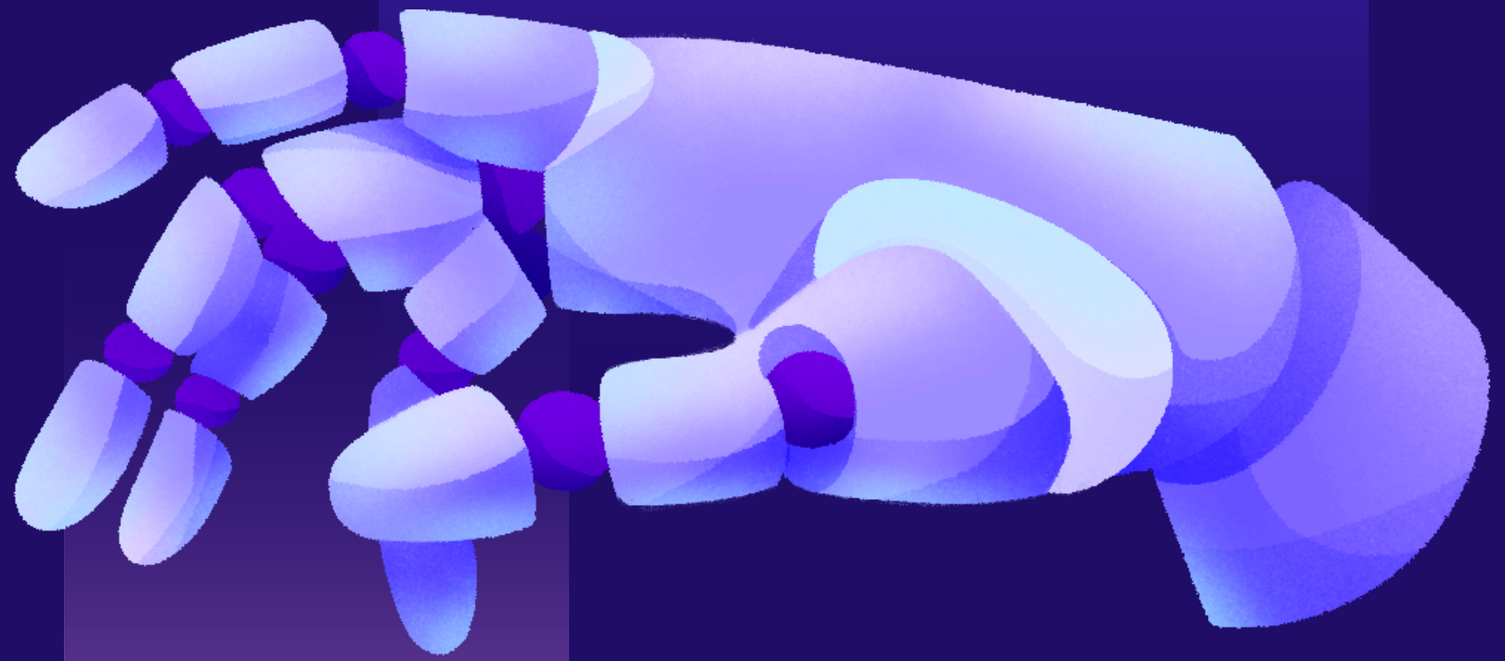
A RULE-BASED EXPERT SYSTEM FOR DIAGNOSING MULTIPLE DISEASES BASED ON UNIQUE SYMPTOMS



