on the patient's quality of life, and offers guidance on further actions for the user.
- "max_new_tokens=1000": This parameter limits the length of the generated response to a maximum of 1,000 tokens (words or fragments). This approach ensures that the entire response generated by the chatbot is visible to the user.
- "response": The response returned from the Hugging Face API is in a dictionary structure. The key ,"generated_text", contains the text generated by the language model. If this key is present in the response, the text is returned. If the "generated_text" key is not present, the entire response is returned directly, allowing for the identification of any errors or missing responses by returning all the data.

```
def search_disease_genes(disease):
    mayoclinic_results = search_mayoclinic(disease)
    google_results = search_google(disease)

    mayoclinic_length = sum(len(item) for item in mayoclinic_results)
    if mayoclinic_length < 500:
        return "No diseases are found for this symptom."
    return generate_chatbot_response_genes(" ".join(google_results),
disease)
```

This part is largely similar to "def search_disease(symptoms):", with the exception that it is used in the context of searching for a disease, rather than a symptom. It was written as a different function, with the potential for a change in the search mechanics.

```
def generate_chatbot_response_genes(combined_abstracts, disease):
    try:
        client = InferenceClient(model="HuggingFaceH4/zephyr-7b-beta",
token=hf_token)
        prompt = (
            f"Using the following abstracts, provide a detailed list of
all the genes and bacteria related to the disease '{disease}'. "
            f"Include the following in your response:\n"
            f"- A list of top 5 most relevant genes associated with the
disease, with a brief description of their role.\n"
```

```
            f"- A list of top 5 most relevant bacteria related to the
disease, with a brief explanation of their impact.\n"
            f"Briefly explain how these genes and bacteria could be
connected to this disease and briefly discuss their potential impact on
the patient's health. "
            f"Focus on providing a comprehensive and clear list of
information.\n\n"
            f"{combined_abstracts}\n\n"
            "Summarize the key findings and provide a detailed list of
genes and bacteria associated with the disease. "
            "Do not mention the abstracts or articles themselves.
Concentrate on delivering a complete list of relevant genes and
bacteria."
        )
        response = client.text_generation(prompt=prompt,
max_new_tokens=1000)
        return response['generated_text'] if 'generated_text' in
response else response
    except Exception as e:
        return f"Error: {e}"
```

This section is comparable to the previous text generation component. However, this prompt is intended for a response that outlines the five most critical genes and five bacteria associated with a disease, their function in the disease process, their interrelationship, and their potential impact on the patient's health.

```
@app.route('/', methods=['GET', 'POST'])
def home():
    chatbot_response = None
    if request.method == 'POST':
        symptoms = request.form.get('symptoms')
        disease = request.form.get('disease')
        if symptoms:
            chatbot_response = search_disease(symptoms)
        elif disease:
            chatbot_response = search_disease_genes(disease)
    return render_template('index3.html', response=chatbot_response)S
```

This code forms the core functionality of the Flask application. It receives symptoms or disease information from the user via the web page and generates a chatbot response accordingly.

- “if request.method == 'POST'”: This check determines whether there is a POST request to the page. If a symptom or disease is submitted by the user, this results in a POST request, and this code block will execute. If a GET request is made, this part is skipped.
- “request.form.get('symptoms')”: Retrieves the symptoms entered by the user in the form. If symptoms were entered, this value is stored here.

- “request.form.get('disease')”: Retrieves the disease entered by the user in the form. If a disease was entered, this value is stored here.
- “chatbot_response = search_disease(symptoms)”: If symptoms have been entered by the user, this line executes the “search_disease” function and assigns the results to the “chatbot_response” variable.
- “chatbot_response = search_disease_genes(disease)”: If a disease has been entered by the user, this line executes the “search_disease_genes” function and assigns the results to the “chatbot_response” variable.

```
if __name__ == '__main__':
    app.run(debug=True)
```

This part is the main run point for the program.

