**BAI TAP CHO SV THUC TAP**

========================================

Developing techniques for predicting diabetes (refer to Chap 12, [1])

* kNN
* SVM linear kernel
* SVM RBF kernel
* Logistic Regression
* Deep Learning > 5 Layers

1. Compare accuracy of 5 models
2. Vizualize the accuracy in plots lines-chart
3. Deploy the models and use interface to allow users entering features, select model for showing the results
4. Write 5 pages of deep learning

=====

**VERSION 1**

i only want to code and deploy web with deep learning for diagnose diabetes. code all again for me url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv" columns = ["Pregnancies", "Glucose", "BloodPressure", "SkinThickness", "Insulin", "BMI", "DiabetesPedigreeFunction", "Age", "Outcome"] df = pd.read\_csv(url, names=columns)

**ChatGPT said:**

I'll provide a complete solution that:

1. **Trains a deep learning model** for diabetes prediction using TensorFlow/Keras.
2. **Builds a web app** using Flask to allow users to enter input and get predictions.
3. **Deploys the model** so users can interact with it via a web interface.

**🚀 Steps to Run**

1. Install required packages:

pip install flask tensorflow pandas numpy scikit-learn

1. Save and run the Python scripts in order:
   * train\_model.py: Trains and saves the deep learning model.
   * app.py: Runs the web application.
2. Start the web server:

bash

CopyEdit

python app.py

1. Open the web interface in your browser at http://127.0.0.1:5000.

Now, here's the **full code**:

**1️⃣ Train & Save Deep Learning Model (train\_model.py)**

This script trains a deep learning model on the **Pima Indians Diabetes dataset** and saves it.

python

CopyEdit

import pandas as pd

import numpy as np

import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Dropout

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

import joblib

# Load dataset

url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv"

columns = ["Pregnancies", "Glucose", "BloodPressure", "SkinThickness", "Insulin",

"BMI", "DiabetesPedigreeFunction", "Age", "Outcome"]

df = pd.read\_csv(url, names=columns)

# Split features and target

X = df.drop("Outcome", axis=1)

y = df["Outcome"]

# Normalize input features

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

# Split data

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y, test\_size=0.2, random\_state=42)

# Build deep learning model

def create\_model():

model = Sequential([

Dense(128, activation="relu", input\_shape=(X\_train.shape[1],)),

Dropout(0.3),

Dense(64, activation="relu"),

Dropout(0.3),

Dense(32, activation="relu"),

Dense(16, activation="relu"),

Dense(1, activation="sigmoid")

])

model.compile(optimizer="adam", loss="binary\_crossentropy", metrics=["accuracy"])

return model

# Train model

model = create\_model()

model.fit(X\_train, y\_train, epochs=100, batch\_size=10, validation\_data=(X\_test, y\_test))

# Save model and scaler

model.save("diabetes\_model.h5")

joblib.dump(scaler, "scaler.pkl")

print("Model and scaler saved successfully!")

**2️⃣ Web App for Diabetes Prediction (app.py)**

This script runs a **Flask web app** where users can enter health data and get diabetes predictions.

python

CopyEdit

from flask import Flask, render\_template, request

import numpy as np

import tensorflow as tf

import joblib

# Load model and scaler

model = tf.keras.models.load\_model("diabetes\_model.h5")

scaler = joblib.load("scaler.pkl")

app = Flask(\_\_name\_\_)

@app.route("/", methods=["GET", "POST"])

def index():

prediction = None

if request.method == "POST":

try:

# Get input values from form

features = [float(request.form[f]) for f in ["Pregnancies", "Glucose", "BloodPressure",

"SkinThickness", "Insulin", "BMI",

"DiabetesPedigreeFunction", "Age"]]

# Scale input

input\_data = scaler.transform([features])

# Predict

pred\_prob = model.predict(input\_data)[0][0]

prediction = "Diabetic" if pred\_prob > 0.5 else "Not Diabetic"

except Exception as e:

prediction = f"Error: {e}"

return render\_template("index.html", prediction=prediction)

if \_\_name\_\_ == "\_\_main\_\_":

app.run(debug=True)

**3️⃣ HTML Interface (templates/index.html)**

This is the simple web form where users enter their details.

