**Equipment Maintenance Prediction Folder Structure**

Equipment\_maintenance\_prediction/

├── charts/

│ └── trend\_graph.png

├── fastapi\_app/

│ ├── \_\_pycache\_\_/

│ ├── auth.py

│ ├── calendar.py

│ ├── database.py

│ ├── dependencies.py

│ ├── equipments.py

│ ├── hash\_password.py

│ ├── llm\_engine.py

│ ├── main.py

│ ├── maintenance.py

│ ├── models.py

│ ├── predict.py

│ └── users.py

├── saved\_models/

│ ├── lgbm\_model.pkl

│ ├── lstm\_model.h5

│ ├── scaler.pkl

│ ├── corrective\_model.pkl

│ ├── preventive\_model.pkl

│ ├── replacement\_model.pkl

│ ├── multi\_priority\_scaler.pkl

├── Current code structure and its codes.docx

├── daily\_predict.py

├── equipment\_priority\_features.csv

├── generate\_equipment\_report.py

├── generate\_priority\_features.py

├── generate\_priority\_lables.py

├── hospital\_equipment\_system.db

├── labeled\_corrective\_data.csv

├── labeled\_preventive\_data.csv

├── labeled\_replacement\_data.csv

├── ml\_models.py

├── preprocess.py

├── processed\_equipment\_data.csv

├── schema.py

├── train\_priority\_model.py

└── xyz.py

**Database schema:**

**Database: hospital\_equipment\_system.db**

Table: equipment

   - equipment\_id (TEXT) [PK]

   - type (TEXT)

   - manufacturer (TEXT)

   - location (TEXT)

   - criticality (TEXT)

   - installation\_date (TEXT)

Table: personnel

   - personnel\_id (TEXT) [PK]

   - name (TEXT)

   - role (TEXT)

   - department (TEXT)

   - experience\_years (REAL)

   - username (TEXT)

   - password (TEXT)

Table: equipment\_assignments

   - assignment\_id (TEXT) [PK]

   - equipment\_id (TEXT)

   - personnel\_id (TEXT)

Table: sqlite\_sequence

   - name ()

   - seq ()

Table: failure\_predictions

   - prediction\_id (INTEGER) [PK]

   - equipment\_id (TEXT)

   - prediction\_date (TEXT)

   - needs\_maintenance\_10\_days (INTEGER)

   - failure\_probability (REAL)

Table: high\_error\_state

   - equipment\_id (TEXT) [PK]

   - start\_date (TEXT)

   - streak\_days (INTEGER)

Table: usage\_logs

   - log\_id (INTEGER)

   - equipment\_id (TEXT)

   - timestamp (TIMESTAMP)

   - usage\_hours (REAL)

   - patients\_served (REAL)

   - workload\_level (REAL)

   - avg\_cpu\_temp (REAL)

   - error\_count (REAL)

Table: maintenance\_logs

   - maintenance\_id (TEXT) [PK]

   - equipment\_id (TEXT)

   - date (TEXT)

   - maintenance\_type (TEXT)

   - downtime\_hours (REAL)

   - cost\_inr (REAL)

   - issue\_description (TEXT)

   - parts\_replaced (TEXT)

   - vendor (TEXT)

   - technician\_id (TEXT)

   - service\_rating (INTEGER)

   - response\_time\_hours (REAL)

   - completion\_status (TEXT)

   - warranty\_covered (TEXT)

   - status (TEXT)

**CODES:**

**preprocess.py:**

import sqlite3

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

# Connect to the DB

conn = sqlite3.connect("hospital\_equipment\_system.db")

# Load data

usage\_df = pd.read\_sql\_query("SELECT \* FROM usage\_logs", conn)

pred\_df = pd.read\_sql\_query("SELECT \* FROM failure\_predictions", conn)

equip\_df = pd.read\_sql\_query("SELECT \* FROM equipment", conn)

# Convert timestamp and dates

usage\_df["timestamp"] = pd.to\_datetime(usage\_df["timestamp"])

pred\_df["prediction\_date"] = pd.to\_datetime(pred\_df["prediction\_date"])

equip\_df["installation\_date"] = pd.to\_datetime(equip\_df["installation\_date"])

# Sort usage logs

usage\_df = usage\_df.sort\_values(["equipment\_id", "timestamp"])

# Merge usage with predictions for target label

# We assume prediction was made based on past 5-day logs — so we align on dates

merged\_df = pd.merge\_asof(

    usage\_df.sort\_values("timestamp"),

    pred\_df.sort\_values("prediction\_date"),

    by="equipment\_id",

    left\_on="timestamp",

    right\_on="prediction\_date",

    direction="forward",

    tolerance=pd.Timedelta("10D")  # only if prediction happens within next 10 days

)

# Drop rows without labels (NaN)

merged\_df = merged\_df.dropna(subset=["needs\_maintenance\_10\_days"])

# View info

print(merged\_df.head())

print(merged\_df.info())

# Check missing values

print("\nMissing values:")

print(merged\_df.isnull().sum())

# EDA: Plot correlations

plt.figure(figsize=(10, 6))

sns.heatmap(merged\_df.select\_dtypes(include="number").corr(), annot=True, cmap="coolwarm")

plt.title("Feature Correlation Heatmap")

plt.tight\_layout()

plt.show()

# Plot class distribution

sns.countplot(x="needs\_maintenance\_10\_days", data=merged\_df)

plt.title("Target Distribution (Maintenance Needed in 10 Days)")

plt.show()

# Distribution of numeric features

numeric\_cols = ["usage\_hours", "patients\_served", "workload\_level", "avg\_cpu\_temp", "error\_count"]

for col in numeric\_cols:

    plt.figure(figsize=(6, 4))

    sns.histplot(data=merged\_df, x=col, kde=True)

    plt.title(f"Distribution of {col}")

    plt.tight\_layout()

    plt.show()

# Close DB connection

conn.close()

merged\_df.to\_csv("processed\_equipment\_data.csv", index=False)

**ml\_models.py**

# ml\_model2.py

import numpy as np

import pandas as pd

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

from sklearn.metrics import classification\_report, confusion\_matrix, roc\_auc\_score, roc\_curve

from lightgbm import LGBMClassifier

import matplotlib.pyplot as plt

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import LSTM, Dense, Input

from tensorflow.keras.callbacks import EarlyStopping

from sklearn.preprocessing import StandardScaler

# Data loading

df = pd.read\_csv("processed\_equipment\_data.csv")

df["timestamp"] = pd.to\_datetime(df["timestamp"])

features = ["usage\_hours", "patients\_served", "workload\_level", "avg\_cpu\_temp", "error\_count"]

target = "needs\_maintenance\_10\_days"

df[features] = df[features].fillna(0)