- app.run(debug=True): This line starts the Flask application. “debug=True:” indicates that the Flask application will run in development mode. In this mode, debug messages are automatically shown and the application is automatically restarted when a change is made to the code.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Symptom-Disease Bot</title>
    <style>
        body {
            font-family: 'Arial Unicode MS';
            background-color: #f0f4f8;
            color: #333;
            margin: 0;
            padding: 0;
            display: flex;
            flex-direction: column;
            align-items: center;
            height: 200vh;
        }

        .form-container {
            display: flex;
            justify-content: space-between;
            gap: 30px;
            flex-wrap: wrap;
            margin-bottom: 20px;
            max-width: 800px;
            width: 100%;
        }

        .form-wrapper {
            flex: 1;
```

```
            min-width: 300px;
            max-width: 400px;
            box-sizing: border-box;
            margin: 0 15px; /* Added margin to create space between
containers */
        }

        .container {
            background-color: #fff;
            box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
            border-radius: 8px;
            padding: 20px;
            text-align: center;
            margin: 20px 0;
            width: 70%;
        }

        h1 {
            font-size: 1.8em;
            margin-bottom: 15px;
        }

        form {
            margin-bottom: 20px;
        }

        label {
            display: block;
            font-weight: bold;
            margin-bottom: 10px;
        }

        input[type="text"] {
            width: 100%;
            padding: 12px;
            margin-bottom: 10px;
            border: 1px solid #ccc;
            border-radius: 4px;
            box-sizing: border-box;
            font-size: 1.1em;
        }

        button {
            color: #fff;
            border: none;
            padding: 12px 24px;
            font-size: 1.1em;
            cursor: pointer;
            border-radius: 4px;
            transition: background-color 0.3s;
```

```
        }

        .orange {
            background-color: #ff7b00;
        }

        .orange:hover {
            background-color: #d26600;
        }

        .blue {
            background-color: #007bff;
        }

        .blue:hover {
            background-color: #0056b3;
        }

        h2 {
            font-size: 1.5em;
            margin-bottom: 10px;
        }

        pre {
            background-color: #f8f9fa;
            border: 1px solid #e9ecef;
            padding: 15px;
            border-radius: 4px;
            white-space: pre-wrap;
            text-align: left;
        }
    </style>
</head>
<body>
    <div class="form-container">
        <div class="form-wrapper">
            <div class="container">
                <h1 class="orange">Symp-Dis Bot</h1>
                <form method="POST">
                    <label for="symptoms">Enter your symptoms:</label>
                    <input type="text" id="symptoms" name="symptoms"
required>
                    <button type="submit"
class="orange">Search</button>
                </form>
            </div>
        </div>
        <div class="form-wrapper">
            <div class="container">
                <h1 class="blue">Gene-Bacteria Bot</h1>
```

```
                <form method="POST">
                    <label for="disease">Enter the disease:</label>
                    <input type="text" id="disease" name="disease"
required>
                    <button type="submit" class="blue">Search</button>
                </form>
            </div>
        </div>
    </div>