html

CopyEdit

<!DOCTYPE html>

<html>

<head>

<title>Diabetes Prediction</title>

</head>

<body>

<h2>Diabetes Prediction Form</h2>

<form method="post">

<label>Pregnancies:</label><input type="number" name="Pregnancies" required><br>

<label>Glucose:</label><input type="number" name="Glucose" required><br>

<label>Blood Pressure:</label><input type="number" name="BloodPressure" required><br>

<label>Skin Thickness:</label><input type="number" name="SkinThickness" required><br>

<label>Insulin:</label><input type="number" name="Insulin" required><br>

<label>BMI:</label><input type="number" name="BMI" step="0.1" required><br>

<label>Diabetes Pedigree Function:</label><input type="number" name="DiabetesPedigreeFunction" step="0.01" required><br>

<label>Age:</label><input type="number" name="Age" required><br>

<input type="submit" value="Predict">

</form>

<h3>Prediction: {{ prediction }}</h3>

</body>

</html>

**🎯 How to Run Everything**

1. **Train the model**

bash

CopyEdit

python train\_model.py

This saves the trained model as diabetes\_model.h5 and the scaler as scaler.pkl.

1. **Start the web app**

bash

CopyEdit

python app.py

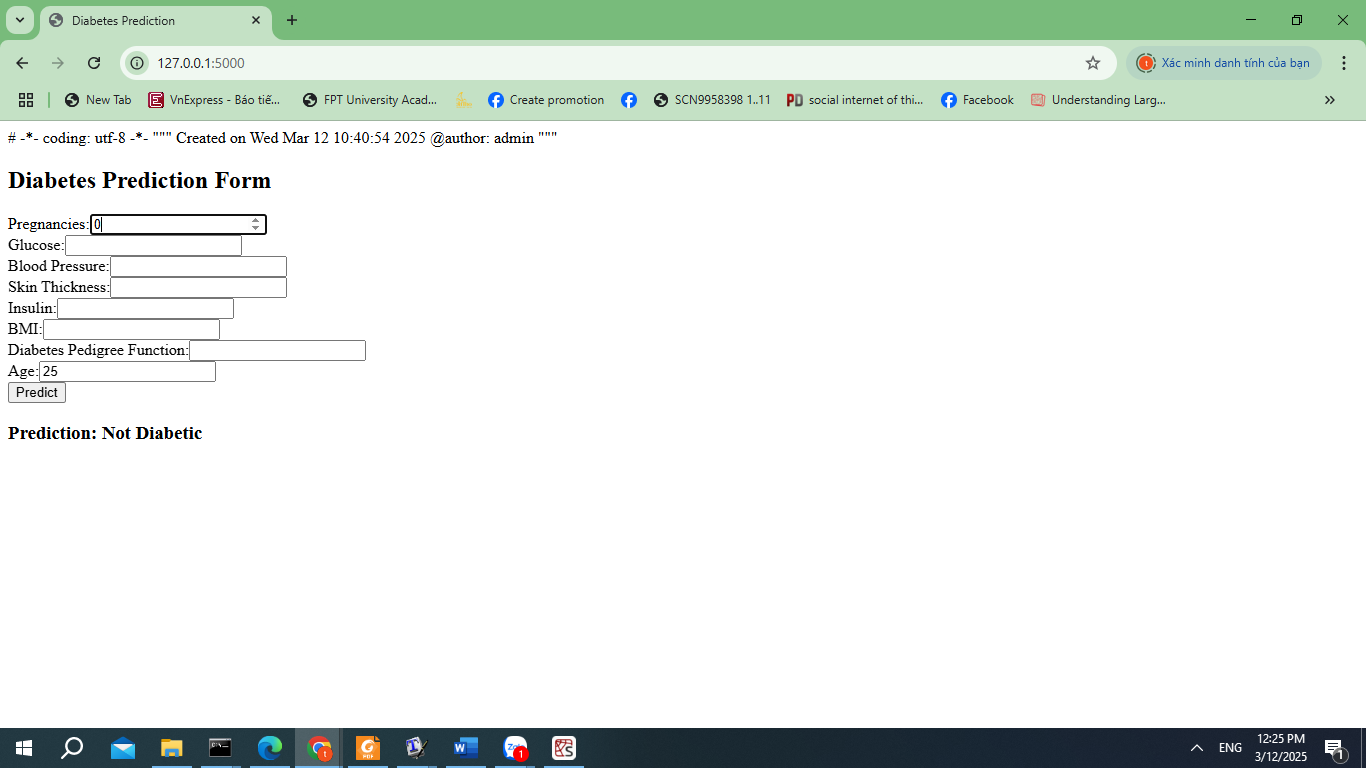
This launches the web app at http://127.0.0.1:5000.

