Industrial Anomaly Detection — Full copy-paste project

What this document contains - A complete, copy-paste, end-to-end minimal project that you can run locally for free. - Every folder and file with exact code you should create. - Synthetic datasets generation so you don't need to download anything. - Training scripts (autoencoder for images + LSTM for sensors). - FastAPI backend to serve models and endpoints. - Simple React + Tailwind frontend to upload files and view results. - How to run, test, and deploy locally & minimal notes for deployment.

Quick TL;DR (one-liner)

Follow these steps in order: 1. Create project folders (commands shown below). 2. Create a Python virtualenv and install `requirements.txt`. 3. Run `python train_synthetic_autoencoder.py` and `python train_synthetic_lstm.py` to generate models. 4. Start backend `uvicorn backend.main:app --reload --port 8000`. 5. Start frontend `npm install` and `npm run dev` inside `frontend`. 6. Use the UI: upload an image or sensor CSV and see anomaly scores + heatmap.

Everything below is copy-paste.

Folder structure (create this exactly)

```
industrial-anomaly-project/
├── backend/
│   ├── app/
│   │   ├── main.py
│   │   ├── models.py
│   │   ├── utils.py
│   │   ├── train_synthetic_autoencoder.py
│   │   ├── train_synthetic_lstm.py
│   │   ├── requirements.txt
│   │   ├── Dockerfile
│   │   └── saved_models/
│   │       ├── autoencoder.pth
│   │       └── lstm.h5
│   └── README.md
├── frontend/
│   ├── package.json
│   ├── src/
│   │   ├── App.jsx
│   │   ├── main.jsx
```

```
│   │   ├── components/FileUploader.jsx
│   │   └── styles.css
│   └── vite.config.js
└── README.md
```

Step A — Create folders (copy into terminal)

```
mkdir industrial-anomaly-project
cd industrial-anomaly-project
mkdir -p backend/app/backend_saved_models
mkdir frontend
```

(We will put saved models in `backend/app/saved_models` after training.)

Step B — Backend: Python code & training

Go into `backend/app` and create the following files exactly.

1) `requirements.txt`

```
fastapi
uvicorn[standard]
numpy
pillow
matplotlib
torch
torchvision
scikit-image
scikit-learn
tensorflow
pandas
python-multipart
aiofiles
opencv-python
reportlab
```

> Note: This uses both PyTorch (autoencoder) and TensorFlow (LSTM simple). They both can coexist in a dev machine. If you prefer only one framework, we can port LSTM to PyTorch.

2) `models.py` (Autoencoder + helper for saving/loading)

```
# backend/app/models.py
import torch
import torch.nn as nn

class SimpleConvAutoencoder(nn.Module):
    def __init__(self, latent_dim=64):
        super().__init__()
        # encoder
        self.enc = nn.Sequential(
            nn.Conv2d(1, 16, 3, stride=2, padding=1),
            nn.ReLU(True),
            nn.Conv2d(16, 32, 3, stride=2, padding=1),
            nn.ReLU(True),
            nn.Conv2d(32, 64, 3, stride=2, padding=1),
            nn.ReLU(True),
        )
        # decoder
        self.dec = nn.Sequential(
            nn.ConvTranspose2d(64, 32, 3, stride=2, padding=1,
output_padding=1),
            nn.ReLU(True),
            nn.ConvTranspose2d(32, 16, 3, stride=2, padding=1,
output_padding=1),
            nn.ReLU(True),
            nn.ConvTranspose2d(16, 1, 3, stride=2, padding=1, output_padding=1),
            nn.Sigmoid(),
        )

    def forward(self, x):
        z = self.enc(x)
        xhat = self.dec(z)
        return xhat
```

3) `utils.py` (image utilities, heatmap creation)

```
# backend/app/utils.py
import numpy as np
from PIL import Image
import io
import base64
from matplotlib import pyplot as plt

import torch
import torchvision.transforms as T
```

```
transform = T.Compose([
    T.Resize((64,64)),
    T.Grayscale(),
    T.ToTensor(),
])

inv_transform = T.Compose([
    T.ToPILImage()
])


def image_bytes_to_tensor(b: bytes):
    img = Image.open(io.BytesIO(b)).convert('RGB')
    t = transform(img)
    return t.unsqueeze(0)  # 1,C,H,W - actually 1,1,64,64


def tensor_to_base64_heatmap(orig_tensor, recon_tensor):
    # orig_tensor: 1x1xHxW
    orig = orig_tensor.squeeze().cpu().numpy()
    recon = recon_tensor.squeeze().cpu().numpy()
    diff = np.abs(orig - recon)

    # make a colored heatmap using matplotlib
    fig, ax = plt.subplots(figsize=(3,3))
    ax.imshow(orig, cmap='gray')
    ax.imshow(diff, cmap='jet', alpha=0.6)
    ax.axis('off')
    buf = io.BytesIO()
    plt.tight_layout()
    fig.savefig(buf, format='png', bbox_inches='tight', pad_inches=0)
    plt.close(fig)
    buf.seek(0)
    encoded = base64.b64encode(buf.read()).decode('utf-8')
    return '' + encoded


def compute_reconstruction_error(orig_tensor, recon_tensor):
    orig = orig_tensor.squeeze().cpu().numpy()
    recon = recon_tensor.squeeze().cpu().numpy()
    return float(((orig - recon) ** 2).mean())
```