A Flask-Based Web Application Using Rule-Based Logic

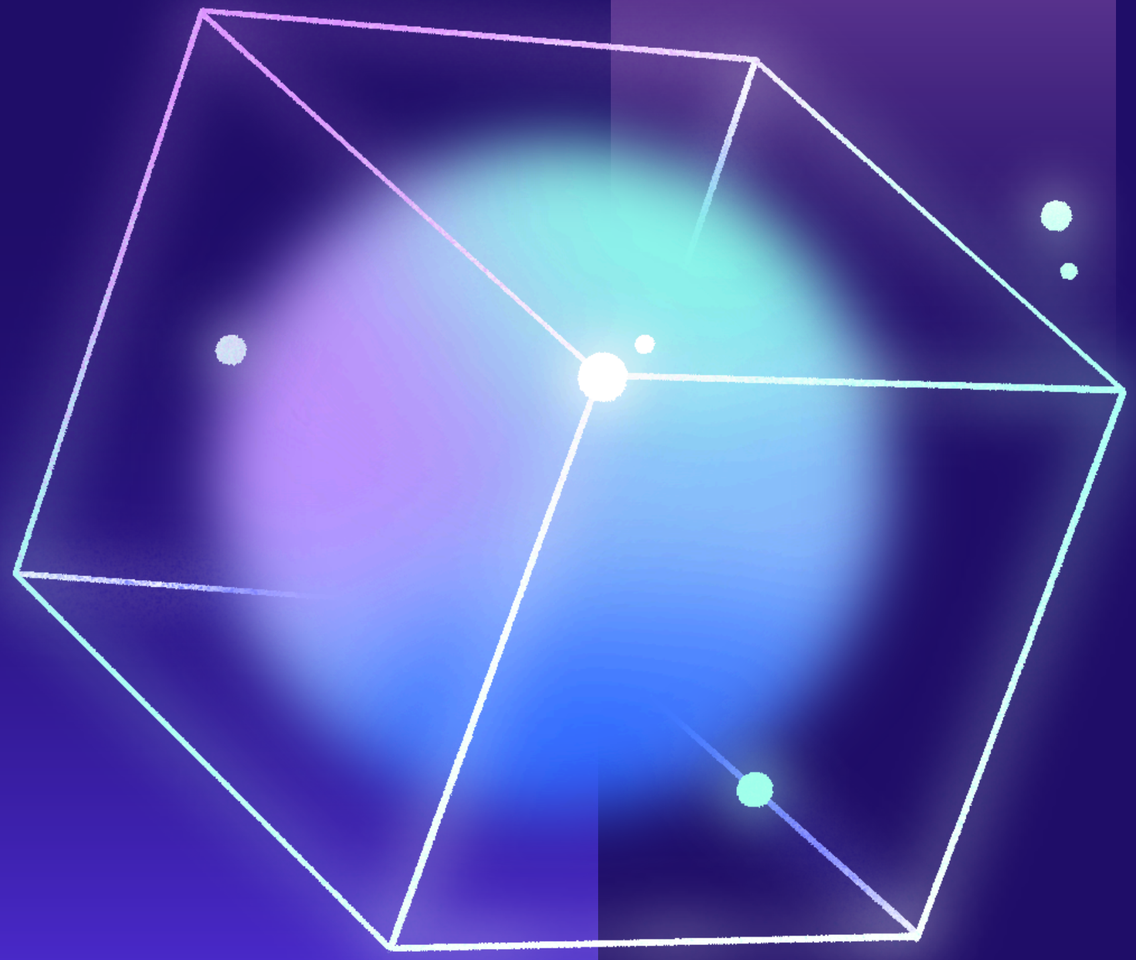
Muhammad Nadeem · Edward Kane

Department of Computer Science

Georgia State University

Built to incorporate key Artificial Intelligence Components

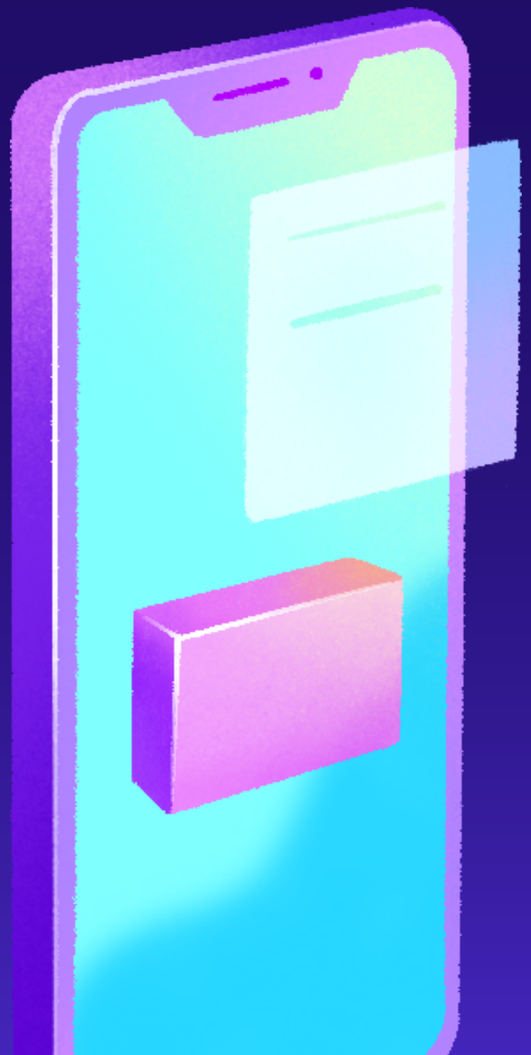
Presented on 21 April, 2025



INTRODUCTION

In today’s healthcare environment, timely and accurate disease diagnosis is crucial for providing effective treatment and improving patient outcomes. This project introduces a web-based expert system built with Flask that uses rule-based logic to help users identify potential illnesses from their symptoms. Unlike