# Standard scaling

scaler = StandardScaler()

df[features] = scaler.fit\_transform(df[features])

# Rolling window for LSTM

sequences, labels = [], []

grouped = df.groupby("equipment\_id")

for \_, group in grouped:

    group = group.sort\_values("timestamp")

    if len(group) >= 6:

        for i in range(len(group) - 5):

            sequences.append(group.iloc[i:i+5][features].values)

            labels.append(group.iloc[i+5][target])

X\_seq = np.array(sequences)

y\_seq = np.array(labels)

# Train/test split for LSTM

X\_train\_seq, X\_test\_seq, y\_train\_seq, y\_test\_seq = train\_test\_split(

    X\_seq, y\_seq, test\_size=0.2, random\_state=42, stratify=y\_seq)

# Flatten for traditional models

X\_flat = X\_seq.reshape(X\_seq.shape[0], -1)

X\_train\_flat, X\_test\_flat, y\_train, y\_test = train\_test\_split(

    X\_flat, y\_seq, test\_size=0.2, random\_state=42, stratify=y\_seq)

# --- LSTM Model ---

lstm\_model = Sequential([

    Input(shape=(5, len(features))),

    LSTM(64),

    Dense(1, activation='sigmoid')

])

lstm\_model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy'])

early\_stop = EarlyStopping(monitor='val\_loss', patience=5, restore\_best\_weights=True)

lstm\_model.fit(

    X\_train\_seq, y\_train\_seq,

    validation\_split=0.2,

    epochs=50,

    batch\_size=32,

    callbacks=[early\_stop],

    verbose=0

)

y\_pred\_lstm\_prob = lstm\_model.predict(X\_test\_seq).flatten()

y\_pred\_lstm = (y\_pred\_lstm\_prob > 0.5).astype(int)

print("\nLSTM Model:")

print(classification\_report(y\_test\_seq, y\_pred\_lstm))

print("Confusion Matrix:\n", confusion\_matrix(y\_test\_seq, y\_pred\_lstm))

print("ROC-AUC:", roc\_auc\_score(y\_test\_seq, y\_pred\_lstm\_prob))

fpr, tpr, \_ = roc\_curve(y\_test\_seq, y\_pred\_lstm\_prob)

plt.plot(fpr, tpr, label="LSTM")

# --- LightGBM Model ---

lgbm = LGBMClassifier()

lgbm.fit(X\_train\_flat, y\_train)

y\_pred\_lgbm\_prob = lgbm.predict\_proba(X\_test\_flat)[:, 1]

y\_pred\_lgbm = (y\_pred\_lgbm\_prob > 0.5).astype(int)

print("\nLightGBM Model:")

print(classification\_report(y\_test, y\_pred\_lgbm))

print("Confusion Matrix:\n", confusion\_matrix(y\_test, y\_pred\_lgbm))

print("ROC-AUC:", roc\_auc\_score(y\_test, y\_pred\_lgbm\_prob))

fpr, tpr, \_ = roc\_curve(y\_test, y\_pred\_lgbm\_prob)

plt.plot(fpr, tpr, label="LightGBM")

# --- Ensemble: LSTM + LightGBM ---

y\_pred\_ensemble\_prob = (y\_pred\_lstm\_prob + y\_pred\_lgbm\_prob) / 2

y\_pred\_ensemble = (y\_pred\_ensemble\_prob > 0.40).astype(int)

print("\nEnsemble Model (LSTM + LightGBM):")

print(classification\_report(y\_test, y\_pred\_ensemble))

print("Confusion Matrix:\n", confusion\_matrix(y\_test, y\_pred\_ensemble))

print("ROC-AUC:", roc\_auc\_score(y\_test, y\_pred\_ensemble\_prob))

fpr, tpr, \_ = roc\_curve(y\_test, y\_pred\_ensemble\_prob)

plt.plot(fpr, tpr, label="Ensemble")

# Save LSTM

lstm\_model.save("lstm\_model.h5")

# Save LightGBM

import joblib

joblib.dump(lgbm, "lgbm\_model.pkl")

# Save Scaler

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

# --- Final ROC Plot ---

plt.title("ROC-AUC Curves")

plt.xlabel("False Positive Rate")

plt.ylabel("True Positive Rate")

plt.legend()

plt.grid(True)

plt.tight\_layout()

plt.show()

# --- CUSTOM INPUT TESTING SECTION ---

# Custom Input Set 1: Risky

input1 = pd.DataFrame([

    [5.5, 18, 0.60, 56.50, 1],

    [6.3, 20, 0.70, 58.30, 2],

    [7.1, 22, 0.80, 60.10, 3],

    [8.0, 24, 0.90, 62.00, 4],

    [9.0, 26, 1.00, 64.00, 5]

], columns=features)

# Custom Input Set 2: Safe

input2 = pd.DataFrame([

    [4.87, 18, 0.49, 55.21, 0],

    [3.10, 12, 0.31, 45.84, 0],

    [3.08, 8, 0.31, 48.86, 0],

    [5.90, 17, 0.59, 47.91, 0],

    [8.84, 15, 0.88, 64.87, 1]

], columns=features)

def predict\_custom\_input(input\_df, label):

    # Scale using training scaler

    input\_scaled = scaler.transform(input\_df)

    # LSTM: shape (1, 5, 5)

    input\_lstm = input\_scaled.reshape(1, 5, len(features))

    pred\_lstm\_prob = lstm\_model.predict(input\_lstm).flatten()[0]

    # LightGBM: shape (1, 25)

    input\_flat = input\_scaled.reshape(1, -1)

    pred\_lgbm\_prob = lgbm.predict\_proba(input\_flat)[:, 1][0]

    # Ensemble

    ensemble\_prob = (pred\_lstm\_prob + pred\_lgbm\_prob) / 2

    ensemble\_pred = int(ensemble\_prob > 0.40)

    print(f"\n {label} Prediction")

    print(f"LSTM prob:     {pred\_lstm\_prob:.4f}")

    print(f"LightGBM prob: {pred\_lgbm\_prob:.4f}")

    print(f"Ensemble prob: {ensemble\_prob:.4f}")

    print(f"Final Prediction: {ensemble\_pred} (1 = Needs Maintenance, 0 = No)")

