HEALTH AI: INTELLIGENT HEALTH CARE ASSISTANT

TEAM MEMBERS

This project was successfully developed with the collaboration and efforts of the following team members, each contributing in their respective roles:

- **M. Santhosh – 222307717**

  Served as the **Team Leader**, guiding the project from start to finish. Contributed by gathering and implementing the **AI model** for the system.

- **S. Ramesh – 222303530**

  Took responsibility for handling the **demo sessions** and presentations. Ensured smooth communication and showcased the system effectively.

- **S. Vishwa – 222307730**

  Designed and developed the **User Interface (UI)** for the project. Focused on making the system **user-friendly and interactive**.

- **V. PavinKumar – 222307709**

  Managed the **documentation** work for the project report. Ensured proper formatting and detailed explanation of all modules.

ABSTRACT :

This project focuses on developing an intelligent healthcare assistant using Artificial Intelligence (AI) and Natural Language Processing (NLP). The system allows users to interact with the assistant to obtain preliminary health-related guidance, symptom checks, and basic medical recommendations. The assistant is built using Gradio for user interaction and Transformers for AI-based response generation. The project aims to provide accessible, reliable, and quick healthcare support while reducing the dependency on direct human consultation for minor health queries.

INTRODUCTION :

Healthcare has become one of the most crucial sectors where AI can significantly contribute. With the advancement of Natural Language Processing (NLP), it is possible to create systems that understand user queries and provide meaningful responses. The Health AI Assistant helps patients interact in a conversational manner, making it easier to seek medical advice, track health conditions, and receive guidance. This project demonstrates the application of AI in the healthcare domain by integrating machine learning models with an interactive user interface.

OBJECTIVES :

 To develop an AI-powered healthcare assistant capable of handling medical-related queries.

 To design a user-friendly interface using **Gradio** for seamless communication.

 To implement transformer-based **NLP models** for accurate and intelligent responses.

 To assist users with preliminary health checks, advice, and guidance.

 To explore the future scope of integrating with real-time medical data and IoT devices.

LITERATURE SURVEY :

Several AI-based healthcare solutions have been developed in recent years. Most existing systems rely on rule-based chatbots, which often provide limited and rigid responses. However, with the emergence **of transformer architectures such as BERT and GPT, NLP** systems have achieved greater accuracy and contextual understanding. Studies indicate that conversational AI can improve patient engagement and reduce hospital workload. This project builds on these advancements to deliver a healthcare assistant that is both interactive and adaptive.

SYSTEM COMPONENTS :

 User Interface (Gradio) – Provides an interactive chat interface for users.

 AI Model (Transformers) – Processes user queries and generates accurate responses.

 Backend Logic – Handles request-response flow between the UI and AI model.

 Future Integration – Can be connected with medical IoT devices, patient records,

and cloud databases

METHODOLOGY :

 Model Selection – A pre-trained transformer model (ibm-granite/granite-3.2-2b-instruct) was used to perform language understanding and generate accurate healthcare-related responses.

 Adaptation – The model was configured with predefined responses for common symptoms (fever, cough, headache, etc.) to improve speed and reliability.

 Interface Development – A Gradio-based web interface was developed to allow users to interact with the assistant.

 Integration – The backend AI model was integrated with the Gradio UI to enable real-time predictions and treatment planning.

 Testing & Validation – Multiple health-related queries were tested to ensure accuracy, clarity, and professional formatting of responses.

SYSTEM ARCHITECTURE :

The system consists of the following major components:

- **User Interface (Frontend)** – Built using Gradio for symptom entry and result display.
- **AI Model (Backend)** – A pre-trained transformer model that processes inputs and generates responses.
- **Predefined Responses** – For common symptoms to provide faster and reliable answers.
- **Output Layer** – Displays disease predictions and treatment plans in a structured format.