traditional systems that require clinical testing or in-person evaluation, this approach leverages **Logical Reasoning** and expert-defined symptom sets to simulate how a medical expert would assess symptoms.

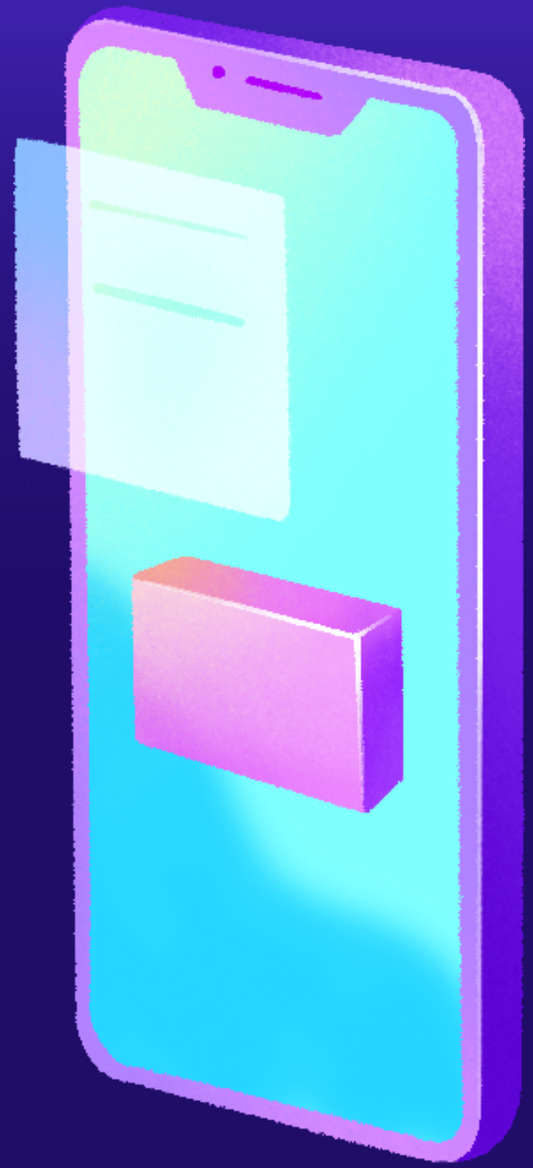
These systems fall under the branch of Artificial Intelligence known as Expert Systems, which are built to emulate human decision-making using knowledge bases and inference engines. By integrating this technology, we aim to support preliminary diagnosis, especially in areas with limited access to healthcare professionals or in situations requiring quick, remote evaluations.



