PROJECT REPORT

HEALTH AI

Intelligent Healthcare Assistant Using IBM Granite

YEAR	**: 2025 – 2026**
COLLEGE NAME	: **K.C.S KASI NADAR COLLEGE OF ARTS & SCIENCE**
CODE	:
DEPARTMENT	: **COMPUTER SCIENCE**
PROGRAM	: **B.SC**
SEMESTER	:**VI**

PROJECT SUBMITTED TO: UNIVERSITY OF MADRAS / NAAN MUDALVAN

COURSE NAME :

TEAM LEADER: **ANBARASAN.M**

MEMBERS:

1. BALAMURUGAN.S
2. DINESH KUMAR.S
3. LOKESH.B

GUIDED BY: MRS.R.PADMADEVI

Introduction

Healthcare is one of the most essential sectors for human well-being, but it faces multiple challenges across the globe. Rapid population growth, rising lifestyle-related diseases, and limited availability of trained medical professionals have created a situation where millions of people struggle to receive timely diagnosis and treatment. In rural and underdeveloped regions, this problem becomes even more critical due to lack of healthcare infrastructure, fewer doctors, and limited access to modern medical facilities. Even in urban areas, patients often face long waiting times, delayed consultations, and increasing healthcare costs. These challenges highlight the urgent need for innovative solutions that can make healthcare more accessible, affordable, and efficient.

To address these issues, the **Health AI Assistant** has been developed as an **AI-powered solution** using advanced technologies like **IBM Granite language models** and **Gradio user interfaces**. This system is designed to act as a smart healthcare companion that can support patients and doctors by providing quick insights, predictions, and treatment suggestions. Unlike traditional static medical systems, this assistant leverages **Natural Language Processing (NLP)** and **Machine Learning (ML)** to understand user queries, analyze symptoms, and generate meaningful medical insights. By providing instant feedback, it helps patients become more informed about their health conditions and guides them to seek timely professional care.

The key goal of the Health AI Assistant is not to replace doctors or medical professionals, but to act as a **supportive system**. For example, a patient experiencing fever, cough, and fatigue can enter their symptoms into the system, which will predict possible conditions such as flu, viral infection, or even more serious illnesses. At the same time, the system emphasizes that the results are **for informational purposes only** and advises users to consult healthcare professionals for accurate diagnosis and treatment. This responsible design ensures ethical usage and prevents misuse of AI-generated insights.

Another important aspect of the project is **personalized treatment planning**. By considering factors such as age, gender, and medical history, the system generates more tailored recommendations. This allows patients to receive relevant advice that aligns with their individual health conditions. Additionally, the assistant is capable of **simplifying complex medical reports**, making them more understandable for non-medical users. This feature is particularly useful for elderly patients or those unfamiliar with medical terminology.

The **Gradio interface** ensures that the system is easy to use and accessible for all users, regardless of their technical knowledge. With a simple dashboard, patients can interact with the assistant through text-based inputs and instantly receive structured, user-friendly outputs. Doctors can also use the tool to save time on preliminary checks, allowing them to focus more on critical cases. This feature is particularly useful for elderly patients or those unfamiliar with medical terminology. This responsible design ensures ethical usage and prevents misuse of AI-generated insights.

Features of Health AI

The **Health AI Assistant** is designed to make healthcare more accessible and intelligent through a set of practical features. These features aim to support both patients and doctors with timely insights, simple explanations, and easy interaction.

1. Disease Prediction and Anomaly Detection

The system analyzes symptoms entered by the user and provides a list of **possible conditions**. For example, fever and cough may indicate flu or viral infection. In addition, the **anomaly detection** feature identifies unusual or dangerous patterns, such as recurring chest pain, and highlights them as urgent cases. This helps in both **early awareness and preventive care**.

2. Personalized Treatment Planning

By considering details like **age, gender, and medical history**, the system suggests general treatment options and home remedies. A young user with mild fever may get advice to rest and stay hydrated, while an older patient with diabetes may be guided to consult a doctor quickly. This personalization makes the recommendations **more relevant and practical**.

3. Medical Report Summarization and Insights

Medical reports are often complex, filled with technical terms. The assistant converts them into **simpler summaries** so that patients can easily understand their health status. For instance, it may explain results as "blood sugar normal, cholesterol slightly high." This feature improves **clarity and patient awareness**.

