FOLDER STRUCTURE:

```
predictive_maintenance_project/
│
├── sensor_data/              # Stores simulated sensor CSV files
│   ├── sensor_readings.csv     # Main sensor readings file (timestamps, temperature, vibration, etc.)
│
├── thermal_images/           # Stores all captured thermal images
│   ├── image1.jpg
│   ├── image2.jpg
│   ├── ...
│
├── scripts/                  # Python scripts for processing
│   ├── edge_capture.py         # Captures data from edge device
│   ├── thermal_processing.py   # Classifies images (normal/anomaly)
│   ├── test_api.py             # API testing script
│
├── notebooks/                # Jupyter notebooks for model testing/training
│   ├── train_cnn.py            # CNN training script
│   ├── dataset/                # Training dataset for model
│       ├── anomaly/              # Anomalous images for training
│       ├── normal/               # Normal images for training
│
├── api/                      # Flask API for cloud deployment
│   ├── server.py               # Main API server

│
├── dashboard/                # Streamlit dashboard for monitoring
│   ├── dashboard_app.py        # Main dashboard script
│
├── requirements.txt          # Dependencies list
```

dashboard.py:-

```
import os
import streamlit as st
import pandas as pd
import time
import sys
import cv2
import numpy as np
import base64
import matplotlib.pyplot as plt
import seaborn as sns

# Ensure scripts folder is accessible
```

```python
sys.path.append(os.path.abspath(os.path.join(os.path.dirname(__file__), "..")))
from scripts.thermal_processing import classify_image  # Import classification function

# ================================
# Define Paths
# ================================
BASE_DIR = os.path.dirname(os.path.abspath(__file__))  # dashboard/
PROJECT_DIR = os.path.dirname(BASE_DIR)  # predictive_maintenance_project/
THERMAL_DIR = os.path.join(PROJECT_DIR, "thermal_images")  # Outside notebooks/
SENSOR_DIR = os.path.join(PROJECT_DIR, "sensor_data")  # Outside notebooks/
SENSOR_FILE = os.path.join(SENSOR_DIR, "sensor_readings.csv")

# ================================
# Streamlit UI
# ================================
st.set_page_config(page_title="HEMM Thermal Monitoring", layout="centered")  # 🔹 Reduce window size
st.title("🚜 HEMM Predictive Maintenance Dashboard")
st.sidebar.header("📌 Options")

# ================================
# Auto Refresh Section (Every 5 Seconds)
# ================================
st.sidebar.subheader("🔄 Auto Refresh: **Every 5 Seconds**")

# Placeholder elements for live updates
sensor_table = st.empty()  # Live sensor table

# ================================
# Utility Functions
# ================================
def apply_colormap(image_path):
    """Apply false color mapping to enhance thermal images."""
    img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)  # Read in grayscale
    img = cv2.applyColorMap(img, cv2.COLORMAP_JET)  # Apply colormap
    return img  # Return colorized image

def get_thumbnail_html(image_filename, width=50):
    """Generate an HTML thumbnail for the image using false color mapping."""
    image_path = os.path.join(THERMAL_DIR, image_filename)
    if os.path.exists(image_path):
        colored_img = apply_colormap(image_path)  # Colorized version
        _, buffer = cv2.imencode(".jpg", colored_img)  # Encode as JPEG
        base64_img = base64.b64encode(buffer).decode()
```

```
        return f'<a href="{image_path}" target="_blank"><img
src="{base64_img}" width="{width}"></a>'
    return "N/A"

def get_condition_indicator(image_filename):
    """Return an HTML snippet showing a green tick if normal, red alert if
anomaly."""