PROBLEM STATEMENT

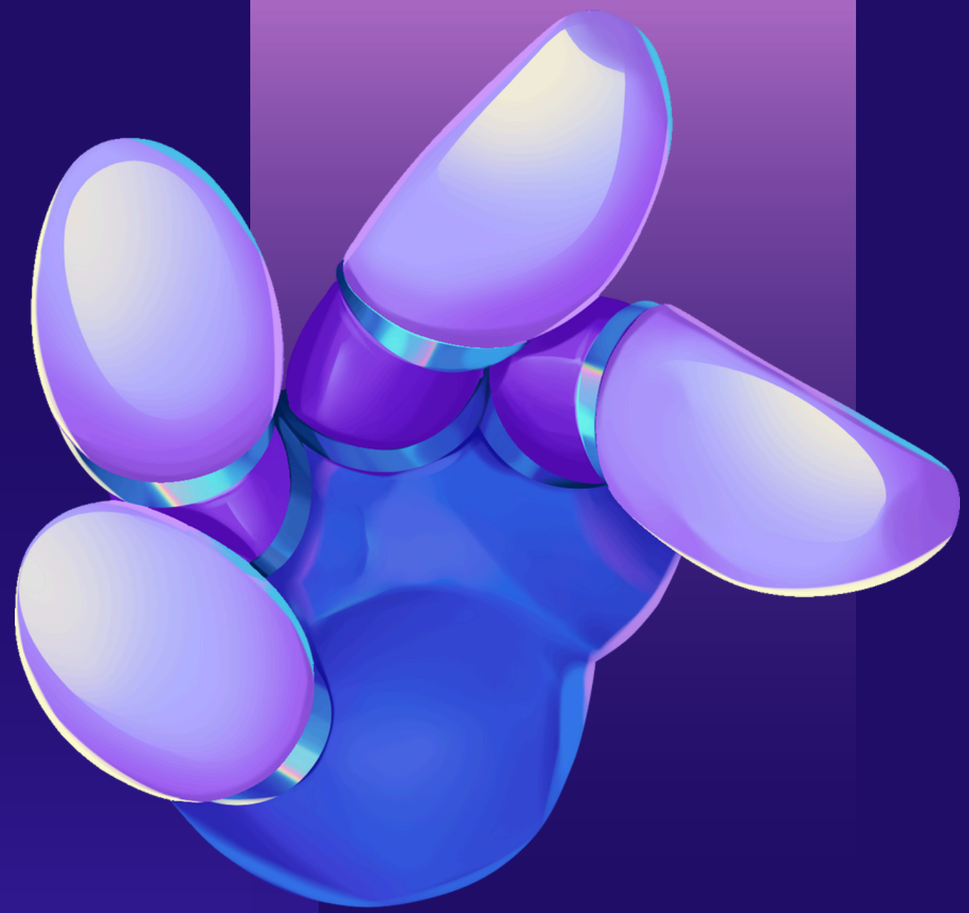
Diagnosing illnesses that share similar symptoms such as the Flu, COVID-19, or the Common Cold can be extremely challenging. This symptom overlap often leads to misdiagnosis, delayed treatment, and increased health risks. The situation is even more difficult in communities with limited access to healthcare professionals, where quick, accurate evaluations are essential.

To address this, we developed a rule-based expert system that uses non-overlapping symptom sets to identify diseases with greater precision. By ensuring each disease is characterized by unique symptoms, the system improves diagnostic accuracy and offers a valuable tool for preliminary medical decision-making. This foundation supports future enhancements like web deployment, wearable tech integration, and smart data analysis.



PROJECT OBJECTIVES

- **Develop an Expert System:** Design and implement a rule-based system that uses unique, non-overlapping symptom sets to accurately diagnose a wide range of common diseases.
- **Improve Diagnostic Accuracy:** Reduce symptom confusion and eliminate misclassification by ensuring each disease has its own clearly defined symptom profile, enhancing system reliability.
- **Enhance Accessibility:** Provide a lightweight, easy-to-use diagnostic tool that can be accessed in remote or underserved areas, where timely access to healthcare professionals is limited.
- **Optimized Decision Making:** Support users in identifying possible illnesses early, helping them make informed decisions about when and how to seek professional medical care.