4. User-Friendly Interactive Interface

Built with **Gradio**, the interface is clean and easy to use. Tabs for disease prediction and treatment planning ensure smooth navigation, while outputs are presented in a structured manner. Even users with little technical knowledge can interact with it comfortably, making the system **accessible to all**.

5. Anomaly Detection

The Health AI Assistant also includes **anomaly detection** to identify unusual or risky health patterns. By analyzing past records or repeated symptoms, the system can alert the patient if something is unusual or requires immediate medical attention. For example, if a patient frequently reports chest pain, the system can highlight it as a **potential emergency** and recommend immediate consultation. This proactive feature adds a layer of **preventive care** and ensures timely medical intervention.

Sample Code

```
# ----------------------------
# 1. Install Dependencies
# ----------------------------
!pip install -q transformers accelerate gradio torch plotly pandas

# ----------------------------
# 2. Import Libraries
# ----------------------------
import gradio as gr
import torch
import plotly.graph_objects as go
import plotly.express as px
import pandas as pd
import random
from datetime import datetime, timedelta
from transformers import AutoTokenizer, AutoModelForCausalLM

# ----------------------------
# 3. Load Model & Tokenizer
# ----------------------------
model_name = "ibm-granite/granite-3.2-2b-instruct"

tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForCausalLM.from_pretrained(
    model_name,
    torch_dtype=torch.float16 if torch.cuda.is_available() else torch.float32,
    device_map="auto" if torch.cuda.is_available() else None
)

if tokenizer.pad_token is None:
    tokenizer.pad_token = tokenizer.eos_token

# ----------------------------
# 4. Enhanced Response Generation
# ----------------------------
def generate_response(prompt, max_length=1024, temperature=0.7):
    inputs = tokenizer(
        prompt,
        return_tensors="pt",
        truncation=True,
        max_length=1024,
        padding=True
    )

    if torch.cuda.is_available():
        inputs = {k: v.to(model.device) for k, v in inputs.items()}
    with torch.no_grad():
        outputs = model.generate(
            **inputs,
            max_length=max_length,
            temperature=temperature,
            do_sample=True,
            pad_token_id=tokenizer.eos_token_id
        )

    response = tokenizer.decode(outputs[0], skip_special_tokens=True)
    response = response.replace(prompt, "").strip()
    return response

# ----------------------------
# 5. Symptom Checker with Risk Assessment
# ----------------------------
def symptom_checker(symptoms, severity, duration, additional_info):
    risk_level = "Low"
    if severity >= 7 or "severe" in symptoms.lower() or "emergency" in symptoms.lower():
        risk_level = "High"
    elif severity >= 5:
        risk_level = "Moderate"

    prompt = f"""
You are a medical AI assistant. Analyze the following symptoms and provide a comprehensive assessment.

Patient Information:
- Symptoms: {symptoms}
- Severity Level: {severity}/10
- Duration: {duration}
- Additional Information: {additional_info}

Please provide:
1. Most likely conditions (with probability estimates)
2. Recommended immediate actions
3. When to seek medical attention
4. Home care suggestions
5. Red flag symptoms to watch for

**MEDICAL DISCLAIMER: This is for informational purposes only. Always consult healthcare professionals.**

Assessment:
"""

    analysis = generate_response(prompt, max_length=1500)

    # Create risk visualization
    risk_colors = {"Low": "#22c55e", "Moderate": "#f59e0b", "High": "#ef4444"}
    risk_fig = go.Figure(go.Indicator(
        mode = "gauge+number",
        value = severity,
        domain = {'x': [0, 1], 'y': [0, 1]},
        title = {'text': f"Risk Level: {risk_level}"},
        gauge = {
            'axis': {'range': [None, 10]},
            'bar': {'color': risk_colors[risk_level]},
            'steps': [
                {'range': [0, 3], 'color': "#dcfce7"},
                {'range': [3, 6], 'color': "#fef3c7"},
                {'range': [6, 10], 'color': "#fee2e2"}],
            'threshold': {
                'line': {'color': "red", 'width': 4},
                'thickness': 0.75,
                'value': 8}}))

    risk_fig.update_layout(
        height=300,
        font={'color': "darkblue", 'family': "Arial"},
        paper_bgcolor='rgba(0,0,0,0)',
        plot_bgcolor='rgba(0,0,0,0)'
    )

    return analysis, risk_fig

# ---------------------------
# 6. Personalized Treatment Planner
# ---------------------------
def treatment_planner(condition, age, gender, weight, allergies, medications, lifestyle):
    prompt = f"""
Create a comprehensive, personalized treatment plan based on the patient profile below.