    {% if response %}
    <div class="container">
        <h2>Bot's Response:</h2>
        <pre>{{ response }}</pre>
    </div>
    {% endif %}
</body>
</html>
```

This part is the HTML template that creates the user interface and integrates with the Flask app. This page allows users to enter symptom or disease information and shows the chatbot's response.

- “<!DOCTYPE html>”: Specifies the HTML5 standard.
- “<html lang="en"”>: Specifies that the language of the HTML document is English.
- “<meta charset="UTF-8"”>: Sets the charset of the page to UTF-8 so that different characters are displayed correctly.
- “<title>”Symptom-Disease Bot</title>: Sets the title that appears in the browser tab.
- “<style>”: Contains the CSS style information for the page.

For the general style settings:
- “Body”: Sets the basic style of the page like font, background color, color, margin, padding. The content is centered using flexbox and the height of the page is set to 200vh.
- The ".form-container" provides a containerized, flexible and extensible layout in which to place the forms, while the ".form-wrapper" defines the area around the form, limiting the width and providing spacing. ".container" determines the appearance of content boxes

Header and text styles:
- “h1, h2”: Sets the font size and margin for headings.

Form elements:
- “input[type="text"]”: Sets the style of text input fields
- “button”: Sets the style of buttons such as color, padding, border radius.

Form Container:
- “<div class="form-container">”: The main container in which forms are placed. It uses flexbox to arrange its contents horizontally.

- The “Symptom-Disease Bot” form receives symptom information from the user and sends a POST request. The “Gene-Bacteria Bot” form receives disease information from the user and sends a POST request.

Display the response:

- “{% if response %}”: If the response variable exists, the response will be displayed.
- “<div class="container">”: A container used to display the response. The response is displayed formatted within the “<pre>” tag.

The interface looks like this:

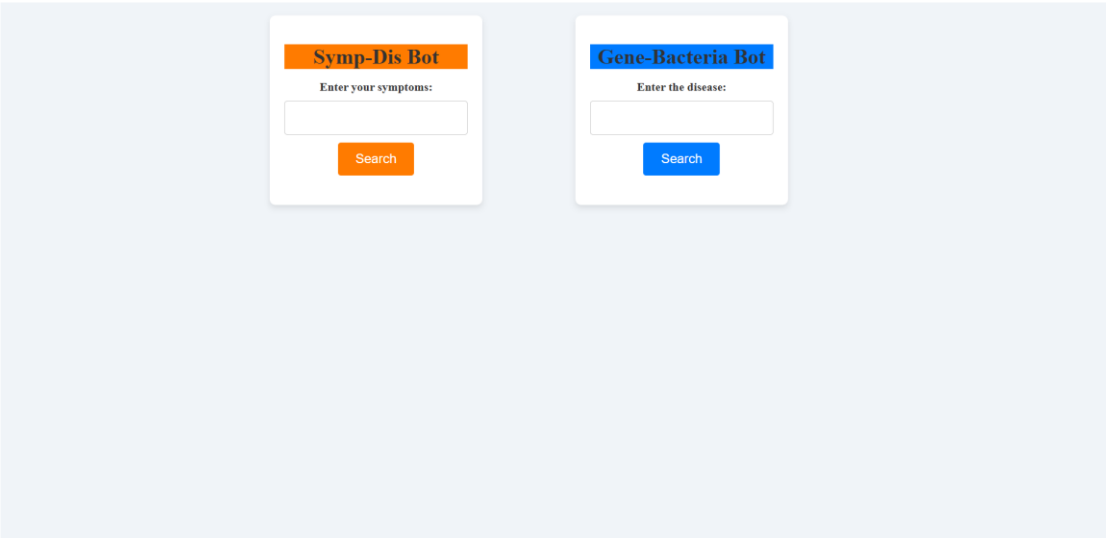

Figure 1: The interface of the chatbot.

iv. Results

It can be affirmed that the project was a success. The symptom-disease and disease-gene sections of the chatbot function precisely as intended by the supervisor. The responses are consistent, comprehensive, and, most importantly, dependable. A challenging objective was accomplished for someone relatively inexperienced in software development and with limited time. The guidance and assistance provided by supervisors and colleagues, who shared their insights and experiences, were invaluable in this process.

The figure above (Figure 1) shows the interface in its default state. When a question is posed, the interface is updated as shown below.

Bot's Response:

Chest pain can take many forms, from a sharp stab to a dull ache, and sometimes it feels crushing or burning. While chest pain can be a sign of a heart attack, it can also indicate other conditions. Here are five diseases that chest pain can be a symptom of:

1. Acid reflux: Also known as gastroesophageal reflux disease (GERD), acid reflux occurs when stomach acid flows back into the esophagus. This can cause a burning sensation in the chest, especially after eating. If left untreated, acid reflux can lead to esophagitis, which is inflammation of the esophagus.

2. Pleuritis: This is an inflammation or irritation of the lungs and chest lining. It can cause sharp pain when breathing, coughing, or sneezing. Pleuritis can be caused by infections, injuries, or underlying medical conditions.

3. Costochondritis: This is inflammation of the cartilage that connects the ribs to the breastbone. It can cause sharp, localized pain in the chest, especially when breathing deeply or coughing. Costochondritis is often mistaken for a heart attack, but it is not a serious condition.

4. Pulmonary embolism: This is a blockage in the lungs, usually caused by a blood clot. It can cause chest pain, shortness of breath, and coughing up blood. Pulmonary embolism is a serious condition that requires immediate medical attention.

5. Angina: This is chest pain or discomfort caused by a lack of oxygen to the heart. It can be triggered by physical activity, stress, or cold weather. Angina is a symptom of coronary artery disease, which is the buildup of plaque in the arteries that supply blood to the heart.

If you are experiencing chest pain, it's important to seek medical attention, especially if the pain is severe, persistent, or accompanied by other symptoms such as shortness of breath, sweating, or dizziness. Your healthcare provider can help determine the cause of your chest pain and provide appropriate treatment. In the meantime, you can try to manage your symptoms by avoiding triggers, such as spicy foods or stressful situations, and taking over-the-counter medications for pain and inflammation. Remember to always consult with your healthcare provider before starting any new treatments or making significant lifestyle changes.

Figure 2: Answer of the symptom-disease bot when "chest pain" is given as input.