# Run on custom inputs

predict\_custom\_input(input1, "Input Set 1 (Risky)")

predict\_custom\_input(input2, "Input Set 2 (Safe)")

**generate\_priority\_features.py**

# generate\_priority\_features.py

import sqlite3

import pandas as pd

from datetime import datetime

conn = sqlite3.connect("hospital\_equipment\_system.db")

# Equipment Age

equipment\_df = pd.read\_sql("SELECT equipment\_id, installation\_date FROM equipment", conn)

equipment\_df["installation\_date"] = pd.to\_datetime(equipment\_df["installation\_date"])

equipment\_df["equipment\_age"] = (pd.Timestamp.today() - equipment\_df["installation\_date"]).dt.days // 365

# Downtime and Failures

maintenance\_df = pd.read\_sql("SELECT equipment\_id, downtime\_hours, response\_time\_hours FROM maintenance\_logs", conn)

agg\_maintenance = maintenance\_df.groupby("equipment\_id").agg({

    "downtime\_hours": "sum",

    "response\_time\_hours": "mean"

}).reset\_index()

agg\_maintenance["num\_failures"] = maintenance\_df.groupby("equipment\_id").size().values

# Failure prediction from model

failure\_df = pd.read\_sql("SELECT equipment\_id, needs\_maintenance\_10\_days FROM failure\_predictions", conn)

# Merge all

df = equipment\_df.merge(agg\_maintenance, on="equipment\_id", how="left")

df = df.merge(failure\_df, on="equipment\_id", how="left")

df = df.fillna({

    "downtime\_hours": 0,

    "response\_time\_hours": 0,

    "num\_failures": 0,

    "needs\_maintenance\_10\_days": 0  # Default: not predicted to fail

})

df.to\_csv("equipment\_priority\_features.csv", index=False)

print(" Saved: equipment\_priority\_features.csv")

**generate\_priority\_labels.py**

# generate\_priority\_labels.py

import pandas as pd

df = pd.read\_csv("equipment\_priority\_features.csv")

# Use predictive + historical features

features = [

    "equipment\_age",

    "downtime\_hours",

    "num\_failures",

    "response\_time\_hours",

    "needs\_maintenance\_10\_days"

]

def assign\_label\_by\_quantile(series):

    q1 = series.quantile(0.33)

    q2 = series.quantile(0.66)

    def label(value):

        if value <= q1: return "Low"

        elif value <= q2: return "Medium"

        else: return "High"

    return series.apply(label)

# Assign based on combined indicators

df["preventive\_label"] = assign\_label\_by\_quantile(df["equipment\_age"])

df["corrective\_label"] = assign\_label\_by\_quantile(df["num\_failures"] + df["downtime\_hours"] + 50 \* df["needs\_maintenance\_10\_days"])

df["replacement\_label"] = assign\_label\_by\_quantile(df["equipment\_age"] + df["num\_failures"] + 30 \* df["needs\_maintenance\_10\_days"])

# Save labeled datasets

df[features + ["preventive\_label"]].to\_csv("labeled\_preventive\_data.csv", index=False)

df[features + ["corrective\_label"]].to\_csv("labeled\_corrective\_data.csv", index=False)

df[features + ["replacement\_label"]].to\_csv("labeled\_replacement\_data.csv", index=False)

print("Saved quantile-based label files for all 3 types.")

# Show counts

for mtype in ["preventive", "corrective", "replacement"]:

    print(f"{mtype.capitalize()}:", df[f"{mtype}\_label"].value\_counts().to\_dict())

**train\_priority\_model.py**

# train\_priority\_models.py

import pandas as pd

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

from sklearn.svm import SVC

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import classification\_report

import joblib

# Updated feature list includes prediction

features = [

    "equipment\_age",

    "downtime\_hours",

    "num\_failures",

    "response\_time\_hours",

    "needs\_maintenance\_10\_days"

]

scaler = StandardScaler()

for mtype in ["preventive", "corrective", "replacement"]:

    df = pd.read\_csv(f"labeled\_{mtype}\_data.csv")

    X = df[features]

    y = df[f"{mtype}\_label"].map({"Low": 0, "Medium": 1, "High": 2})

    if y.nunique() < 2:

        print(f"Skipping {mtype} model training: only 1 class present ({y.unique()[0]})")

        continue

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

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

    model = SVC(kernel="rbf", probability=True)

    model.fit(X\_train, y\_train)

    print(f"\n{mtype.upper()} MODEL:")

    print(classification\_report(y\_test, model.predict(X\_test)))

    joblib.dump(model, f"saved\_models/{mtype}\_model.pkl")

# Save scaler

joblib.dump(scaler, "saved\_models/multi\_priority\_scaler.pkl")

print("Training complete.")

**daily\_predict.py:**

import numpy as np

import pandas as pd

import joblib

import sqlite3

from tensorflow.keras.models import load\_model

from sklearn.preprocessing import StandardScaler

# --- Load saved models and scaler ---

lstm\_model = load\_model('saved\_models/lstm\_model.h5')

lgbm\_model = joblib.load('saved\_models/lgbm\_model.pkl')

scaler = joblib.load('saved\_models/scaler.pkl')

# --- Connect to database and fetch latest usage logs ---

conn = sqlite3.connect("hospital\_equipment\_system.db")  # Replace with actual DB path

query = """

SELECT equipment\_id, timestamp, usage\_hours, patients\_served, workload\_level, avg\_cpu\_temp, error\_count

FROM usage\_logs

ORDER BY equipment\_id, timestamp DESC

"""

df = pd.read\_sql\_query(query, conn)

conn.close()

# --- Preprocess ---

features = ["usage\_hours", "patients\_served", "workload\_level", "avg\_cpu\_temp", "error\_count"]

df["timestamp"] = pd.to\_datetime(df["timestamp"])

df = df.sort\_values(["equipment\_id", "timestamp"], ascending=[True, False])

equipment\_ids = df["equipment\_id"].unique()

sequences = []

equipment\_map = []

for eq\_id in equipment\_ids:

    eq\_data = df[df["equipment\_id"] == eq\_id]

    if len(eq\_data) >= 5:

        recent\_logs = eq\_data.head(5).sort\_values("timestamp")  # oldest to newest

        X\_scaled = scaler.transform(recent\_logs[features])

        sequences.append(X\_scaled)

        equipment\_map.append(eq\_id)

if not sequences:

    print("Not enough data for any equipment.")