Patient Profile:
- Medical Condition: {condition}
- Age: {age} years
- Gender: {gender}
- Weight: {weight} kg
- Known Allergies: {allergies}
- Current Medications: {medications}
- Lifestyle Factors: {lifestyle}

Please provide:
1. Evidence-based treatment recommendations
2. Lifestyle modifications
3. Dietary suggestions
4. Exercise recommendations (if appropriate)
5. Monitoring schedule
6. Potential drug interactions to avoid
7. Follow-up timeline
```

```
**MEDICAL DISCLAIMER: This plan should be reviewed with a healthcare provider.**

Personalized Treatment Plan:
"""

    treatment_plan = generate_response(prompt, max_length=1800, temperature=0.6)

    # Create treatment timeline visualization
    timeline_data = {
        'Week': [1, 2, 3, 4, 8, 12, 24],
        'Action': [
            'Start treatment & lifestyle changes',
            'Monitor initial response',
            'Assess side effects',
            'First follow-up appointment',
            '2-month evaluation',
            '3-month comprehensive review',
            '6-month long-term assessment'
        ],
        'Priority': ['High', 'High', 'Medium', 'High', 'Medium', 'High', 'Medium']
    }

    df_timeline = pd.DataFrame(timeline_data)
    timeline_fig = px.scatter(df_timeline, x='Week', y='Action',
                              color='Priority', size=[3, 3, 2, 3, 2, 3, 2],
                              color_discrete_map={'High': '#ef4444', 'Medium': '#f59e0b'})
    timeline_fig.update_layout(
        title="Treatment Timeline",
        height=400,
        paper_bgcolor='rgba(0,0,0,0)',
        plot_bgcolor='rgba(0,0,0,0)'
    )

    return treatment_plan, timeline_fig

# ----------------------------
# 7. Health Metrics Tracker
# ----------------------------
def health_tracker(metric_type, current_value, target_value, time_period):
    # Simulate progress data
    dates = [datetime.now() - timedelta(days=x) for x in range(30, 0, -1)]

    # Generate realistic progress curve
    progress_values = []
    start_val = float(current_value)
    target_val = float(target_value)

    for i in range(30):
        # Add some realistic variation
        noise = random.uniform(-0.1, 0.1)
        progress = (i / 29) * 0.7  # 70% progress over 30 days
        value = start_val + (target_val - start_val) * progress + noise
        progress_values.append(value)

    df_progress = pd.DataFrame({
        'Date': dates,
        'Value': progress_values,
        'Target': [target_val] * 30
    })

    progress_fig = px.line(df_progress, x='Date', y=['Value', 'Target'],
                          title=f'{metric_type} Progress Tracking',
                          color_discrete_map={'Value': '#3b82f6', 'Target': '#ef4444'})
    progress_fig.update_layout(
        height=400,
        paper_bgcolor='rgba(0,0,0,0)',
        plot_bgcolor='rgba(0,0,0,0)'
    )

    # Calculate progress percentage
    progress_pct = ((progress_values[-1] - start_val) / (target_val - start_val)) * 100
    progress_pct = max(0, min(100, progress_pct))

    return progress_fig, f"Current Progress: {progress_pct:.1f}% towards target"

# ---------------------------
# 8. Medical Q&A Chatbot
# ---------------------------
def medical_chatbot(message, history):
    prompt = f"""
You are a knowledgeable medical AI assistant. Answer the user's health-related question with
accurate, helpful information while emphasizing the importance of professional medical
consultation.

User Question: {message}

Conversation History: {history[-3:] if history else "None"}

Please provide:
- Clear, evidence-based information
- Practical advice when appropriate
- Clear indication when professional consultation is needed
- Empathetic and supportive tone

**Always remind users this is for informational purposes only.**

Response:
"""
```

```
    response = generate_response(prompt, max_length=1000, temperature=0.6)
    history.append((message, response))
    return "", history

# ---------------------------
# 9. Emergency Symptom Checker
# ---------------------------
def emergency_checker(symptoms):
    emergency_keywords = [
        "chest pain", "difficulty breathing", "severe bleeding", "unconscious",
        "stroke", "heart attack", "severe headache", "high fever", "seizure",
        "severe abdominal pain", "difficulty swallowing", "severe allergic reaction"
    ]

    is_emergency = any(keyword in symptoms.lower() for keyword in emergency_keywords)

    if is_emergency:
        return """
🚨 EMERGENCY ALERT 🚨

Based on your symptoms, this may require immediate medical attention.

RECOMMENDED ACTIONS:
1. Call emergency services (911/108) immediately
2. Do not delay seeking medical care
3. If possible, have someone accompany you to the hospital
4. Bring a list of current medications

This is a medical emergency screening tool. When in doubt, seek immediate medical attention.