Bot's Response:

Diarrhea is a common symptom that can be a sign of various underlying conditions. While it's often caused by viruses, it can also be a symptom of other diseases. Here are five diseases that you should be aware of:

1. Inflammatory Bowel Diseases (IBD): IBD is a chronic condition that affects the digestive system. It includes Crohn's disease and ulcerative colitis. Diarrhea is a common symptom of both conditions, and it can be severe and persistent. In addition to diarrhea, other symptoms may include abdominal pain, fatigue, and weight loss.

2. Irritable Bowel Syndrome (IBS): IBS is a functional gastrointestinal disorder that affects the large intestine. It's a chronic condition that causes abdominal pain, bloating, and diarrhea or constipation. The exact cause of IBS is unknown, but it's believed to be related to changes in the gut's motility and sensitivity.

3. Celiac Disease: Celiac disease is an autoimmune disorder that affects the small intestine. It's triggered by consuming gluten, a protein found in wheat, barley, and rye. Diarrhea is a common symptom of celiac disease, and it can be severe and persistent. Other symptoms may include bloating, gas, and weight loss.

4. Food Allergies: Food allergies can cause a range of symptoms, including diarrhea. Common food allergens include milk, soy, wheat, and peanuts. Symptoms may be mild or severe, and they can affect different parts of the body, including the digestive system.

5. Medications: Certain medications can cause diarrhea as a side effect. Antibiotics, laxatives, and chemotherapy drugs are common culprits. If you're taking medication and experiencing diarrhea, talk to your healthcare provider about alternative options.

If you're experiencing diarrhea, it's essential to stay hydrated and replace lost electrolytes. Drink plenty of water, broths, and sports drinks to prevent dehydration. Avoid caffeinated and alcoholic beverages, as they can worsen diarrhea. It's also essential to eat small, frequent meals and avoid spicy, fatty, and high-fiber foods. If your diarrhea persists for more than a few days, or if you're experiencing severe symptoms, it's essential to talk to your healthcare provider. They can help diagnose the underlying condition and provide appropriate treatment.

Figure 3: Answer of the symptom-disease bot when "diarrhea" is given as input.

Bot's Response:

Top 5 most relevant genes associated with schizophrenia:
1. CACNA1C: This gene encodes a subunit of a calcium channel that plays a crucial role in neuronal excitability and synaptic plasticity. Studies have shown that variations in this gene are associated with an increased risk of schizophrenia.
2. COMT: This gene encodes an enzyme that breaks down dopamine, a neurotransmitter linked to reward and motivation. Variations in this gene have been linked to differences in dopamine levels, which may contribute to the development of schizophrenia.
3. NR3C: This gene encodes a protein that regulates the stress response. Studies have shown that variations in this gene are associated with an increased risk of schizophrenia, particularly in individuals who have experienced childhood trauma.
4. ZNF804A: This gene encodes a transcription factor that plays a role in neuronal development and synaptic plasticity. Variations in this gene have been linked to an increased risk of schizophrenia, particularly in individuals with a family history of the disorder.
5. ANK3: This gene encodes a protein that is involved in the formation and maintenance of synapses. Variations in this gene have been linked to an increased risk of schizophrenia, particularly in individuals with a history of prenatal exposure to viruses.

Top 5 most relevant bacteria related to schizophrenia:
1. Lactobacillus: This bacterium is commonly found in the gut and has been linked to a reduced risk of schizophrenia. Studies have shown that individuals with schizophrenia have lower levels of Lactobacillus in their gut microbiome.
2. Prevotella: This bacterium is also commonly found in the gut and has been linked to an increased risk of schizophrenia. Studies have shown that individuals with schizophrenia have higher levels of Prevotella in their gut microbiome.
3. Faecalibacterium: This bacterium is involved in the production of short-chain fatty acids, which are important for gut health. Studies have shown that individuals with schizophrenia have lower levels of Faecalibacterium in their gut microbiome.
4. Bacteroides: This bacterium is involved in the breakdown of complex carbohydrates. Studies have shown that individuals with schizophrenia have higher levels of Bacteroides in their gut microbiome.
5. Ruminococcus: This bacterium is involved in the fermentation of dietary fibers. Studies have shown that individuals with schizophrenia have lower levels of Ruminococcus in their gut microbiome.

The connection between these genes and bacteria is not yet fully understood, but it is thought that they may contribute to the development of schizophrenia through their effects on brain function and gut health. For example, variations in genes involved in neuronal excitability and synaptic plasticity may lead to abnormal brain development, while changes in the gut microbiome may affect the production of neurotransmitters and inflammatory molecules that can impact brain function. Further research is needed to fully understand the role of these genes and bacteria in schizophrenia.