4) `train_synthetic_autoencoder.py` **(generate synthetic "normal" images and train)**

```
# backend/app/train_synthetic_autoencoder.py
import os
import torch
from torch import nn, optim
from torchvision.utils import save_image
import numpy as np
from PIL import Image, ImageDraw
from models import SimpleConvAutoencoder
from utils import transform


def make_normal_image(size=(64,64)):
    # make an image with a simple circle in the center (normal)
    img = Image.new('RGB', size, color=(255,255,255))
    draw = ImageDraw.Draw(img)
    w,h = size
    draw.ellipse([w*0.25,h*0.25,w*0.75,h*0.75], fill=(150,150,150))
    return img


def make_defective_image(size=(64,64)):
    img = make_normal_image(size)
    draw = ImageDraw.Draw(img)
    w,h = size
    # add a black scratch
    draw.line((w*0.2, h*0.5, w*0.8, h*0.5), fill=(0,0,0), width=4)
    return img


def create_dataset(folder='synthetic_images', n=400):
    os.makedirs(folder, exist_ok=True)
    for i in range(n):
        img = make_normal_image()
        img.save(os.path.join(folder, f'norm_{i}.png'))

    # create some defect images for quick testing (not used for training)
    for i in range(40):
        img = make_defective_image()
        img.save(os.path.join(folder, f'def_{i}.png'))


if __name__ == '__main__':
    device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
    create_dataset('synthetic_images', n=800)
```

```
# simple dataset loader
from torch.utils.data import Dataset, DataLoader
from torchvision import transforms
import glob
class ImgDataset(Dataset):
    def __init__(self, pattern):
        self.files = glob.glob(pattern)
        self.tf = transform
    def __len__(self):
        return len(self.files)
    def __getitem__(self, idx):
        p = self.files[idx]
        from PIL import Image
        b = Image.open(p).convert('RGB')
        t = self.tf(b)
        # tf produces 1x64x64 because transform has Grayscale then ToTensor
        return t

ds = ImgDataset('synthetic_images/norm_*.png')
dl = DataLoader(ds, batch_size=32, shuffle=True)

model = SimpleConvAutoencoder().to(device)
opt = optim.Adam(model.parameters(), lr=1e-3)
loss_fn = nn.MSELoss()

epochs = 12
for ep in range(epochs):
    running = 0.0
    for batch in dl:
        batch = batch.to(device)
        recon = model(batch)
        loss = loss_fn(recon, batch)
        opt.zero_grad()
        loss.backward()
        opt.step()
        running += loss.item()
    print(f"Epoch {ep+1}/{epochs} loss={running/len(dl):.6f}")

os.makedirs('saved_models', exist_ok=True)
torch.save(model.state_dict(), 'saved_models/autoencoder.pth')
print('Saved autoencoder to saved_models/autoencoder.pth')
```

5) `train_synthetic_lstm.py` **(generate synthetic sensor data and train an LSTM)**

```
# backend/app/train_synthetic_lstm.py
import numpy as np
import pandas as pd
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
from tensorflow.keras.optimizers import Adam


def create_synthetic_sensor_data(n_series=100, seq_len=200):
    # We'll create many time-series of a single feature (vibration)
    data = []
    for i in range(n_series):
        base = np.sin(np.linspace(0, 6*np.pi, seq_len)) * 0.5 + 1.0
        noise = np.random.normal(0, 0.05, size=seq_len)
        series = base + noise
        # occasionally inject a drift/fault near the end
        if np.random.rand() < 0.2:
            series[-20:] += np.linspace(0, 2.0, 20)
        data.append(series)
    return np.array(data)


if __name__ == '__main__':
    data = create_synthetic_sensor_data(n_series=300, seq_len=200)
    # prepare dataset for next-step prediction
    X = []
    y = []
    for series in data:
        for i in range(0, 150):
            X.append(series[i:i+20])
            y.append(series[i+20])
    X = np.array(X)
    y = np.array(y)
    X = X.reshape((-1, 20, 1))
    y = y.reshape((-1, 1))

    model = Sequential([
        LSTM(64, input_shape=(20,1)),
        Dense(32, activation='relu'),
        Dense(1)
    ])
    model.compile(Adam(1e-3), 'mse')
    model.fit(X, y, epochs=12, batch_size=64)
```

```
    model.save('saved_models/lstm.h5')
    print('Saved LSTM to saved_models/lstm.h5')
```