    image_path = os.path.join(THERMAL_DIR, image_filename)
    if os.path.exists(image_path):
        label, confidence = classify_image(image_path)
        confidence_percent = float(confidence) * 100  # Convert to percentage
        if label.lower() == "normal":
            return f'<span style="color: green; font-weight: bold;">✔️ Normal
({confidence_percent:.2f}%)</span>'
        else:
            return f'<span style="color: red; font-weight: bold;">🚨 Anomaly
({confidence_percent:.2f}%)</span>'
    return "N/A"

# ================================
# Display Latest Captured Thermal Image
# ================================
st.subheader("🖼️ Latest Captured Thermal Image")

latest_image = None
if os.path.exists(SENSOR_FILE):
    df = pd.read_csv(SENSOR_FILE)
    if not df.empty:
        latest_image = df.iloc[-1]["Image_Filename"]  # Get the latest image
        image_path = os.path.join(THERMAL_DIR, latest_image)
    else:
        image_path = None
else:
    image_path = None

if image_path and os.path.exists(image_path):
    colored_img = apply_colormap(image_path)
    st.image(colored_img, caption="Latest Thermal Image (Colorized)",
channels="BGR", use_container_width=False, width=400)
else:
    st.warning("No thermal image available. Run `edge_capture.py` first.")

# ================================
# Automatic Classification of Latest Image (Normal vs Anomaly)
# ================================
```

```
if image_path and os.path.exists(image_path):
    st.subheader("🔍 Automatic Classification Result")
    label, confidence = classify_image(image_path)  # Get prediction and confidence
score
    confidence_percent = float(confidence)  # Convert to percentage

    if label.lower() == "normal":
        st.success(f"✅ **Normal Condition** ({confidence_percent:.2f})")
    else:
        st.error(f"🚨 **Anomaly Detected!** ({confidence_percent:.2f})")

# ================================
# Fault Prediction Trends (Reduced Window Size)
# ================================
with st.container():
    st.subheader("📈 Fault Prediction Trends")

    if os.path.exists(SENSOR_FILE):
        df = pd.read_csv(SENSOR_FILE)
        df["Timestamp"] = pd.to_datetime(df["Timestamp"])
        df["Anomaly"] = df["Image_Filename"].apply(lambda x: 1 if "anomaly" in
x.lower() else 0)

        anomaly_trend = df.resample("H", on="Timestamp")["Anomaly"].sum()

        fig, ax = plt.subplots(figsize=(8, 4))  # 🔹 Limit graph size
        ax.plot(anomaly_trend.index, anomaly_trend.values, marker="o",
linestyle="-", color="red", label="Anomalies")
        ax.set_title("Anomaly Trends Over Time")
        ax.set_xlabel("Time")
        ax.set_ylabel("Number of Anomalies")
        ax.legend()

        st.pyplot(fig)
    else:
        st.warning("No data available for fault trend analysis.")

# ================================
# Heatmap of Anomalies (Reduced Window Size)
# ================================
with st.container():
    st.subheader("🌡️ Heatmap of Anomalies by Sensor Location")

    if os.path.exists(SENSOR_FILE):
        df = pd.read_csv(SENSOR_FILE)
```

```
        df["Timestamp"] = pd.to_datetime(df["Timestamp"])
        df["Hour"] = df["Timestamp"].dt.hour
        df["Anomaly"] = df["Image_Filename"].apply(lambda x: 1 if "anomaly" in
x.lower() else 0)

        heatmap_data = df.pivot_table(index="Sensor_Location", columns="Hour",
values="Anomaly", aggfunc="sum", fill_value=0)

        fig, ax = plt.subplots(figsize=(8, 4))  # 🔹 Reduce graph size
        sns.heatmap(heatmap_data, cmap="Reds", linewidths=0.5, annot=True,
fmt=".0f", ax=ax)
        ax.set_title("Anomalies Heatmap by Sensor Location")

        st.pyplot(fig)
    else:
        st.warning("No sensor data available for heatmap analysis.")

st.sidebar.subheader("🔧 Future Features")
st.sidebar.text("- Live HEMM tracking")
st.sidebar.text("- Temperature heatmaps")
st.sidebar.text("- Fault prediction trends")
while True:
    if os.path.exists(SENSOR_FILE):
        df = pd.read_csv(SENSOR_FILE)
        df = df.sort_values(by="Timestamp", ascending=False).head(10)  # Show latest
10 readings

        # Add clickable thumbnails
        df["Thumbnail"] = df["Image_Filename"].apply(lambda x:
get_thumbnail_html(x))
        # Add condition indicator column
        df["Condition"] = df["Image_Filename"].apply(lambda x:
get_condition_indicator(x))

        # Rearrange columns (optional)
        cols_order = ["Timestamp", "HEMM_ID", "Sensor_Location", "Temperature",
"Vibration", "Condition", "Thumbnail"]
        df = df[cols_order]

        # Display updated table as HTML
        sensor_table.write(df.to_html(escape=False, index=False),
unsafe_allow_html=True)
    else:
        sensor_table.warning("No sensor data available yet.")