    exit()

X\_seq = np.array(sequences)

X\_flat = X\_seq.reshape(X\_seq.shape[0], -1)

# --- Predict ---

lstm\_probs = lstm\_model.predict(X\_seq).flatten()

lgbm\_probs = lgbm\_model.predict\_proba(X\_flat)[:, 1]

ensemble\_probs = (lstm\_probs + lgbm\_probs) / 2

ensemble\_preds = (ensemble\_probs > 0.4).astype(int)

# --- Output result ---

result\_df = pd.DataFrame({

    "equipment\_id": equipment\_map,

    "maintenance\_needed": ensemble\_preds,

    "confidence\_score": ensemble\_probs

})

print(result\_df)

**generate\_equipment\_report.py**

import sqlite3

import pandas as pd

import matplotlib.pyplot as plt

import os

from datetime import datetime

import warnings

warnings.filterwarnings('ignore')

DB\_PATH = "hospital\_equipment\_system.db"

CHARTS\_DIR = "charts"

os.makedirs(CHARTS\_DIR, exist\_ok=True)

def fetch\_equipment\_metrics(equipment\_id: str):

    conn = sqlite3.connect(DB\_PATH)

    # 1. Equipment Age

    eq\_df = pd.read\_sql("SELECT equipment\_id, installation\_date FROM equipment WHERE equipment\_id = ?", conn, params=(equipment\_id,))

    if eq\_df.empty:

        raise ValueError(f"No equipment found for ID: {equipment\_id}")

    eq\_df["installation\_date"] = pd.to\_datetime(eq\_df["installation\_date"])

    eq\_df["equipment\_age"] = (pd.Timestamp.today() - eq\_df["installation\_date"]).dt.days // 365

    # 2. Maintenance metrics

    maint\_df = pd.read\_sql("SELECT \* FROM maintenance\_logs WHERE equipment\_id = ?", conn, params=(equipment\_id,))

    downtime = maint\_df["downtime\_hours"].sum()

    response\_time = maint\_df["response\_time\_hours"].mean() if not maint\_df.empty else 0

    num\_failures = maint\_df.shape[0]

    # 3. Usage logs for plotting trends

    usage\_df = pd.read\_sql("SELECT \* FROM usage\_logs WHERE equipment\_id = ?", conn, params=(equipment\_id,))

    conn.close()

    usage\_df["timestamp"] = pd.to\_datetime(usage\_df["timestamp"])

    if usage\_df.empty:

        raise ValueError(f"No usage logs found for {equipment\_id}")

    usage\_df['date'] = usage\_df['timestamp'].dt.date

    daily\_usage = usage\_df.groupby('date').agg({

        'usage\_hours': 'mean',

        'avg\_cpu\_temp': 'mean',

        'workload\_level': 'mean',

        'error\_count': 'sum',

        'timestamp': 'first'

    }).reset\_index()

    daily\_usage['date'] = pd.to\_datetime(daily\_usage['date'])

    daily\_usage = daily\_usage.sort\_values('date')

    # 4. Classification labels

    pm\_path = "labeled\_preventive\_data.csv"

    cm\_path = "labeled\_corrective\_data.csv"

    rp\_path = "labeled\_replacement\_data.csv"

    pm\_label = pd.read\_csv(pm\_path).set\_index("equipment\_id").loc[equipment\_id, "preventive\_label"]

    cm\_label = pd.read\_csv(cm\_path).set\_index("equipment\_id").loc[equipment\_id, "corrective\_label"]

    rp\_label = pd.read\_csv(rp\_path).set\_index("equipment\_id").loc[equipment\_id, "replacement\_label"]

    # 5. Plot trends and save to charts/trend\_graph.png

    chart\_path = os.path.join(CHARTS\_DIR, "trend\_graph.png")

    fig, axs = plt.subplots(4, 1, figsize=(14, 14), sharex=True)

    axs[0].plot(daily\_usage['date'], daily\_usage['usage\_hours'], marker='o', label='Avg Usage Hours', color='teal')

    axs[0].set\_ylabel("Usage Hours")

    axs[0].set\_title(f"Daily Usage Trend - {equipment\_id}", fontweight='bold')

    axs[0].legend(); axs[0].grid(True)

    axs[1].plot(daily\_usage['date'], daily\_usage['avg\_cpu\_temp'], marker='x', label='Avg CPU Temp', color='coral')

    axs[1].set\_ylabel("CPU Temp (°C)")

    axs[1].legend(); axs[1].grid(True)

    axs[2].plot(daily\_usage['date'], daily\_usage['workload\_level'], marker='s', label='Workload Level', color='purple')

    axs[2].set\_ylabel("Workload Level")

    axs[2].legend(); axs[2].grid(True)

    axs[3].plot(daily\_usage['date'], daily\_usage['error\_count'], marker='^', label='Error Count', color='red')

    axs[3].set\_ylabel("Error Count"); axs[3].set\_xlabel("Date")

    axs[3].legend(); axs[3].grid(True)

    plt.xticks(rotation=45)

    stats = f"""

    Total Days: {len(daily\_usage)}

    Avg Usage Hours: {daily\_usage['usage\_hours'].mean():.1f}

    Avg CPU Temp: {daily\_usage['avg\_cpu\_temp'].mean():.1f}°C

    Avg Workload: {daily\_usage['workload\_level'].mean():.1f}

    Total Errors: {daily\_usage['error\_count'].sum()}

    """

    plt.figtext(0.02, 0.02, stats, fontsize=10,

                bbox=dict(boxstyle="round", facecolor="lightyellow", alpha=0.7))

    plt.tight\_layout()

    plt.subplots\_adjust(bottom=0.15)

    plt.savefig(chart\_path, dpi=300, bbox\_inches='tight')

    plt.close()