""", "🚨 EMERGENCY - Seek immediate medical care!"
    else:
        return """
Based on your symptoms, this does not appear to be an immediate emergency. However:

RECOMMENDED ACTIONS:
1. Monitor symptoms closely
2. Contact your healthcare provider if symptoms worsen
3. Seek medical attention if you develop new concerning symptoms
4. Consider scheduling a routine appointment for evaluation

Remember: This tool cannot replace professional medical judgment. Trust your instincts about your
health.
""", "ℹ Non-emergency - Monitor and consult healthcare provider"

# ---------------------------
# 10. Build Enhanced Gradio App
# ---------------------------
def create_custom_css():
    return """
    .gradio-container {
        background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
        font-family: 'Inter', sans-serif;
    }
    .gr-form {
        background: rgba(255, 255, 255, 0.95);
        backdrop-filter: blur(10px);
        border-radius: 15px;
        padding: 20px;
        box-shadow: 0 8px 32px rgba(0, 0, 0, 0.1);
        border: 1px solid rgba(255, 255, 255, 0.18);
    }
    .gr-button {
        background: linear-gradient(45deg, #667eea, #764ba2);
        border: none;
        border-radius: 25px;
        color: white;
        font-weight: 600;
        transition: all 0.3s ease;
        box-shadow: 0 4px 15px rgba(0, 0, 0, 0.2);
    }
    .gr-button:hover {
        transform: translateY(-2px);
        box-shadow: 0 6px 20px rgba(0, 0, 0, 0.3);
    }
    .medical-header {
        text-align: center;
        background: rgba(255, 255, 255, 0.1);
        backdrop-filter: blur(10px);
        border-radius: 15px;
        padding: 20px;
        margin-bottom: 20px;
        border: 1px solid rgba(255, 255, 255, 0.18);
    }
    """

with gr.Blocks(theme=gr.themes.Soft(), css=create_custom_css()) as app:
    gr.HTML("""
    <div class="medical-header">
        <h1 style="color: white; font-size: 3em; margin: 0; text-shadow: 2px 2px 4px rgba(0,0,0,0.3);">
            🏥 Advanced Medical AI Assistant
        </h1>
        <p style="color: rgba(255,255,255,0.9); font-size: 1.2em; margin: 10px 0 0 0;">
            Intelligent Healthcare Guidance & Analysis Platform
        </p>
        <div style="background: rgba(255, 87, 87, 0.2); border-radius: 10px; padding: 15px; margin-
top: 15px; border-left: 4px solid #ff5757;">
            <strong style="color: white;">⚠️ MEDICAL DISCLAIMER:</strong>
            <span style="color: rgba(255,255,255,0.9);">This AI assistant provides informational
guidance only. Always consult qualified healthcare professionals for diagnosis and treatment
decisions.</span>
```

```
      </div>
    </div>
    """)

    with gr.Tab("🔍 Smart Symptom Analysis", elem_classes="gr-form"):
      gr.Markdown("### Comprehensive Symptom Assessment with Risk Evaluation")

      with gr.Row():
        with gr.Column(scale=2):
          symptoms_input = gr.Textbox(
            label="📝 Describe Your Symptoms",
            placeholder="e.g., persistent headache with nausea, fatigue for 3 days...",
            lines=4
          )
          with gr.Row():
            severity_input = gr.Slider(
              label="🌡 Severity Level (1-10)",
              minimum=1, maximum=10, value=5,
              info="Rate your discomfort level"
            )
            duration_input = gr.Dropdown(
              choices=["Less than 1 day", "1-3 days", "4-7 days", "1-2 weeks", "More than 2
weeks"],
              label="⏰ Duration",
              value="1-3 days"
            )
          additional_input = gr.Textbox(
            label="➕ Additional Information",
            placeholder="Recent travel, medication changes, family history...",
            lines=2
          )
          analyze_btn = gr.Button("🔍 Analyze Symptoms", variant="primary", size="lg")

        with gr.Column(scale=1):
          risk_gauge = gr.Plot(label="📊 Risk Assessment")

      symptom_analysis = gr.Textbox(
        label="📋 Detailed Analysis & Recommendations",
        lines=15,
        show_copy_button=True
      )
      analyze_btn.click(
        symptom_checker,
        inputs=[symptoms_input, severity_input, duration_input, additional_input],
        outputs=[symptom_analysis, risk_gauge]
      )
    with gr.Tab("📋 Personalized Treatment Planner", elem_classes="gr-form"):
      gr.Markdown("### AI-Powered Treatment Planning with Timeline")
```

```
        with gr.Row():
            with gr.Column():
                condition_input = gr.Textbox(
                    label="🏥 Medical Condition",
                    placeholder="e.g., Type 2 Diabetes, Hypertension..."
                )
                with gr.Row():
                    age_input = gr.Number(label="🎂 Age", value=, minimum=1, maximum=120)
                    weight_input = gr.Number(label="⚖️ Weight (kg)", value=, minimum=30,