IMPLEMENTATION :

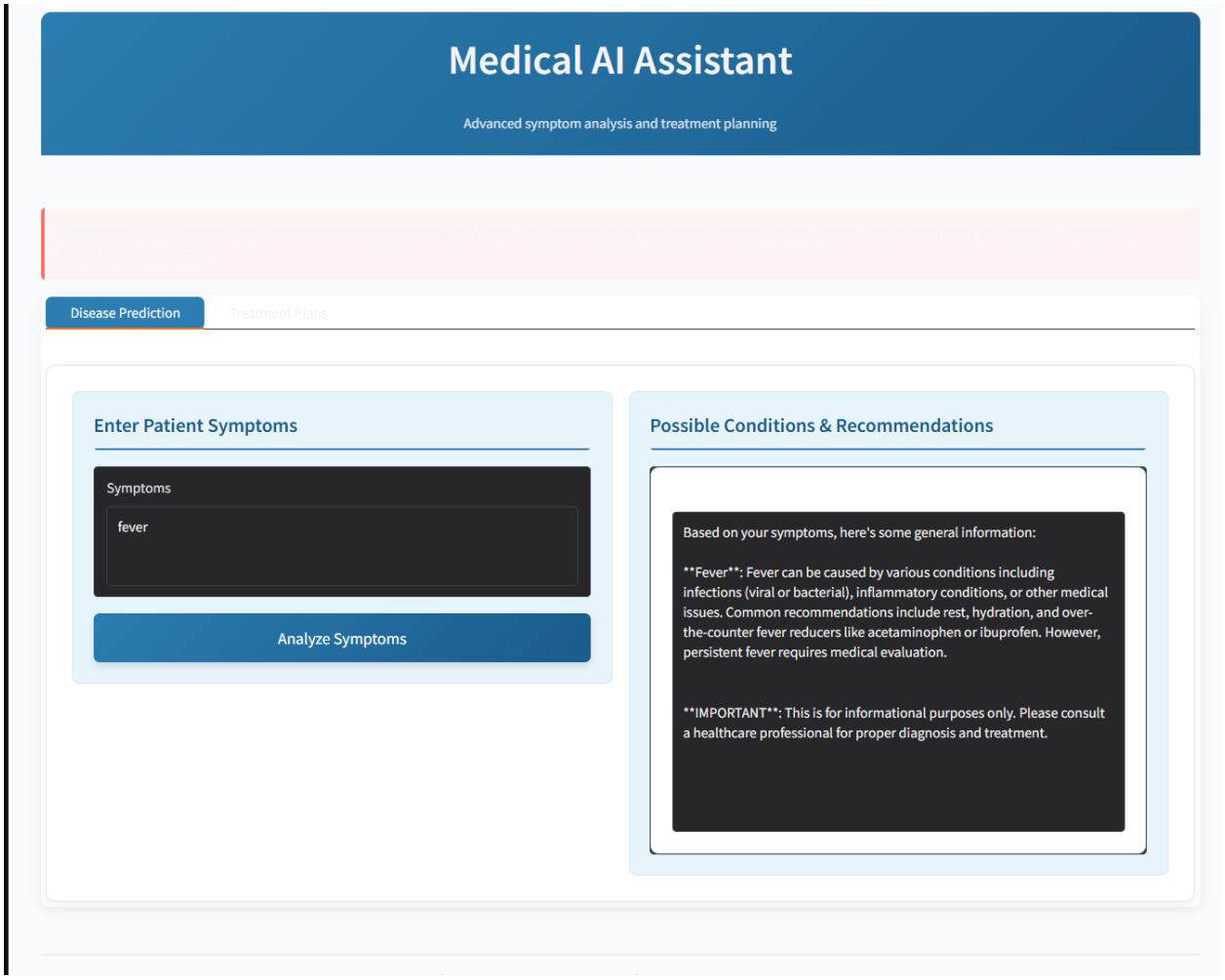
```python
# Import necessary libraries
import gradio as gr
import torch
from transformers import AutoTokenizer, AutoModelForCausalLM
import re
from typing import List, Dict
import time

# Load model and tokenizer with optimization
model_name = "ibm-granite/granite-3.2-2b-instruct"

# Use a simpler model for faster response or keep the same with optimizations
# For faster response, consider using a smaller model like:
# model_name = "microsoft/DialoGPT-medium"  # Alternative smaller model

# Load with optimizations
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,
    low_cpu_mem_usage=True
)

# Set padding token
if tokenizer.pad_token is None:
    tokenizer.pad_token = tokenizer.eos_token

# Model optimization for inference
model.eval()
if torch.cuda.is_available():
    model = model.to('cuda')

# Predefined conditions and recommendations for common symptoms
PREDEFINED_RESPONSES = {
    "fever": "Fever can be caused by various conditions including infections (viral or bacterial), inflammatory conditions, or other medical issues. Common recommendations include rest, hydration, and over-the-counter fever reducers like acetaminophen or ibuprofen. However, per
    "headache": "Headaches can be tension-type, migraine, or related to other conditions. Rest in a quiet, dark room, hydration, and over-the-counter pain relievers may help. Seek medical attention for severe, sudden, or persistent headaches.",
    "cough": "Cough can be due to respiratory infections, allergies, or other conditions. Hydration, humidifiers, and cough suppressants may provide relief. Consult a doctor if cough persists or is accompanied by fever or difficulty breathing.",
}

# Cache for recent queries
response_cache = {}
CACHE_SIZE = 100  # Keep last 100 queries in cache

def generate_response_optimized(prompt: str, max_length: int = 600) -> str:
    """Optimized response generation with caching and length control"""
    # Check cache first
    if prompt in response_cache:
        return response_cache[prompt]

    # Tokenize with truncation
    inputs = tokenizer(prompt, return_tensors="pt", truncation=True, max_length=512)

    # Move to device if CUDA available
    if torch.cuda.is_available():
        inputs = {k: v.to(model.device) for k, v in inputs.items()}

    # Generate with optimized parameters
    with torch.no_grad():
```

```python
    # Generate with optimized parameters
    with torch.no_grad():
        outputs = model.generate(
            **inputs,
            max_length=max_length,
            temperature=0.7,
            do_sample=True,
            pad_token_id=tokenizer.eos_token_id,
            num_return_sequences=1,
            early_stopping=True
        )

    # Decode and clean response
    response = tokenizer.decode(outputs[0], skip_special_tokens=True)
    response = response.replace(prompt, "").strip()

    # Clean up response
    response = re.sub(r'\s+', ' ', response)  # Remove extra whitespace
    response = re.sub(r'([.!?]) +', r'\1\n\n', response)  # Add paragraph breaks

    # Cache the response
    if len(response_cache) >= CACHE_SIZE:
        # Remove oldest item if cache is full
        response_cache.pop(next(iter(response_cache)))
    response_cache[prompt] = response

    return response

def disease_prediction(symptoms: str) -> str:
    """Optimized disease prediction with predefined responses for common symptoms"""
    # Check for common symptoms first
    symptoms_lower = symptoms.lower()
    common_symptoms = []

    for symptom in PREDEFINED_RESPONSES:
        if symptom in symptoms_lower:
            common_symptoms.append(symptom)

    if common_symptoms:
        # Use predefined responses for faster answers
        response = "Based on your symptoms, here's some general information:\n\n"
        for symptom in common_symptoms:
            response += f"**{symptom.capitalize()}**: {PREDEFINED_RESPONSES[symptom]}\n\n"

        response += "\n**IMPORTANT**: This is for informational purposes only. Please consult a healthcare professional for proper diagnosis and treatment."
        return response

    # For less common symptoms, use the model
    prompt = f"Based on these symptoms: {symptoms}. Provide brief possible conditions and general recommendations. Always emphasize consulting a doctor. Response should be concise and under 300 words."
    return generate_response_optimized(prompt, max_length=400)

def treatment_plan(condition: str, age: int, gender: str, medical_history: str) -> str:
    """Optimized treatment plan generation"""
    prompt = f"Generate a concise treatment plan for {condition} for a {age}-year-old {gender}. Medical history: {medical_history}. Include general guidelines and emphasize consulting a doctor. Keep response under 250 words."