    # 6. Return combined metrics for LLM

    return {

        "equipment\_id": equipment\_id,

        "equipment\_age": int(eq\_df["equipment\_age"].iloc[0]),

        "downtime\_hours": float(downtime),

        "num\_failures": int(num\_failures),

        "response\_time\_hours": float(round(response\_time, 2)),

        "predicted\_to\_fail": cm\_label == "High" or rp\_label == "High",

        "maintenance\_needs": {

            "preventive": pm\_label,

            "corrective": cm\_label,

            "replacement": rp\_label,

        },

        "chart\_path": chart\_path

    }

**fastapi\_app/auth.py**

#auth.py

from fastapi import APIRouter, Depends, HTTPException, status

from fastapi.security import OAuth2PasswordRequestForm

from passlib.context import CryptContext

from jose import jwt

from datetime import datetime, timedelta

import sqlite3

SECRET\_KEY = "your-secret-key"

ALGORITHM = "HS256"

ACCESS\_TOKEN\_EXPIRE\_MINUTES = 60

pwd\_context = CryptContext(schemes=["bcrypt"], deprecated="auto")

router = APIRouter()

def verify\_password(plain\_password, hashed\_password):

    return pwd\_context.verify(plain\_password, hashed\_password)

def create\_access\_token(data: dict):

    to\_encode = data.copy()

    expire = datetime.utcnow() + timedelta(minutes=ACCESS\_TOKEN\_EXPIRE\_MINUTES)

    to\_encode.update({"exp": expire})

    return jwt.encode(to\_encode, SECRET\_KEY, algorithm=ALGORITHM)

@router.post("/login")

def login(form\_data: OAuth2PasswordRequestForm = Depends()):

    conn = sqlite3.connect("hospital\_equipment\_system.db")

    cursor = conn.cursor()

    cursor.execute("SELECT username, password, role FROM personnel WHERE username = ?", (form\_data.username,))

    user = cursor.fetchone()

    conn.close()

    if not user or not verify\_password(form\_data.password, user[1]):

        raise HTTPException(status\_code=401, detail="Invalid username or password")

    token = create\_access\_token({"sub": user[0], "role": user[2]})

    return {

        "access\_token": f"Bearer {token}",

        "token\_type": "bearer",

        "role": user[2]

    }

**fastapi\_app/calender.py**

#calendar.py

from fastapi import APIRouter

router = APIRouter()

@router.get("/calendar")

def get\_calendar():

    return {"message": "Calendar endpoint coming soon!"}

**fastapi\_app/database.py**

#database.py

import sqlite3

import os

def get\_db():

    BASE\_DIR = os.path.dirname(os.path.abspath(\_\_file\_\_))

    db\_path = os.path.join(BASE\_DIR, "..", "hospital\_equipment\_system.db")

    return sqlite3.connect(db\_path)

**fastapi\_app/dependencies.py**

#dependencies.py

from fastapi import Depends, HTTPException, status

from fastapi.security import OAuth2PasswordBearer

from jose import JWTError, jwt

oauth2\_scheme = OAuth2PasswordBearer(tokenUrl="login")

SECRET\_KEY = "your-secret-key"

ALGORITHM = "HS256"

def get\_current\_user(token: str = Depends(oauth2\_scheme)):

    credentials\_exception = HTTPException(

        status\_code=status.HTTP\_401\_UNAUTHORIZED,

        detail="Could not validate credentials",

        headers={"WWW-Authenticate": "Bearer"},

    )

    try:

        if not token.startswith("Bearer "):

            raise credentials\_exception

        token = token.split(" ")[1]

        payload = jwt.decode(token, SECRET\_KEY, algorithms=[ALGORITHM])

        username: str = payload.get("sub")

        role: str = payload.get("role").lower()

        if username is None or role is None:

            raise credentials\_exception

        return {"username": username, "role": role}

    except JWTError:

        raise credentials\_exception

def require\_role(\*roles):

    def role\_checker(user: dict = Depends(get\_current\_user)):

        if user["role"] not in roles:

            raise HTTPException(status\_code=403, detail="Unauthorized for this role")

        return user

    return role\_checker

**fastapi\_app/equipments.py**

#equipments.py

from fastapi import APIRouter, HTTPException, Depends, Query

from typing import Optional

from fastapi.responses import JSONResponse

import matplotlib.pyplot as plt

import sqlite3, io, base64

import pandas as pd

from fastapi\_app.dependencies import get\_current\_user, require\_role

from pydantic import BaseModel

router = APIRouter()

def get\_db():

    return sqlite3.connect("hospital\_equipment\_system.db")

# Pydantic model

class EquipmentIn(BaseModel):

    equipment\_id: str

    type: str

    manufacturer: str

    location: str

    criticality: str

    installation\_date: str

@router.get("/")

def list\_equipments(type: Optional[str] = Query(None), location: Optional[str] = Query(None), user=Depends(get\_current\_user)):

    conn = get\_db()

    cursor = conn.cursor()

    query = "SELECT \* FROM equipment WHERE 1=1"

    params = []

    if type:

        query += " AND type = ?"

        params.append(type)

    if location:

        query += " AND location = ?"

        params.append(location)

    if user["role"] == "technician":

        query += " AND equipment\_id IN (SELECT DISTINCT equipment\_id FROM maintenance\_logs WHERE status = 'Scheduled')"

    cursor.execute(query, params)

    rows = cursor.fetchall()

    conn.close()

    return {"equipments": rows}

@router.get("/{equipment\_id}")

def get\_equipment(equipment\_id: str, user=Depends(get\_current\_user)):

    conn = get\_db()

    cursor = conn.cursor()

    if user["role"] == "technician":

        cursor.execute("SELECT COUNT(\*) FROM maintenance\_logs WHERE equipment\_id = ? AND status = 'Scheduled'", (equipment\_id,))

        if cursor.fetchone()[0] == 0:

            raise HTTPException(status\_code=403, detail="Not authorized for this equipment")

    cursor.execute("SELECT \* FROM equipment WHERE equipment\_id = ?", (equipment\_id,))

    row = cursor.fetchone()

    df = pd.read\_sql\_query("SELECT \* FROM usage\_logs WHERE equipment\_id = ? ORDER BY timestamp", conn, params=(equipment\_id,))

    if df.empty:

        conn.close()

        return {"equipment": row, "trend\_plot": None}

    df["timestamp"] = pd.to\_datetime(df["timestamp"])

    plt.figure(figsize=(6, 3))

    plt.plot(df["timestamp"], df["error\_count"], label="Error Count")

    plt.plot(df["timestamp"], df["avg\_cpu\_temp"], label="CPU Temp")

    plt.legend()

    plt.title(f"Trends - {equipment\_id}")

    plt.tight\_layout()

    buf = io.BytesIO()

    plt.savefig(buf, format="png")

    plt.close()

    buf.seek(0)

    img\_base64 = base64.b64encode(buf.read()).decode("utf-8")

    conn.close()

    return {"equipment": row, "trend\_plot": f"data:image/png;base64,{img\_base64}"}

@router.post("/", dependencies=[Depends(require\_role("admin"))])

def add\_equipment(data: EquipmentIn, user=Depends(get\_current\_user)):

    conn = get\_db()

    cursor = conn.cursor()

    cursor.execute("""

        INSERT INTO equipment (equipment\_id, type, manufacturer, location, criticality, installation\_date)

        VALUES (?, ?, ?, ?, ?, ?)