maximum=300)
                    gender_input = gr.Dropdown(
                        choices=["Male", "Female", "Other", "Prefer not to say"],
                        label="👤 Gender",
                        value=""
                    )
                allergies_input = gr.Textbox(
                    label="🚫 Known Allergies",
                    placeholder="e.g., Penicillin, Nuts, Shellfish..."
                )
                medications_input = gr.Textbox(
                    label="💊 Current Medications",
                    placeholder="List current medications and dosages..."
                )
                lifestyle_input = gr.Textbox(
                    label="🏃 Lifestyle Factors",
                    placeholder="Exercise habits, diet, smoking, alcohol use...",
                    lines=3
                )
                plan_btn = gr.Button("📝 Generate Treatment Plan", variant="primary", size="lg")

            with gr.Column():
                timeline_plot = gr.Plot(label="📅 Treatment Timeline")

        treatment_output = gr.Textbox(
            label="📋 Comprehensive Treatment Plan",
            lines=15,
            show_copy_button=True
        )
        plan_btn.click(
            treatment_planner,
            inputs=[condition_input, age_input, gender_input, weight_input, allergies_input,
medications_input, lifestyle_input],
            outputs=[treatment_output, timeline_plot]
        )
    with gr.Tab("📈 Health Progress Tracker", elem_classes="gr-form"):
        gr.Markdown("### Monitor Your Health Metrics & Progress")

        with gr.Row():
```

```
        metric_type = gr.Dropdown(
            choices=["Blood Pressure", "Blood Sugar", "Weight", "Cholesterol", "Heart Rate"],
            label="📊 Health Metric",
            value="Blood Pressure"
        )
        current_value = gr.Number(label="📍 Current Value", value=140)
        target_value = gr.Number(label="🎯 Target Value", value=120)
        time_period = gr.Dropdown(
            choices=["1 month", "3 months", "6 months", "1 year"],
            label="⏱️ Tracking Period",
            value="3 months"
        )

    track_btn = gr.Button("📈 Generate Progress Report", variant="primary")

    with gr.Row():
        progress_plot = gr.Plot(label="📊 Progress Visualization")
        progress_summary = gr.Textbox(label="📝 Progress Summary", lines=3)
    track_btn.click(
        health_tracker,
        inputs=[metric_type, current_value, target_value, time_period],
        outputs=[progress_plot, progress_summary]
    )
with gr.Tab("🚨 Emergency Checker", elem_classes="gr-form"):
    gr.Markdown("### Rapid Emergency Symptom Assessment")
    gr.Markdown("⚡ **Quick screening for symptoms requiring immediate medical attention**")

    emergency_symptoms = gr.Textbox(
        label="🆘 Emergency Symptoms",
        placeholder="Describe urgent symptoms you're experiencing...",
        lines=4
    )
    emergency_btn = gr.Button("🚨 Emergency Assessment", variant="stop", size="lg")

    with gr.Row():
        emergency_result = gr.Textbox(
            label="🚨 Emergency Assessment Result",
            lines=10
        )
        emergency_status = gr.Textbox(
            label="📋 Status",
            lines=1,
            max_lines=1
        )

    emergency_btn.click(
        emergency_checker,
        inputs=[emergency_symptoms],
            outputs=[emergency_result, emergency_status] )

    with gr.Tab("💬 Medical Q&A Chat", elem_classes="gr-form"):
        gr.Markdown("### Interactive Medical Assistant Chat")

        chatbot = gr.Chatbot(
            label="🤖 Medical AI Assistant",
            height=500,
            bubble_full_width=False
        )

        with gr.Row():
            msg_input = gr.Textbox(
                label="💭 Your Question",
                placeholder="Ask any health-related question...",
                scale=4)
            send_btn = gr.Button("📤 Send", variant="primary", scale=1)

        msg_input.submit(medical_chatbot, inputs=[msg_input, chatbot], outputs=[msg_input,
chatbot])
        send_btn.click(medical_chatbot, inputs=[msg_input, chatbot], outputs=[msg_input, chatbot])

    gr.HTML("""
    <div style="text-align: center; margin-top: 30px; padding: 20px; background:
rgba(255,255,255,0.1); border-radius: 15px;">
        <h3 style="color: white; margin: 0;">🔬 Powered by Advanced AI Technology</h3>
        <p style="color: rgba(255,255,255,0.8); margin: 10px 0 0 0;">
            Combining machine learning with medical knowledge for intelligent healthcare assistance
        </p>
    </div>
    """)

# ----------------------------
# 11. Launch Enhanced App
# ----------------------------
if __name__ == "__main__":
    app.launch(
        share=True,
        server_name="0.0.0.0",
        server_port=7860,
        show_error=True
    )
```