METHODOLOGY – SYSTEM FLOW

Phase 01

System Design using Disease Mapping:

- Defined non-overlapping symptoms for each illness
- Each disease has a unique symptom set (no duplication)
- Built foundation for clear, rule-based logic

Phase 02

Rule Development & Implementation:

- Created rules for matching user-input symptoms
- Used a three-symptom threshold to confirm diagnosis
- Prevents misclassification from minor overlaps or vague inputs

Phase 03

System Validation & Testing:

- Ran test cases with simulated symptom combinations
- Verified each rule returned consistent diagnoses
- Ensured robustness and reliability with varied user inputs

SYSTEM ARCHICTECTURE

Knowledge Base Component: The knowledge base contains a dictionary of diseases and their corresponding unique symptom sets. Each disease is carefully defined with non-overlapping symptoms to allow for clear and accurate classification. This database serves as the core reference for diagnosis.

Inference Engine Component: The inference engine acts as the system's logic processor. It compares user-reported symptoms to the knowledge base and applies a three-symptom threshold rule. If three or more symptoms match a disease, the system outputs a confident diagnosis; otherwise, it returns "Diagnosis Unclear".

Component Interaction: Together, the knowledge base and inference engine simulate the reasoning process of a human expert. This layered structure allows the system to perform fast, explainable, and repeatable diagnoses, forming the backbone of the expert system's functionality.

SYSTEM WORKFLOW SUMMARY

◆ **USER INPUT:**

- USERS TYPE SYMPTOMS INTO A TEXT FIELD
- INPUT IS COLLECTED AND SENT TO THE SYSTEM

◆ **RULE ENGINE:**

- MATCHES SYMPTOMS WITH PRE-DEFINED DISEASE RULES
- USES A 3-SYMPTOM THRESHOLD FOR ACCURACY
- APPLIES CLEAR, EXPLAINABLE LOGIC (NOT BLACK-BOX AI)

◆ **DIAGNOSIS OUTPUT:**

- DISPLAYS MATCHING DISEASE AND MATCHED SYMPTOMS
- GIVES INSTANT FEEDBACK BASED ON THE RULE SYSTEM

CODE WALKTHROUGH

🖥️ 1. Platform Details

- Built using HTML + CSS + Flask (Python)
- Runs locally in a browser — no installation needed
- Fully responsive and works on any modern system

🧠 2. How It Works

- Users type symptoms into a simple input box
- On clicking “Diagnose,” the system checks against the rule base
- The matched disease is displayed instantly

👨‍💼 3. Design Philosophy

- Clean, minimal, easy to use for non-tech users
- No machine learning or complex forms
- Transparency is the focus — users see what’s happening

Full Project Breakdown: Part 1 – Backend & Templates

🧠 app.py – Flask Backend

- Controls all routing (home, diagnose, about, future)
- Hosts the DiagnosisExpertSystem logic
- Connects the symptom form to backend rule matching

🖼️ HTML Templates – Frontend Pages

- home.html → Main intro and Try It Now link
- index.html → Symptom entry & diagnosis display
- about.html → Course/project explanation
- future.html → Long-term vision for AI

Full Project Breakdown: Part 2 – Styling & Docs

🎨 style.css – Unified Styling

- One file used across all pages
- Handles layout, color schemes, responsiveness
- Button animations, hover effects, and visual polish

📂 Other Files

- navbar.html – Reusable header navigation
- README & User Manual – Setup instructions
- IEEE Paper – Project documentation & architecture

Code Demo:

```python
class DiagnosisExpertSystem:
    def __init__(self):
        # Each disease has **unique symptoms** with no overlap between diseases
        self.diseases = {
            'Flu': ['fever', 'body ache', 'fatigue', 'chills', 'sore throat'],
            'Common Cold': ['runny nose', 'sore throat', 'mild cough', 'sneezing'],
            'COVID-19': ['dry cough', 'loss of taste or smell', 'shortness of breath'],
            'Strep Throat': ['swollen lymph nodes', 'red spots on the roof of mouth', 'painful swallowing'],
            'Sinus Infection': ['sinus pressure', 'headache', 'nasal congestion', 'facial pain'],
            'Pneumonia': ['chest pain', 'coughing up mucus', 'shortness of breath'],
            'Allergies': ['itchy eyes', 'sneezing', 'nasal congestion', 'watery eyes'],
            'Bronchitis': ['persistent cough', 'wheezing', 'chest discomfort'],
            'Mononucleosis': ['swollen lymph nodes', 'extreme fatigue', 'sore throat']
        }

    def diagnose(self, symptoms):
        # If only one symptom is entered, find diseases that match it
        if len(symptoms) == 1:
            matched_diseases = []
            for disease, disease_symptoms in self.diseases.items():
                if symptoms[0] in disease_symptoms:
                    matched_diseases.append(disease)
            if matched_diseases:
                return f"Possible diseases based on symptom '{symptoms[0]}': {', '.join(matched_diseases)}"
            else:
                return f"No diseases found matching the symptom '{symptoms[0]}'"

        # Check symptoms against all diseases for multiple symptoms with a threshold of 2 matches
        for disease, disease_symptoms in self.diseases.items():
            matching_symptoms = [symptom for symptom in symptoms if symptom in disease_symptoms]
            # If 2 or more symptoms match, return the disease
            if len(matching_symptoms) >= 2:  # Adjusted threshold
                return f"Disease: {disease} with symptoms: {', '.join(matching_symptoms)}"

        return "Diagnosis unclear"
```