```

```
    # 🔄 Refresh every 5 seconds
    time.sleep(5)
    st.rerun()  # Forces Streamlit to reload the content
```

edge_capture.py:-

After running this file thermal_images folder, sensor_data folder, and dataset folder will be automatically created which will contain the data

```
import os
import cv2
import numpy as np
import time
from datetime import datetime
import random

# ================================
# Set Paths (Saving in Notebooks/)
# ================================
BASE_DIR = "notebooks"
DATASET_DIR = os.path.join(BASE_DIR, "dataset")
THERMAL_DIR = "thermal_images"
SENSOR_DIR =  "sensor_data"

# Ensure directories exist
for folder in [DATASET_DIR, THERMAL_DIR, SENSOR_DIR, os.path.join(DATASET_DIR,
"normal"), os.path.join(DATASET_DIR, "anomaly")]:
    os.makedirs(folder, exist_ok=True)
HEMM_IDS = ["HEMM_01", "HEMM_02", "HEMM_03", "HEMM_04", "HEMM_05"]
SENSOR_LOCATIONS = ["Engine", "Hydraulics", "Transmission", "Cooling System",
"Brakes"]

# ================================
# Simulate Thermal Image Capture
# ================================
def generate_thermal_image(image_type="normal"):
    """Generates a more realistic thermal image with better visual
representation."""
    img = np.zeros((224, 224), dtype=np.uint8)  # Start with a black (cold) base
image

    if image_type == "anomaly":
        # Generate hotspots (bright white/yellow zones)
        num_hotspots = random.randint(2, 5)  # Randomly place 2-5 hotspots
        for _ in range(num_hotspots):
```

```
            x, y = random.randint(50, 174), random.randint(50, 174)  # Random center
            radius = random.randint(20, 50)
            intensity = random.randint(180, 255)  # Bright intensity for hot zones
            cv2.circle(img, (x, y), radius, (intensity,), -1, lineType=cv2.LINE_AA)

    else:  # Normal image
        # Generate smooth temperature variations using gradients
        x_gradient = np.tile(np.linspace(100, 180, 224, dtype=np.uint8), (224, 1))
        y_gradient = np.tile(np.linspace(100, 180, 224, dtype=np.uint8), (224, 1)).T
        img = cv2.addWeighted(x_gradient, 0.5, y_gradient, 0.5, 0)

    # Apply false color mapping (thermal effect)
    img_colored = cv2.applyColorMap(img, cv2.COLORMAP_JET)

    filename = f"{image_type}_{datetime.now().strftime('%Y%m%d_%H%M%S')}.jpg"
    dataset_path = os.path.join(DATASET_DIR, image_type, filename)
    thermal_path = os.path.join(THERMAL_DIR, filename)

    cv2.imwrite(dataset_path, img_colored)
    cv2.imwrite(thermal_path, img_colored)

    print(f"[INFO] Saved thermal image: {dataset_path}")
    return filename



# ================================
# Simulate Sensor Data Generation
# ================================
def generate_sensor_data():
    """Simulates generating sensor readings."""
    hemm_id = random.choice(HEMM_IDS)  # Random HEMM ID
    sensor_location = random.choice(SENSOR_LOCATIONS)  # Random Sensor Location

    temperature = round(random.uniform(50, 100), 2)  # Temperature range
    vibration = round(random.uniform(0.5, 5.0), 2)  # Vibration range

    # Determine image type
    image_type = "normal" if temperature < 80 and vibration < 4 else "anomaly"
    image_filename = generate_thermal_image(image_type)

    # Prepare data
    timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
```

```
    data_line =
f"{timestamp},{hemm_id},{sensor_location},{temperature},{vibration},{image_filename
}\n"

    # Save to CSV
    sensor_file = os.path.join(SENSOR_DIR, "sensor_readings.csv")
    write_header = not os.path.exists(sensor_file)

    with open(sensor_file, "a") as f:
        if write_header:

f.write("Timestamp,HEMM_ID,Sensor_Location,Temperature,Vibration,Image_Filename\n")
        f.write(data_line)

    print(f"[INFO] Recorded sensor data: {data_line.strip()}")

# ================================
# Continuous Data Generation
# ================================
if __name__ == "__main__":
    print("[INFO] Starting Edge Capture Simulation...")