Project Structure

The **Health AI Assistant** is built with a simple yet effective architecture that brings together AI models, an interactive interface, and optional storage support. The project is divided into four main layers:

1. Frontend (Gradio) – User Dashboard

The **frontend** uses **Gradio** to provide a clean and interactive dashboard. Users can input symptoms or conditions and instantly receive predictions. Tabs for **Disease Prediction** and **Treatment Planning** ensure easy navigation. Since Gradio is browser-based, no complex setup is needed, making the system **accessible to both patients and doctors**. It also supports **real-time interaction**, which allows quick feedback and smoother user experience. The flexibility of Gradio also makes it easy to **extend the interface with new modules** in the future.

2. Backend (PyTorch + Hugging Face Transformers)

The **backend** handles AI processing tasks. Built with **PyTorch** and **Hugging Face Transformers**, it manages tokenization, model execution, and response generation. It acts as the **engine** of the project, ensuring smooth interaction between user inputs and AI outputs. The backend also adapts to hardware, using GPU when available for faster performance. This design ensures that even large-scale inputs are processed **efficiently and reliably**. It can also be scaled to **cloud-based environments** for larger deployments.

3. ML Model – IBM Granite LLM

At the core lies the **IBM Granite 3.2-2B Instruct model**, which performs **natural language processing (NLP)** tasks. It enables the system to understand patient queries, predict possible conditions, and suggest treatments. Unlike static chatbots, this model generates **context-aware responses**, making the assistant more intelligent and reliable. The model's adaptability also allows **fine-tuning for specialized healthcare use cases** in the future. Its flexibility makes it suitable for **multilingual healthcare systems** as well.

Overall, the project structure combines **frontend usability, backend processing, AI-powered predictions, and optional storage**. This modular design makes the system **scalable and adaptable**, allowing future enhancements such as wearable device integration, multilingual support, and cloud deployment. The clear separation of layers ensures easier maintenance, faster updates, and long-term **sustainability of the project**. It also enables smooth collaboration among developers working on different modules.

Project Execution

The execution of the **Health AI Assistant** demonstrates how the system functions in practice and showcases the value it provides to both patients and doctors. The application has been implemented with a clean and simple interface so that anyone, regardless of technical background, can interact with it. Execution mainly revolves around three key modules supported by the dashboard, which are explained below.

1. Disease Prediction Tab
The **Disease Prediction Tab** is designed as the first point of interaction for users. A text box allows patients to enter their symptoms in natural language, for example, "fever, headache, cough, and fatigue." Once submitted, the backend model powered by **IBM Granite LLM** processes the input and generates possible conditions. Typical outputs could include common illnesses such as influenza, seasonal viral fever, or even more serious infections like dengue or COVID-19.
The predictions are displayed in a clear, easy-to-read format so that users can quickly understand the results. To ensure safe usage, every output includes a disclaimer stating that the system is for **informational purposes only** and cannot replace professional diagnosis. This feature serves as an **early health awareness tool**, helping patients decide whether to manage symptoms at home or consult a doctor.