- In this demo, the system presents a list of diseases, each with its own non-overlapping symptom set ensuring accurate rule-based matching.

It accepts both Single and Multi-symptom inputs:

- Single symptoms return all possible related diseases
- Multi-symptom input applies a 3-symptom threshold to determine a clear diagnosis

- Once matched, the system outputs the Disease name and the Confirmed symptoms used in the diagnosis.

Code Demo Part 2:

```python
# Function to get symptoms from the user
def get_user_input(first_input=True):
    if first_input:
        print("\nWelcome to the Expert Diagnosis System!")
        print("\nPlease provide your symptoms so we can help diagnose your condition.")
        print("\nYou can either type a single symptom or a list of symptoms separated by commas (e.g., 'fever, body ache, fatigue').")
    else:
        print("\nPlease go ahead and enter your next set of symptoms below")

    symptoms_input = input("Enter your symptoms: ").lower().split(',')  # Get input and convert to lowercase
    symptoms_input = [symptom.strip() for symptom in symptoms_input]  # Clean up extra spaces
    return symptoms_input

# Main program execution
if __name__ == "__main__":
    expert_system = DiagnosisExpertSystem()

    first_input = True  # Flag to track if it's the first input (show welcome message)

    while True:
        # Get symptoms from user (show welcome message if first input, else skip it)
        symptoms_input = get_user_input(first_input)

        # Diagnose based on input
        diagnosis = expert_system.diagnose(symptoms_input)
        print(diagnosis)

        # Ask user if they want to continue
        print("\nWould you like to enter more symptoms? (yes/no)")
        continue_input = input().lower()

        if continue_input != 'yes':
            print("\nThank you for using the expert system!")
            break
        else:
            first_input = False  # After first input, don't show the welcome message again
```

- The expert system runs through an interactive console interface with clear, beginner-friendly instructions for both new and returning users.
- It supports comma-separated input, automatically cleans the data, and begins the diagnosis loop allowing users to perform multiple symptom checks in one session.
- The system shows a custom Welcome message on first launch and ends with a polite Thank-you message when the user exits. This keeps the experience simple, intuitive, and easy to repeat.

Code Demo Output:

```
Please provide your symptoms so we can help diagnose your condition.

You can either type a single symptom or a list of symptoms separated by commas (e.g., 'fever, body ache, fatigue').
Enter your symptoms: Chills
Possible diseases based on symptom 'chills': Flu

Would you like to enter more symptoms? (yes/no)
Yes

Please go ahead and enter your next set of symptoms below
Enter your symptoms: Swollen Lymph Nodes
Possible diseases based on symptom 'swollen lymph nodes': Strep Throat, Mononucleosis

Would you like to enter more symptoms? (yes/no)
Yes

Please go ahead and enter your next set of symptoms below
Enter your symptoms: Shortness of Breath
Possible diseases based on symptom 'shortness of breath': COVID-19, Pneumonia
```

- The system was tested using a variety of realistic input combinations to validate its accuracy.