    try:
        while True:
            generate_sensor_data()
            time.sleep(5)  # Simulate a new reading every 5 seconds
    except KeyboardInterrupt:
        print("\n[INFO] Stopping Edge Capture Simulation.")
```

thermal_processing.py:-

```
import os
import tensorflow as tf
import numpy as np
import cv2

# Load the trained model
BASE_DIR = os.path.dirname(os.path.abspath(__file__))  # scripts/
MODEL_PATH = os.path.join(BASE_DIR, "../notebooks/scripts/thermal_cnn_model.h5")
model = tf.keras.models.load_model(MODEL_PATH)

# Preprocessing function
def preprocess_image(image_path):
    img = cv2.imread(image_path)
    img = cv2.resize(img, (224, 224))
    img = img / 255.0  # Normalize
```

```
    img = np.expand_dims(img, axis=0)  # Add batch dimension
    return img

# Classification function
def classify_image(image_path):
    img = preprocess_image(image_path)  # Convert image for model input
    predictions = model.predict(img)[0]  # ✅ Take the first (only) output

    if len(predictions) == 2:  # If it's a two-class problem
        normal_score, anomaly_score = predictions
        confidence = max(normal_score, anomaly_score)  # Highest confidence
        label = "Normal" if normal_score > anomaly_score else "Anomaly"

    else:  # Handle single-class sigmoid case
        anomaly_score = predictions[0]
        confidence = anomaly_score if anomaly_score > 0.5 else 1 - anomaly_score
        label = "Anomaly" if anomaly_score > 0.5 else "Normal"

    return label, confidence  # Return both label and confidence
```

server.py:-

```
from flask import Flask, request, jsonify
import os
import pandas as pd

app = Flask(__name__)

# Ensure a folder exists to save incoming sensor data
os.makedirs("received_sensor_data", exist_ok=True)

@app.route('/upload_sensor', methods=['POST'])
def upload_sensor():
    data = request.get_json()
    if not data:
        return jsonify({"error": "No data received"}), 400

    # Convert received JSON data to DataFrame
    try:
        sensor_df = pd.DataFrame([data])
        # Save the DataFrame as a CSV file with a unique name (using timestamp)
        filename =
f"received_sensor_data/sensor_{pd.Timestamp.now().strftime('%Y%m%d_%H%M%S')}.csv"
        sensor_df.to_csv(filename, index=False)
```

```
        return jsonify({"status": "success", "message": f"Data saved as
{filename}"}), 200
    except Exception as e:
        return jsonify({"error": str(e)}), 500

@app.route('/')
def home():
    return "Flask API for Predictive Maintenance is Running!"

if __name__ == '__main__':
    app.run(debug=True, port=5000)
```

train_cnn.ipynb

```
# %%
import os
print("Anomaly images:", os.listdir("../dataset/anomaly"))
print("Normal images:", os.listdir("../dataset/normal"))



# %%
import os

print("Current Working Directory:", os.getcwd())
print("Dataset Exists:", os.path.exists("../dataset"))
print("Anomaly Folder Exists:", os.path.exists("../dataset/anomaly"))
print("Normal Folder Exists:", os.path.exists("../dataset/normal"))

print("Anomaly images:", len(os.listdir("../dataset/anomaly")))
print("Normal images:", len(os.listdir("../dataset/normal")))

# List a few image names to confirm
print("Sample anomaly images:", os.listdir("../dataset/anomaly")[:5])
print("Sample normal images:", os.listdir("../dataset/normal")[:5])



# %%
import os

anomaly_files = os.listdir("../dataset/anomaly")
normal_files = os.listdir("../dataset/normal")

# Check file extensions
print("Anomaly file types:", set([f.split('.')[-1] for f in anomaly_files]))
print("Normal file types:", set([f.split('.')[-1] for f in normal_files]))
```

```
# %%
import cv2

# Try opening a sample anomaly image
img_path = "../dataset/anomaly/" + os.listdir("../dataset/anomaly")[0]  # First
image
img = cv2.imread(img_path)

if img is None:
   print("Image is unreadable. Possibly corrupted:", img_path)
else:
   print("Image loaded successfully!")