    """, (

        data.equipment\_id, data.type, data.manufacturer,

        data.location, data.criticality, data.installation\_date

    ))

    conn.commit()

    conn.close()

    return {"message": "Equipment added"}

@router.put("/{equipment\_id}", dependencies=[Depends(require\_role("admin"))])

def update\_equipment(equipment\_id: str, data: EquipmentIn, user=Depends(get\_current\_user)):

    conn = get\_db()

    cursor = conn.cursor()

    cursor.execute("""

        UPDATE equipment SET type = ?, manufacturer = ?, location = ?, criticality = ?, installation\_date = ?

        WHERE equipment\_id = ?

    """, (

        data.type, data.manufacturer, data.location,

        data.criticality, data.installation\_date, equipment\_id

    ))

    conn.commit()

    conn.close()

    return {"message": "Equipment updated"}

@router.delete("/{equipment\_id}", dependencies=[Depends(require\_role("admin"))])

def delete\_equipment(equipment\_id: str, user=Depends(get\_current\_user)):

    conn = get\_db()

    cursor = conn.cursor()

    cursor.execute("DELETE FROM equipment WHERE equipment\_id = ?", (equipment\_id,))

    conn.commit()

    conn.close()

    return {"message": "Equipment deleted"}

**fastapi\_app/hash\_password.py**

from passlib.context import CryptContext

import sqlite3

pwd\_context = CryptContext(schemes=["bcrypt"], deprecated="auto")

conn = sqlite3.connect("hospital\_equipment\_system.db")

cursor = conn.cursor()

cursor.execute("SELECT username, password FROM personnel")

rows = cursor.fetchall()

for username, password in rows:

    hashed = pwd\_context.hash(password)

    cursor.execute("UPDATE personnel SET password = ? WHERE username = ?", (hashed, username))

conn.commit()

conn.close()

print("All passwords hashed with bcrypt.")

**fastapi\_app/main.py**

#main.py

from fastapi import FastAPI

from fastapi\_app.auth import router as auth\_router

from fastapi\_app.equipments import router as equipment\_router

from fastapi\_app.maintenance import router as maintenance\_router

from fastapi\_app.predict import router as predict\_router

from fastapi\_app.users import router as user\_router

from fastapi\_app.calendar import router as calendar\_router

app = FastAPI(title="Hospital Equipment Maintenance API")

# Register routers

app.include\_router(auth\_router, tags=["Auth"])

app.include\_router(equipment\_router, prefix="/equipments", tags=["Equipments"])

app.include\_router(maintenance\_router, prefix="/maintenance-log", tags=["Maintenance Logs"])

app.include\_router(predict\_router, prefix="/predict", tags=["Prediction"])

app.include\_router(user\_router, prefix="/users", tags=["Users"])

app.include\_router(calendar\_router, prefix="/calendar", tags=["Calendar"])

**fastapi\_app/models.py**

#models.py

from pydantic import BaseModel

from typing import Optional

class Equipment(BaseModel):

    equipment\_id: str

    type: str

    manufacturer: str

    location: str

    criticality: str

    installation\_date: str

class MaintenanceLog(BaseModel):

    maintenance\_id: str

    equipment\_id: str

    date: str

    maintenance\_type: str

    downtime\_hours: float

    cost\_inr: float

    issue\_description: str

    parts\_replaced: str

    vendor: str

    technician\_id: str

    service\_rating: int

    response\_time\_hours: float

    completion\_status: str

    warranty\_covered: str

    next\_service\_due: str

    status: str

class UserCreate(BaseModel):

    personnel\_id: str

    name: str

    role: str

    department: str

    experience\_years: float

    username: str

    password: str

**fastapi\_app/predict.py**

#predict.py

from fastapi import APIRouter, Depends

from fastapi\_app.dependencies import get\_current\_user

import numpy as np

import pandas as pd

import sqlite3

import joblib

from tensorflow.keras.models import load\_model

import os

router = APIRouter()

BASE\_DIR = os.path.dirname(os.path.abspath(\_\_file\_\_))

lstm\_model = load\_model(os.path.join(BASE\_DIR, "..", "saved\_models", "lstm\_model.h5"))

lgbm\_model = joblib.load(os.path.join(BASE\_DIR, "..", "saved\_models", "lgbm\_model.pkl"))

scaler = joblib.load(os.path.join(BASE\_DIR, "..", "saved\_models", "scaler.pkl"))

@router.post("/", summary="Predict maintenance for all equipment")

def predict\_maintenance(user=Depends(get\_current\_user)):

    conn = sqlite3.connect("hospital\_equipment\_system.db")

    query = """

    SELECT equipment\_id, timestamp, usage\_hours, patients\_served, workload\_level, avg\_cpu\_temp, error\_count

    FROM usage\_logs

    ORDER BY equipment\_id, timestamp DESC

    """

    df = pd.read\_sql\_query(query, conn)

    conn.close()

    df["timestamp"] = pd.to\_datetime(df["timestamp"])

    df = df.sort\_values(["equipment\_id", "timestamp"], ascending=[True, False])

    features = ["usage\_hours", "patients\_served", "workload\_level", "avg\_cpu\_temp", "error\_count"]

    equipment\_ids = df["equipment\_id"].unique()

    sequences = []

    equipment\_map = []

    for eq\_id in equipment\_ids:

        eq\_data = df[df["equipment\_id"] == eq\_id]

        if len(eq\_data) >= 5:

            recent\_logs = eq\_data.head(5).sort\_values("timestamp")