Example Test Case:

- **Input**: ['fever', 'body ache', 'fatigue', 'chills']
- **Output**: Disease: Flu

Another Test Case:

- **Input**: ['dry cough', 'loss of taste or smell', 'shortness of breath']
- **Output**: Disease: COVID-19

```
Would you like to enter more symptoms? (yes/no)
No

Thank you for using the expert system!
PS C:\Users\User\Documents\Expert_System>
```

- The code is built with every test case scenario in mind ensuring fully thought out functionality.

FUTURE EXPANSION

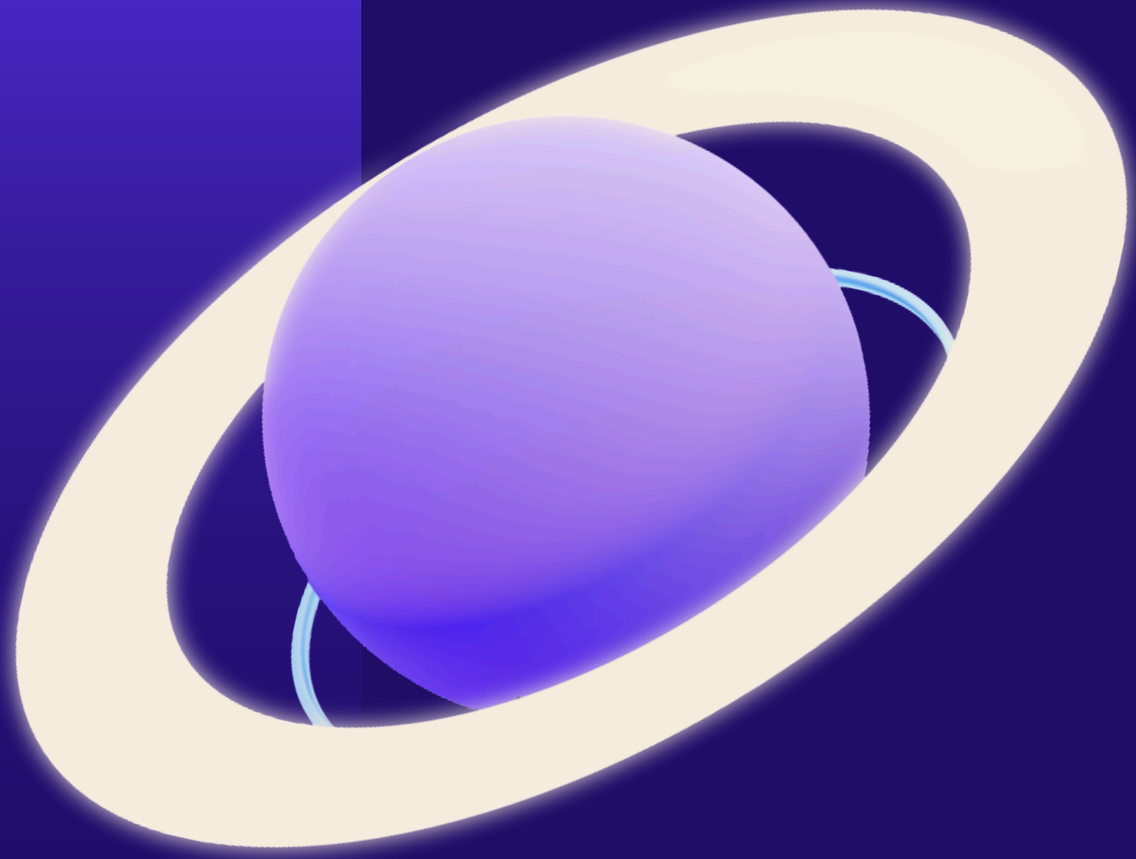
Add More Diseases & Severity Levels:
Expand the system’s library to cover a broader range of illnesses, along with symptom severity to improve diagnostic accuracy.

Develop Mobile & Web Applications:
Create user-friendly apps to make the tool accessible in real-time even without technical skills or desktop access.

Integrate Machine Learning:
Leverage ML to improve the system’s intelligence over time, especially for handling ambiguous or overlapping symptom profiles.

Clinical Validation & Real-World Deployment:
Collaborate with medical experts to clinically validate the system and deploy it for use in education, remote care, and rural health initiatives

THANKS FOR LISTENING



mnadeem2@student.gsu.edu

ekane7@student.gsu.edu