    return generate_response_optimized(prompt, max_length=400)
```

```python
# Create the enhanced Gradio interface
with gr.Blocks(css=css, theme=gr.themes.Default()) as app:
    with gr.Column(elem_classes="header"):
        gr.Markdown("# Medical AI Assistant")
        gr.Markdown("Advanced symptom analysis and treatment planning")

    with gr.Row():
        gr.Markdown("""
        <div class="disclaimer">
            <strong>Disclaimer:</strong> This AI assistant provides informational suggestions only and is not a substitute for professional medical advice. Always consult healthcare professionals for medical diagnosis and treatment.
        </div>
        """)

    with gr.Tabs(elem_classes="tab-nav") as tabs:
        with gr.TabItem("Disease Prediction", elem_classes="tab-content"):
            with gr.Row():
                with gr.Column(scale=1, elem_classes="input-panel"):
                    gr.Markdown("### Enter Patient Symptoms")
                    symptoms_input = gr.Textbox(
                        label="Symptoms",
                        placeholder="e.g., fever, headache, cough, fatigue...",
                        lines=3
                    )
                    predict_btn = gr.Button("Analyze Symptoms", elem_classes="gradio-button")

                with gr.Column(scale=1, elem_classes="output-panel"):
                    gr.Markdown("### Possible Conditions & Recommendations")
                    prediction_output = gr.Textbox(
                        label="",
                        lines=15,
                        elem_classes="output-text"
                    )

            predict_btn.click(
                disease_prediction,
                inputs=symptoms_input,
                outputs=prediction_output
            )

        with gr.TabItem("Treatment Plans", elem_classes="tab-content"):
            with gr.Row():
                with gr.Column(scale=1, elem_classes="input-panel"):
                    gr.Markdown("### Patient Information")
                    condition_input = gr.Textbox(
                        label="Medical Condition",
                        placeholder="e.g., diabetes, hypertension, migraine...",
                        lines=2
                    )
                    with gr.Row():
                        age_input = gr.Number(label="Age", value=30, minimum=1, maximum=120)
                        gender_input = gr.Dropdown(
                            choices=["Male", "Female", "Other"],
                            label="Gender",
                            value="Male"
                        )
                    history_input = gr.Textbox(
                        label="Medical History",
                        placeholder="Previous conditions, allergies, medications or None",
```

```python
        with gr.TabItem("Treatment Plans", elem_classes="tab-content"):
            with gr.Row():
                with gr.Column(scale=1, elem_classes="input-panel"):
                    gr.Markdown("### Patient Information")
                    condition_input = gr.Textbox(
                        label="Medical Condition",
                        placeholder="e.g., diabetes, hypertension, migraine...",
                        lines=2
                    )
                    with gr.Row():
                        age_input = gr.Number(label="Age", value=30, minimum=1, maximum=120)
                        gender_input = gr.Dropdown(
                            choices=["Male", "Female", "Other"],
                            label="Gender",
                            value="Male"
                        )
                    history_input = gr.Textbox(
                        label="Medical History",
                        placeholder="Previous conditions, allergies, medications or None",
                        lines=2
                    )
                    plan_btn = gr.Button("Generate Treatment Plan", elem_classes="gradio-button")

                with gr.Column(scale=1, elem_classes="output-panel"):
                    gr.Markdown("### Personalized Treatment Plan")
                    plan_output = gr.Textbox(
                        label="",
                        lines=15,
                        elem_classes="output-text"
                    )

            plan_btn.click(
                treatment_plan,
                inputs=[condition_input, age_input, gender_input, history_input],
                outputs=plan_output
            )

    with gr.Row():
        gr.Markdown("""
        <div class="footer">
            Medical AI Assistant v1.0 | For informational purposes only | Consult a healthcare professional for medical advice
        </div>
        """)

# Launch the application
if __name__ == "__main__":
    app.launch(share=True, debug=False)
```