            X\_scaled = scaler.transform(recent\_logs[features])

            sequences.append(X\_scaled)

            equipment\_map.append(eq\_id)

    if not sequences:

        return {"message": "Not enough data for any equipment."}

    X\_seq = np.array(sequences)

    X\_flat = X\_seq.reshape(X\_seq.shape[0], -1)

    lstm\_probs = lstm\_model.predict(X\_seq).flatten()

    lgbm\_probs = lgbm\_model.predict\_proba(X\_flat)[:, 1]

    ensemble\_probs = (lstm\_probs + lgbm\_probs) / 2

    ensemble\_preds = (ensemble\_probs > 0.4).astype(int)

    results = []

    for eid, pred, prob in zip(equipment\_map, ensemble\_preds, ensemble\_probs):

        results.append({

            "equipment\_id": eid,

            "maintenance\_needed": int(pred),

            "confidence\_score": round(float(prob), 4)

        })

    return {"predictions": results}

**fastapi\_app/users.py**

#users.py

from fastapi import APIRouter, Depends, HTTPException

from pydantic import BaseModel

import sqlite3

from fastapi\_app.dependencies import get\_current\_user, require\_role

router = APIRouter()

def get\_db():

    return sqlite3.connect("hospital\_equipment\_system.db")

# --- Pydantic Model for input ---

class UserIn(BaseModel):

    personnel\_id: str

    name: str

    role: str  # e.g., admin, biomedical, technician

    department: str

    experience\_years: int

    username: str

    password: str

# --- Show current logged-in user’s full profile ---

@router.get("/me")

def who\_am\_i(user=Depends(get\_current\_user)):

    conn = get\_db()

    cursor = conn.cursor()

    cursor.execute("""

        SELECT personnel\_id, name, role, department, experience\_years, username

        FROM personnel WHERE username = ?

    """, (user["username"],))

    result = cursor.fetchone()

    conn.close()

    if not result:

        raise HTTPException(status\_code=404, detail="User not found")

    keys = ["personnel\_id", "name", "role", "department", "experience\_years", "username"]

    return dict(zip(keys, result))

# --- List all users (admin only) ---

@router.get("/", dependencies=[Depends(require\_role("admin"))])

def list\_users():

    conn = get\_db()

    cursor = conn.cursor()

    cursor.execute("""

        SELECT personnel\_id, name, role, department, experience\_years FROM personnel

    """)

    users = cursor.fetchall()

    conn.close()

    return {"users": users}

# --- Add a new user (admin only) ---

@router.post("/", dependencies=[Depends(require\_role("admin"))])

def add\_user(user: UserIn):

    conn = get\_db()

    cursor = conn.cursor()

    cursor.execute("""

        INSERT INTO personnel (personnel\_id, name, role, department, experience\_years, username, password)

        VALUES (?, ?, ?, ?, ?, ?, ?)

    """, (

        user.personnel\_id, user.name, user.role,

        user.department, user.experience\_years,

        user.username, user.password

    ))

    conn.commit()

    conn.close()

    return {"message": "User added"}

# --- Delete user by ID (admin only) ---

@router.delete("/{personnel\_id}", dependencies=[Depends(require\_role("admin"))])

def delete\_user(personnel\_id: str):

    conn = get\_db()

    cursor = conn.cursor()

    cursor.execute("DELETE FROM personnel WHERE personnel\_id = ?", (personnel\_id,))

    conn.commit()

    conn.close()

    return {"message": f"User {personnel\_id} deleted"}

**fastapi\_app/llm\_engine.py**

import requests

import base64

from fastapi import HTTPException

OLLAMA\_URL = "http://localhost:11434/api/generate"

MODEL\_NAME = "llava:7b"

def generate\_explanation\_ollama(equipment\_metrics: dict, role: str, image\_path: str = "") -> str:

    role = role.lower()

    tone = {

        "technician": "Provide a clear and actionable summary using non-technical language.",

        "biomedical": "Provide detailed analysis with technical insights, patterns, and preventive actions.",

        "admin": "Give a summarized overview with justification and potential resource implications."

    }.get(role, "Explain in general terms.")

    prompt = f"""

You are an AI assistant monitoring hospital equipment health.

Role: {role.capitalize()}

Tone: {tone}

Equipment Summary:

- Equipment ID: {equipment\_metrics['equipment\_id']}

- Age: {equipment\_metrics['equipment\_age']} years

- Downtime: {equipment\_metrics['downtime\_hours']} hours

- Failures: {equipment\_metrics['num\_failures']}

- Avg Response Time: {equipment\_metrics['response\_time\_hours']} hrs

- Predicted to Fail: {"Yes" if equipment\_metrics['predicted\_to\_fail'] else "No"}

- Maintenance Priorities:

    - Preventive: {equipment\_metrics['maintenance\_needs']['preventive']}

    - Corrective: {equipment\_metrics['maintenance\_needs']['corrective']}

    - Replacement: {equipment\_metrics['maintenance\_needs']['replacement']}

Explain the equipment's health based on these values in the selected tone.

Format your explanation as \*\*clear bullet points\*\* or \*\*step-by-step suggestions\*\* for easier understanding.

"""

    try:

        base64\_image = ""

        if image\_path:

            with open(image\_path, "rb") as img\_file:

                base64\_image = base64.b64encode(img\_file.read()).decode()

        response = requests.post(

            OLLAMA\_URL,

            json={

                "model": MODEL\_NAME,

                "prompt": prompt,

                "images": [base64\_image] if base64\_image else [],

                "stream": False

            },

            timeout=120

        )

        response.raise\_for\_status()

        return response.json().get("response", "No explanation returned.")

    except Exception as e:

        raise HTTPException(status\_code=500, detail=f"Ollama error: {str(e)}")

**fastapi\_app/maintenance.py**

from fastapi import APIRouter, HTTPException, Depends

from pydantic import BaseModel

from typing import Union

import sqlite3

import pandas as pd

import joblib

from datetime import datetime

from fastapi\_app.llm\_engine import generate\_explanation\_ollama

from fastapi\_app.dependencies import get\_current\_user, require\_role

from fastapi import File, UploadFile

import base64

from generate\_equipment\_report import fetch\_equipment\_metrics

router = APIRouter()

def get\_db():

    return sqlite3.connect("hospital\_equipment\_system.db")