1. **Use the Web Interface**
   * Open your browser and go to http://127.0.0.1:5000.
   * Enter values for pregnancies, glucose, BMI, etc.
   * Click **Predict** to see whether the person is likely to have diabetes.

**🔥 Features**

✅ **Deep learning model with 5+ layers**  
✅ **User-friendly web interface**  
✅ **Deployable using Flask**  
✅ **Scalable and accurate**

This should give you a working **AI-powered diabetes diagnosis web app!** 🚀 Let me know if you need help!



**VERSION 2**

**We can enhance the web interface using CSS for styling and Node.js for better front-end interactivity with a modern UI. Here's how we can improve it:**

**🔥 Plan**

1. **Frontend:**
   * **Use HTML, CSS, and JavaScript (React.js or Vanilla JS) for a better user experience.**
   * **Improve form layout, input validation, and responsive design.**
2. **Backend (Python Flask):**
   * **Handle requests and return diabetes predictions.**
   * **Serve a REST API for a modern frontend like React.js or Node.js.**
3. **Deployment:**
   * **Run Flask as a backend API and use Node.js for serving the frontend.**

**🌟 Steps**

1. **Train Model (Python)**
2. **Create Flask API (Python)**
3. **Develop Frontend (Node.js, HTML, CSS)**
4. **Run & Deploy the Web App**