2. Treatment Plan Tab
The **Treatment Plan Tab** focuses on generating **personalized suggestions** based on patient details such as medical condition, age, gender, and health history. For example, if a user inputs "diabetes" as their condition along with an age of 50 years, the system may suggest a balanced diet, regular exercise, blood sugar monitoring, and consulting a healthcare provider for medication.
The recommendations cover **home remedies, general lifestyle changes, and supportive care tips**, which can be especially helpful in areas with limited access to doctors. However, similar to the prediction tab, this feature also emphasizes that users must seek professional medical guidance for any serious issues. By doing so, the system ensures both **usefulness and ethical responsibility**.

3. Dashboard and Report Summaries
The **Dashboard** integrates the different features of the Health AI system into a single platform. It provides access to both prediction and treatment tabs while also offering additional utilities such as **simplified medical report summaries** and medication reminders. Long and technical health records are converted into concise, patient-friendly statements like "blood sugar normal, cholesterol slightly high." This improves user understanding and reduces confusion.
Reminders help patients stay consistent with their treatment, particularly those managing chronic illnesses such as hypertension or asthma. By combining predictions, treatments, and reminders in one place, the dashboard enhances the overall **usability and effectiveness** of the application.

Additional Project Details

- **Project Demo Link:** [Add Google Drive/Video Link]
- **Project Source Code:** [Add GitHub/Repo Link]

In summary, the **execution of the Health AI project** proves its role as a supportive healthcare tool. By offering disease prediction, treatment planning, report summarization, and reminders through a simple interface, it provides a glimpse into how AI can make healthcare more accessible, affordable, and patient-friendly.

Conclusion

The **Health AI Assistant** project highlights the growing importance of artificial intelligence in addressing challenges faced by the healthcare industry. With rising populations, lifestyle diseases, and shortages of qualified medical professionals, traditional healthcare systems often struggle to provide timely and effective services. This project aims to bridge some of those gaps by offering an AI-driven platform that can assist patients with disease prediction, treatment suggestions, and report simplification.

One of the most impactful contributions of this project is its ability to provide **early awareness through disease prediction**. By allowing users to input symptoms in natural language, the system generates a list of possible medical conditions that may be associated with those symptoms. This feature is not intended to replace professional diagnosis, but it equips patients with an understanding of what their condition might be. In rural or underprivileged areas where access to doctors is limited, such functionality could mean the difference between delaying treatment and seeking timely care.

Another important achievement is the inclusion of **personalized treatment planning**. The system considers factors such as age, gender, and medical history before offering general suggestions. This ensures that advice is not generic but tailored to the individual. For instance, a young adult reporting mild fever might be advised to rest and hydrate, while an elderly patient with a chronic condition may receive recommendations to consult a doctor immediately. These personalized responses make the tool more practical and user-focused.

The feature of **medical report summarization** adds further value by making complex medical data easier to understand. Medical test reports are often filled with jargon and abbreviations that are confusing to the average patient. By simplifying these reports into clear, concise summaries, the Health AI Assistant enables patients to grasp their health status without relying entirely on medical professionals for interpretation. This not only increases patient awareness but also builds confidence in managing personal health.

Additional elements such as **reminders and anomaly detection** extend the usability of the system beyond predictions and treatments. Reminders help patients take medications on time, which is particularly useful for those with long-term conditions like hypertension or diabetes. Anomaly detection, on the other hand, ensures that unusual or potentially dangerous patterns in symptoms are flagged quickly, encouraging users to seek urgent medical attention. Together, these features contribute to preventive care and long-term health management.

From a technical perspective, the **architecture of the system**—which includes a Gradio-based frontend, PyTorch-powered backend, and IBM Granite model—ensures flexibility and scalability. The modular design allows for easier updates, integration of new features, and potential collaboration with healthcare institutions. Future expansions such as wearable device integration, multilingual support, and cloud-based data management could transform the project from a prototype into a comprehensive healthcare solution.

Overall, this project proves that AI can serve as a **supportive companion in healthcare**. While it is not a replacement for doctors, it acts as a tool to enhance decision-making, promote preventive healthcare, and empower patients with knowledge. It creates an environment where patients are more informed, proactive, and engaged in their health journeys.