# %%
import os
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import matplotlib.pyplot as plt

# ✅ Correct dataset path (directly pointing to "dataset/")
dataset_dir = "../dataset"

# ✅ Set parameters
IMG_SIZE = (224, 224)
BATCH_SIZE = 32

train_datagen = ImageDataGenerator(rescale=1./255, validation_split=0.2)
valid_datagen = ImageDataGenerator(rescale=1./255, validation_split=0.2)

# Change the directory to just "dataset/"
train_generator = train_datagen.flow_from_directory(
   directory="../dataset",
   target_size=(224, 224),
   batch_size=32,
   class_mode="binary",
   subset="training"
)

validation_generator = valid_datagen.flow_from_directory(
   directory="../dataset",
   target_size=(224, 224),
   batch_size=32,
```

```
    class_mode="binary",
    subset="validation"
)

# ✅ Check class labels
print("Class indices:", train_generator.class_indices)



# %%
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout

# Define CNN model
model = Sequential([
    Conv2D(32, (3,3), activation="relu", input_shape=(224, 224, 3)),
    MaxPooling2D(2,2),

    Conv2D(64, (3,3), activation="relu"),
    MaxPooling2D(2,2),

    Conv2D(128, (3,3), activation="relu"),
    MaxPooling2D(2,2),

    Flatten(),
    Dense(128, activation="relu"),
    Dropout(0.5),
    Dense(1, activation="sigmoid")  # Binary classification (normal vs anomaly)
])

# Compile model
model.compile(optimizer="adam", loss="binary_crossentropy", metrics=["accuracy"])

# Display model summary
model.summary()



# %%
# Train the model
history = model.fit(
    train_generator,
    steps_per_epoch=len(train_generator),
    validation_data=validation_generator,
    validation_steps=len(validation_generator),
    epochs=10
)
```

```
# %%
# Plot accuracy and loss graphs
plt.figure(figsize=(12,5))

plt.subplot(1,2,1)
plt.plot(history.history["accuracy"], label="Train Accuracy")
plt.plot(history.history["val_accuracy"], label="Validation Accuracy")
plt.legend()
plt.xlabel("Epoch")
plt.ylabel("Accuracy")
plt.title("Training vs Validation Accuracy")

plt.subplot(1,2,2)
plt.plot(history.history["loss"], label="Train Loss")
plt.plot(history.history["val_loss"], label="Validation Loss")
plt.legend()
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.title("Training vs Validation Loss")

plt.show()


# %%
# Save the model
model.save("../scripts/thermal_cnn_model.h5")
print("✅ Model saved as thermal_cnn_model.h5")
```

test.api:-

```
import requests
import json

# Define the URL of the API endpoint
url = "http://127.0.0.1:5000/upload_sensor"

# Create a sample sensor data payload (this mimics what your edge device
might send)
```

```
sample_data = {
    "Timestamp": "2023-01-01 00:30:00",
    "Vibration (g)": 0.55,
    "Oil Pressure (PSI)": 32,
    "RPM": 1520
}

# Send a POST request with the sample data
response = requests.post(url, json=sample_data)

# Print the response from the API
print("Status Code:", response.status_code)
print("Response JSON:", response.json())
```

Steps to run:

```
In api folder (open in integrated terminal)
source ../venv/bin/activate
python server.py

In main folder(open in integrated terminal)

source venv/bin/activate
python edge_capture.py

source venv/bin/activate
streamlit run dashboard/dashboard_app.py
```