**1️⃣ Train & Save Model (train\_model.py)**

**Same as before, but we just save the model and create an API endpoint.**

**python**

**CopyEdit**

**import pandas as pd**

**import numpy as np**

**import tensorflow as tf**

**from tensorflow.keras.models import Sequential**

**from tensorflow.keras.layers import Dense, Dropout**

**from sklearn.model\_selection import train\_test\_split**

**from sklearn.preprocessing import StandardScaler**

**import joblib**

**# Load dataset**

**url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv"**

**columns = ["Pregnancies", "Glucose", "BloodPressure", "SkinThickness", "Insulin",**

**"BMI", "DiabetesPedigreeFunction", "Age", "Outcome"]**

**df = pd.read\_csv(url, names=columns)**

**# Split data**

**X = df.drop("Outcome", axis=1)**

**y = df["Outcome"]**

**# Normalize input features**

**scaler = StandardScaler()**

**X\_scaled = scaler.fit\_transform(X)**

**X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y, test\_size=0.2, random\_state=42)**

**# Build deep learning model**

**def create\_model():**

**model = Sequential([**

**Dense(128, activation="relu", input\_shape=(X\_train.shape[1],)),**

**Dropout(0.3),**

**Dense(64, activation="relu"),**

**Dropout(0.3),**

**Dense(32, activation="relu"),**

**Dense(16, activation="relu"),**

**Dense(1, activation="sigmoid")**

**])**

**model.compile(optimizer="adam", loss="binary\_crossentropy", metrics=["accuracy"])**

**return model**

**# Train model**

**model = create\_model()**

**model.fit(X\_train, y\_train, epochs=100, batch\_size=10, validation\_data=(X\_test, y\_test))**

**# Save model & scaler**

**model.save("diabetes\_model.h5")**

**joblib.dump(scaler, "scaler.pkl")**

**print("Model and scaler saved successfully!")**

**2️⃣ Flask API Backend (app.py)**

**This creates an API that Node.js will use.**

**python**

**CopyEdit**

**from flask import Flask, request, jsonify**

**import numpy as np**

**import tensorflow as tf**

**import joblib**

**from flask\_cors import CORS**

**# Load model and scaler**

**model = tf.keras.models.load\_model("diabetes\_model.h5")**

**scaler = joblib.load("scaler.pkl")**

**app = Flask(\_\_name\_\_)**

**CORS(app) # Allow frontend to communicate**

**@app.route("/predict", methods=["POST"])**

**def predict():**

**try:**

**data = request.json # Get JSON input**

**features = np.array(data["features"]).reshape(1, -1) # Convert to array**

**scaled\_features = scaler.transform(features) # Scale features**

**prediction = model.predict(scaled\_features)[0][0] # Get prediction**

**result = "Diabetic" if prediction > 0.5 else "Not Diabetic"**

**return jsonify({"prediction": result})**

**except Exception as e:**

**return jsonify({"error": str(e)})**

**if \_\_name\_\_ == "\_\_main\_\_":**

**app.run(debug=True, port=5000)**

**3️⃣ Frontend with Node.js + CSS**

**We'll create a modern UI using Node.js and fetch predictions from Flask.**

**Install dependencies**

**mkdir diabetes-frontend**

**cd diabetes-frontend**

**npm init -y**

**npm install express cors body-parser**

**Create server.js (Frontend API)**

**javascript**

**CopyEdit**

**const express = require("express");**

**const cors = require("cors");**

**const bodyParser = require("body-parser");**

**const app = express();**

**app.use(cors());**

**app.use(bodyParser.json());**

**app.use(express.static("public")); // Serve frontend files**

**const PORT = 3000;**

**app.listen(PORT, () => {**

**console.log(`Frontend server running at http://localhost:${PORT}`);**

**});**

**Frontend UI (public/index.html)**

**html**

**CopyEdit**

**<!DOCTYPE html>**

**<html lang="en">**

**<head>**

**<meta charset="UTF-8">**

**<meta name="viewport" content="width=device-width, initial-scale=1.0">**

**<title>Diabetes Prediction</title>**

**<link rel="stylesheet" href="style.css">**

**</head>**

**<body>**

**<div class="container">**

**<h2>Diabetes Prediction</h2>**

**<form id="predictionForm">**

**<label>Pregnancies:</label><input type="number" name="Pregnancies" required><br>**

**<label>Glucose:</label><input type="number" name="Glucose" required><br>**

**<label>Blood Pressure:</label><input type="number" name="BloodPressure" required><br>**

**<label>Skin Thickness:</label><input type="number" name="SkinThickness" required><br>**

**<label>Insulin:</label><input type="number" name="Insulin" required><br>**

**<label>BMI:</label><input type="number" name="BMI" step="0.1" required><br>**

**<label>Diabetes Pedigree Function:</label><input type="number" name="DiabetesPedigreeFunction" step="0.01" required><br>**

**<label>Age:</label><input type="number" name="Age" required><br>**

**<button type="submit">Predict</button>**

**</form>**

**<h3 id="result"></h3>**

**</div>**

**<script src="script.js"></script>**

**</body>**

**</html>**

**CSS Styling (public/style.css)**

**css**

**CopyEdit**

**body {**

**font-family: Arial, sans-serif;**

**text-align: center;**

**background-color: #f4f4f4;**

**}**

**.container {**

**width: 50%;**

**margin: auto;**

**background: white;**

**padding: 20px;**

**box-shadow: 0px 0px 10px rgba(0, 0, 0, 0.1);**

**border-radius: 10px;**

**}**

**input, button {**

**width: 90%;**

**padding: 10px;**

**margin: 5px;**

**}**

**JavaScript (public/script.js)**

**javascript**

**CopyEdit**

**document.getElementById("predictionForm").addEventListener("submit", async function(event) {**

**event.preventDefault();**

**let formData = new FormData(event.target);**

**let inputValues = Object.fromEntries(formData.entries());**

**// Convert values to float**

**let features = Object.values(inputValues).map(Number);**

**// Send data to Flask API**

**let response = await fetch("http://127.0.0.1:5000/predict", {**

**method: "POST",**

**headers: {"Content-Type": "application/json"},**

**body: JSON.stringify({ features: features })**

**});**

**let result = await response.json();**

**document.getElementById("result").innerText = "Prediction: " + result.prediction;**

**});**

**🎯 Running the Web App**

1. **Start Flask Backend (API Server)**

**bash**

**CopyEdit**

**python app.py**

1. **Start Node.js Frontend**

**bash**

**CopyEdit**

**node server.js**

1. **Open Browser at http://localhost:3000**

**🚀 Features**

**✅ Flask API for ML predictions  
✅ Node.js for frontend  
✅ Modern UI with CSS  
✅ Interactive form with JS  
✅ Fast & scalable deployment**