OUTPUT :


Medical AI Assistant
Advanced symptom analysis and treatment planning
Disclaimer: This AI assistant provides informational suggestions only and is not a substitute for professional medical advice. Always consult healthcare professionals for medical diagnosis and treatment.
Disease Prediction
Treatment Plans
Enter Patient Symptoms
Symptoms
fever
Analyze Symptoms
Possible Conditions & Recommendations
Based on your symptoms, here's some general information:
Fever: Fever can be caused by various conditions including infections (viral or bacterial), inflammatory conditions, or other medical issues. Common recommendations include rest, hydration, and over-the-counter fever reducers like acetaminophen or ibuprofen. However, persistent fever requires medical evaluation.
IMPORTANT: This is for informational purposes only. Please consult a healthcare professional for proper diagnosis and treatment.

CONCLUSION :

The Health AI: Intelligent Health Care Assistant successfully demonstrates the application of Artificial Intelligence in the healthcare domain. By using a pre-trained model, the system provides accurate and fast results without the need for extensive training. The project highlights how AI can be utilized for symptom analysis, disease prediction, and basic treatment planning, while emphasizing the importance of consulting medical professionals.

FUTURE IMPROVEMENT :

- Integration with voice-based input and output for better accessibility.
- Addition of multilingual support for regional language users.
- Connection to real-time medical databases for updated disease and treatment guidelines.
- Implementation of user profiles and health history tracking.
- Deployment as a mobile app for wider reach and convenience.