Figure 4: Answer of the gene-bacteria bot when "schizophrenia" is given as input.

Bot's Response:

Top 5 most relevant genes associated with ovarian cancer:
1. BRCA1: This gene is commonly mutated in individuals with hereditary breast and ovarian cancer. It plays a role in DNA repair and cell cycle regulation, and mutations in BRCA1 increase the risk of developing breast and ovarian cancer.
2. BRCA2: Similar to BRCA1, mutations in BRCA2 increase the risk of breast and ovarian cancer. BRCA2 is also involved in DNA repair and cell cycle regulation.
3. TP53: This gene is commonly mutated in various types of cancer, including ovarian cancer. It encodes a tumor suppressor protein that regulates cell growth and division, and mutations in TP53 can lead to uncontrolled cell growth and cancer development.
4. RAD51: This gene is involved in DNA repair and is frequently amplified in ovarian cancer. Amplification of RAD51 is associated with poor prognosis and resistance to chemotherapy.
5. FIGO: This gene is overexpressed in ovarian cancer and is associated with poor prognosis. It encodes a protein involved in cell division and migration, and overexpression of FIGO may contribute to tumor growth and metastasis.

Top 5 most relevant bacteria related to ovarian cancer:
1. Escherichia coli (E. Coli): This bacterium is commonly found in the gut and is associated with increased risk of ovarian cancer. A study found that women with ovarian cancer had higher levels of E. Coli in their ovarian fluid compared to healthy women.
2. Streptococcus gallolyticus (S. Gallolyticus): This bacterium is also found in the gut and has been associated with increased risk of ovarian cancer. A study found that women with ovarian cancer had higher levels of S. Gallolyticus in their ovarian fluid compared to healthy women.
3. Gardnerella vaginalis (G. Vaginalis): This bacterium is commonly found in the vagina and is associated with increased risk of ovarian cancer. A study found that women with ovarian cancer had higher levels of G. Vaginalis in their ovarian fluid compared to healthy women.
4. Prevotella bivia (P. Bivia): This bacterium is commonly found in the vagina and is associated with increased risk of ovarian cancer. A study found that women with ovarian cancer had higher levels of P. Bivia in their ovarian fluid compared to healthy women.
5. Atopobium vaginae (A. Vaginae): This bacterium is commonly found in the vagina and is associated with increased risk of ovarian cancer. A study found that women with ovarian cancer had higher levels of A. Vaginae in their ovarian fluid compared to healthy women.

The exact mechanisms by which these genes and bacteria contribute to ovarian cancer are not fully understood, but they may promote tumor growth and metastasis through various mechanisms, such as DNA repair, cell cycle regulation, and inflammation. Further research is needed to fully elucidate the roles of these genes and bacteria in ovarian cancer.

Figure 5: Answer of the gene-bacteria bot when "ovarian cancer" is given as input.

As illustrated in the figures above, upon receiving a symptom report from the user, the symptom-disease bot provides a brief explanation of the symptom and lists the five diseases with the highest degree of correlation. It provides information about all the listed diseases, including additional distinguishing symptoms. Finally, a recommendation is offered to users with these symptoms.

The Gene-Bacteria Bot initially identifies the five genes most closely associated with the disease in question based on the user's input. It then provides an explanation of the functions of these genes and their significance in the context of the disease. Following this, the bot lists the five bacteria most closely linked to the disease, indicating the location of the bacterium within the body and its relevance to the disease. The response concludes with a summary.

While the project has been a great success, there are still areas for improvement. A reranking algorithm can be implemented to sort the information received from the sites according to its relevance, resulting in more accurate and relevant information about the given symptom or disease. An algorithm could be constructed to pose questions such as "Did this answer help you?" after responses are provided, allowing for fine-tuning of the language model based on the provided answers. Incorporating images and graphics into the responses could enhance their comprehensibility for users. Further enhancements to the filtering mechanisms could enable the system to identify and disregard irrelevant inputs. Although there is always room for improvement in such projects, the time and knowledge gained during the internship were sufficient to achieve this level of development. With the experience and motivation gained, further development of the project is intended in the near future.