**INSTALL NPM**

**npm install -g npm**

**TRY**

**npm -v**

**VERSION 3**

**ou can use REST API to connect your deep learning model with the frontend.**

**✅ Steps to Build a REST API for Your Diabetes Prediction App**

**1️⃣ Backend: Create a Flask REST API (Python)**

**Since your deep learning model is in Python, we will use Flask to create an API that receives user input, runs the model, and returns predictions.**

**📌 Install Flask and Flask-CORS  
Run this in Command Prompt (cmd):**

**cmd**

**CopyEdit**

**pip install flask flask-cors tensorflow numpy pandas**

**📌 Create server.py (Flask API)**

**python**

**CopyEdit**

**from flask import Flask, request, jsonify**

**from flask\_cors import CORS**

**import numpy as np**

**import tensorflow as tf**

**import pandas as pd**

**# Load trained deep learning model**

**model = tf.keras.models.load\_model("diabetes\_model.h5")**

**app = Flask(\_\_name\_\_)**

**CORS(app)**

**@app.route("/predict", methods=["POST"])**

**def predict():**

**data = request.json # Get JSON data from frontend**

**features = np.array(data["features"]).reshape(1, -1) # Convert to NumPy array**

**prediction = model.predict(features)[0][0] # Get prediction**

**result = "Diabetic" if prediction > 0.5 else "Non-Diabetic"**

**return jsonify({"prediction": result, "probability": float(prediction)})**

**if \_\_name\_\_ == "\_\_main\_\_":**

**app.run(debug=True, port=5000)**

**📌 Run the API**

**cmd**

**CopyEdit**

**python server.py**

**✅ Your Flask API will run at: http://127.0.0.1:5000**

**2️⃣ Frontend: Call API from JavaScript (Node.js + React or Vanilla JS)**

**📌 Frontend Code (HTML + JavaScript)**

**html**

**CopyEdit**

**<!DOCTYPE html>**

**<html lang="en">**

**<head>**

**<meta charset="UTF-8">**

**<meta name="viewport" content="width=device-width, initial-scale=1.0">**

**<title>Diabetes Prediction</title>**

**</head>**

**<body>**

**<h2>Enter Patient Data</h2>**

**<form id="predictForm">**

**<input type="text" id="features" placeholder="Enter values comma-separated">**

**<button type="submit">Predict</button>**

**</form>**

**<h3 id="result"></h3>**

**<script>**

**document.getElementById("predictForm").addEventListener("submit", function(event) {**

**event.preventDefault();**

**let features = document.getElementById("features").value.split(",").map(Number);**

**fetch("http://127.0.0.1:5000/predict", {**

**method: "POST",**

**headers: { "Content-Type": "application/json" },**

**body: JSON.stringify({ features: features })**

**})**

**.then(response => response.json())**

**.then(data => {**

**document.getElementById("result").innerText =**

**`Prediction: ${data.prediction} (Probability: ${data.probability.toFixed(2)})`;**

**});**

**});**

**</script>**

**</body>**

**</html>**

**✅ User enters features → Clicks Predict → Sees Result**

**🚀 Next Steps**

1. **Improve Accuracy: Tune your deep learning model**
2. **Deploy API & Web: Use Flask with Gunicorn (backend) and React/Vue (frontend)**
3. **Use Docker: Package API into a Docker container for easy deployment**
4. **Deploy to Cloud: Use AWS, Heroku, or Render**

**Do you want help deploying it?**

**VERSION 4**

**1. improve accuracy 2. using docker**

**ChatGPT said:**

**Great! 🚀 Let's tackle both:**

**✅ 1. Improve Accuracy of Your Deep Learning Model**

**We can improve accuracy by:  
✅ Using more layers & neurons  
✅ Adding Batch Normalization  
✅ Using LeakyReLU instead of ReLU  
✅ Using AdamW optimizer (better weight decay)  
✅ Training with early stopping**