# --- Base model for Technician ---

class MaintenanceBase(BaseModel):

    maintenance\_id: str

    equipment\_id: str

    date: str

    maintenance\_type: str

    downtime\_hours: float

    cost\_inr: float

    issue\_description: str

    parts\_replaced: str

    vendor: str

    technician\_id: str

    completion\_status: str

    warranty\_covered: str

# --- Extended model for Admin/Biomedical ---

class MaintenanceExtended(MaintenanceBase):

    service\_rating: int

    response\_time\_hours: float

    status: str

# --- View all logs (Technician sees only scheduled ones) ---

@router.get("/")

def view\_logs(user=Depends(get\_current\_user)):

    conn = get\_db()

    cursor = conn.cursor()

    query = "SELECT \* FROM maintenance\_logs"

    if user["role"] == "technician":

        query += " WHERE status = 'Scheduled'"

    cursor.execute(query)

    columns = [col[0] for col in cursor.description]

    rows = cursor.fetchall()

    conn.close()

    return {"logs": [dict(zip(columns, row)) for row in rows]}

# --- Add a maintenance log based on role ---

@router.post("/")

def add\_log(

    data: Union[MaintenanceExtended, MaintenanceBase],

    user=Depends(get\_current\_user)

):

    conn = get\_db()

    cursor = conn.cursor()

    if user["role"] == "technician":

        if not isinstance(data, MaintenanceBase) or isinstance(data, MaintenanceExtended):

            raise HTTPException(status\_code=403, detail="Technician not allowed to submit extended fields.")

        fields = list(MaintenanceBase.\_\_fields\_\_.keys())

    elif user["role"] in ["admin", "biomedical"]:

        if not isinstance(data, MaintenanceExtended):

            raise HTTPException(status\_code=400, detail="Admin must submit full log data.")

        fields = list(MaintenanceExtended.\_\_fields\_\_.keys())

    else:

        raise HTTPException(status\_code=403, detail="Unauthorized role.")

    values = [getattr(data, field) for field in fields]

    query = f"""

        INSERT INTO maintenance\_logs ({','.join(fields)})

        VALUES ({','.join(['?']\*len(fields))})

    """

    cursor.execute(query, values)

    conn.commit()

    conn.close()

    return {"message": "Log added"}

# --- Delete maintenance log (admin only) ---

@router.delete("/{maintenance\_id}", dependencies=[Depends(require\_role("admin"))])

def delete\_log(maintenance\_id: str):

    conn = get\_db()

    cursor = conn.cursor()

    cursor.execute("DELETE FROM maintenance\_logs WHERE maintenance\_id = ?", (maintenance\_id,))

    conn.commit()

    conn.close()

    return {"message": f"Maintenance log {maintenance\_id} deleted"}

# === Predict Maintenance Priority ===

@router.get("/priority/{equipment\_id}")

def get\_full\_maintenance\_priority(equipment\_id: str, user=Depends(get\_current\_user)):

    conn = get\_db()

    query = """

    SELECT e.equipment\_id, e.installation\_date,

           COALESCE(SUM(m.downtime\_hours), 0) AS downtime,

           COUNT(m.maintenance\_id) AS failures,

           COALESCE(AVG(m.response\_time\_hours), 0) AS avg\_response,

           COALESCE(f.needs\_maintenance\_10\_days, 0) AS needs\_maintenance\_10\_days

    FROM equipment e

    LEFT JOIN maintenance\_logs m ON e.equipment\_id = m.equipment\_id

    LEFT JOIN failure\_predictions f ON e.equipment\_id = f.equipment\_id

    WHERE e.equipment\_id = ?

    GROUP BY e.equipment\_id

    """

    df = pd.read\_sql\_query(query, conn, params=(equipment\_id,))

    conn.close()

    if df.empty:

        raise HTTPException(status\_code=404, detail="Equipment not found")

    df["installation\_date"] = pd.to\_datetime(df["installation\_date"])

    df["equipment\_age"] = (pd.Timestamp.today() - df["installation\_date"]).dt.days // 365

    features = df[[

        "equipment\_age",

        "downtime",

        "failures",

        "avg\_response",

        "needs\_maintenance\_10\_days"

    ]]

    features.columns = [

        "equipment\_age",

        "downtime\_hours",

        "num\_failures",

        "response\_time\_hours",

        "needs\_maintenance\_10\_days"

    ]

    # Load models

    scaler = joblib.load("saved\_models/multi\_priority\_scaler.pkl")

    X\_scaled = scaler.transform(features)

    def label(pred): return {0: "Low", 1: "Medium", 2: "High"}[pred]

    results = {}

    for mtype in ["preventive", "corrective", "replacement"]:

        model = joblib.load(f"saved\_models/{mtype}\_model.pkl")

        pred = model.predict(X\_scaled)[0]

        results[mtype] = label(pred)

    return {

        "equipment\_id": equipment\_id,

        "predicted\_to\_fail": bool(df["needs\_maintenance\_10\_days"].iloc[0]),

        "maintenance\_needs": results

    }

# fastapi\_app/maintenance.py - updated LLM route

@router.post("/maintenance-log/llm-explanation/{equipment\_id}")

def get\_llm\_explanation(

    equipment\_id: str,

    user=Depends(get\_current\_user)

):

    import base64

    import os

    role = user["role"].lower()

    # 1. Get all required data (includes trend chart generation)

    full\_metrics = fetch\_equipment\_metrics(equipment\_id)

    # 2. Use the chart generated with standard name

    image\_path = full\_metrics["chart\_path"]

    if not os.path.exists(image\_path):

        raise HTTPException(status\_code=404, detail="Trend chart not found")

    with open(image\_path, "rb") as img\_file:

        base64\_chart = base64.b64encode(img\_file.read()).decode()

    # 3. Generate LLM explanation

    explanation = generate\_explanation\_ollama(full\_metrics, role, image\_path)

    # 4. Return everything — merged into one response

    return {

        "equipment\_id": full\_metrics["equipment\_id"],

        "role": role,

        "equipment\_age": full\_metrics["equipment\_age"],

        "downtime\_hours": full\_metrics["downtime\_hours"],

        "num\_failures": full\_metrics["num\_failures"],

        "response\_time\_hours": full\_metrics["response\_time\_hours"],

        "predicted\_to\_fail": full\_metrics["predicted\_to\_fail"],

        "maintenance\_needs": full\_metrics["maintenance\_needs"],

        "explanation": explanation

    }