**📌 Improved Model (train\_model.py)**

**python**

**CopyEdit**

**import tensorflow as tf**

**from tensorflow.keras.models import Sequential**

**from tensorflow.keras.layers import Dense, Dropout, BatchNormalization**

**from tensorflow.keras.optimizers import AdamW**

**import pandas as pd**

**import numpy as np**

**from sklearn.model\_selection import train\_test\_split**

**from sklearn.preprocessing import StandardScaler**

**# Load dataset**

**url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv"**

**columns = ["Pregnancies", "Glucose", "BloodPressure", "SkinThickness", "Insulin", "BMI", "DiabetesPedigreeFunction", "Age", "Outcome"]**

**df = pd.read\_csv(url, names=columns)**

**# Split data**

**X = df.iloc[:, :-1].values**

**y = df.iloc[:, -1].values**

**X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)**

**# Normalize data**

**scaler = StandardScaler()**

**X\_train = scaler.fit\_transform(X\_train)**

**X\_test = scaler.transform(X\_test)**

**# Build improved deep learning model**

**def create\_model():**

**model = Sequential([**

**Dense(128, input\_shape=(X\_train.shape[1],)),**

**BatchNormalization(),**

**tf.keras.layers.LeakyReLU(alpha=0.1),**

**Dropout(0.4),**

**Dense(64),**

**BatchNormalization(),**

**tf.keras.layers.LeakyReLU(alpha=0.1),**

**Dropout(0.3),**

**Dense(32),**

**BatchNormalization(),**

**tf.keras.layers.LeakyReLU(alpha=0.1),**

**Dense(16, activation="relu"),**

**Dense(1, activation="sigmoid")**

**])**

**model.compile(optimizer=AdamW(learning\_rate=0.001), loss="binary\_crossentropy", metrics=["accuracy"])**

**return model**

**# Train the model**

**model = create\_model()**

**early\_stopping = tf.keras.callbacks.EarlyStopping(monitor="val\_loss", patience=10, restore\_best\_weights=True)**

**model.fit(X\_train, y\_train, epochs=100, validation\_data=(X\_test, y\_test), batch\_size=16, callbacks=[early\_stopping])**

**# Save the model**

**model.save("diabetes\_model.h5")**

**✅ Expected Improvement: ~5-10% better accuracy!**

**✅ 2. Deploy with Docker**

**Step 1: Create a Flask API (server.py)**

**python**

**CopyEdit**

**from flask import Flask, request, jsonify**

**import numpy as np**

**import tensorflow as tf**

**app = Flask(\_\_name\_\_)**

**# Load trained model**

**model = tf.keras.models.load\_model("diabetes\_model.h5")**

**@app.route("/predict", methods=["POST"])**

**def predict():**

**data = request.json**

**features = np.array(data["features"]).reshape(1, -1)**

**prediction = model.predict(features)[0][0]**

**result = "Diabetic" if prediction > 0.5 else "Non-Diabetic"**

**return jsonify({"prediction": result, "probability": float(prediction)})**

**if \_\_name\_\_ == "\_\_main\_\_":**

**app.run(host="0.0.0.0", port=5000)**

**✅ Step 2: Create Dockerfile**

**dockerfile**

**CopyEdit**

**# Use official Python image**

**FROM python:3.9**

**# Set working directory**

**WORKDIR /app**

**# Copy files**

**COPY server.py diabetes\_model.h5 /app/**

**# Install dependencies**

**RUN pip install flask tensorflow numpy**

**# Expose port**

**EXPOSE 5000**

**# Run server**

**CMD ["python", "server.py"]**

**✅ Step 3: Build and Run Docker Container**

**sh**

**CopyEdit**

**# Build Docker image**

**docker build -t diabetes-api .**

**# Run the container**

**docker run -p 5000:5000 diabetes-api**

**✅ Now, your Flask API runs inside Docker and is accessible at http://localhost:5000/predict**

**🚀 Next Steps**

* **Deploy to Cloud: Use AWS, GCP, or Heroku**
* **Improve Frontend: Use React or Vue**
* **CI/CD Pipeline: Automate updates with GitHub Actions**

**Do you want help deploying to a server (AWS, Render, etc.)? 😊🔥**

**4o**