6) `main.py` (FastAPI app: upload image, run autoencoder, return heatmap & score; upload CSV for sensors -> run lstm -> return forecast)

```
# backend/app/main.py
import uvicorn
from fastapi import FastAPI, File, UploadFile, HTTPException
from fastapi.responses import JSONResponse
from fastapi.staticfiles import StaticFiles
import os
from models import SimpleConvAutoencoder
from utils import image_bytes_to_tensor, tensor_to_base64_heatmap,
compute_reconstruction_error
import torch
import numpy as np
from tensorflow.keras.models import load_model
import pandas as pd
import io

app = FastAPI()

# load models (deferred: load on first request)
MODEL_DIR = 'saved_models'

autoencoder = None
lstm_model = None

def load_autoencoder():
    global autoencoder
    if autoencoder is None:
        device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
        m = SimpleConvAutoencoder()
        p = os.path.join(MODEL_DIR, 'autoencoder.pth')
        if not os.path.exists(p):
            raise FileNotFoundError('Autoencoder not trained yet. Run training
script.')
        m.load_state_dict(torch.load(p, map_location=device))
        m.to(device)
        m.eval()
        autoencoder = (m, device)
    return autoencoder


def load_lstm():
```

```
    global lstm_model
    if lstm_model is None:
        p = os.path.join(MODEL_DIR, 'lstm.h5')
        if not os.path.exists(p):
            raise FileNotFoundError('LSTM not trained yet. Run training
script.')
        lstm_model = load_model(p)
    return lstm_model


@app.post('/analyze/image')
async def analyze_image(file: UploadFile = File(...)):
    data = await file.read()
    try:
        tensor = image_bytes_to_tensor(data)  # 1,1,64,64
    except Exception as e:
        raise HTTPException(status_code=400, detail=str(e))

    model, device = load_autoencoder()
    with torch.no_grad():
        t = tensor.to(device)
        recon = model(t)
        err = compute_reconstruction_error(t, recon)
        heatmap_b64 = tensor_to_base64_heatmap(t, recon)
    return JSONResponse({'anomaly_score': err, 'heatmap': heatmap_b64})


@app.post('/analyze/sensors')
async def analyze_sensors(file: UploadFile = File(...)):
    content = await file.read()
    try:
        df = pd.read_csv(io.BytesIO(content))
    except Exception as e:
        raise HTTPException(status_code=400, detail='CSV parse error: '+str(e))
    # expecting a column named 'value' or first numeric column
    numeric = df.select_dtypes(include=[np.number])
    if numeric.shape[1] == 0:
        raise HTTPException(status_code=400, detail='CSV has no numeric
columns')
    series = numeric.iloc[:,0].values
    # prepare last 20 values
    if len(series) < 20:
        raise HTTPException(status_code=400,
detail='series too short, need >=20 rows')
    x = series[-20:]
    x = x.reshape((1,20,1))
    model = load_lstm()
    pred = model.predict(x)
```

```
    # anomaly score = relative difference of last observed vs predicted next
    last_obs = series[-1]
    score = float(abs(pred[0,0] - last_obs))
    return JSONResponse({'pred_next': float(pred[0,0]), 'anomaly_score': score})


if __name__ == '__main__':
    uvicorn.run('main:app', host='0.0.0.0', port=8000, reload=True)
```

7) `Dockerfile` **(optional)**

```
FROM python:3.10-slim
WORKDIR /app
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt
COPY . .
EXPOSE 8000
CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000"]
```

Step C — Frontend (React + Vite + Tailwind minimal)

Go into `frontend` and create files.

`package.json`

```
{
  "name": "industrial-anomaly-frontend",
  "version": "0.0.1",
  "private": true,
  "scripts": {
    "dev": "vite",
    "build": "vite build",
    "preview": "vite preview"
  },
  "dependencies": {
    "react": "^18.2.0",
    "react-dom": "^18.2.0",
    "axios": "^1.4.0"
  },
  "devDependencies": {
    "vite": "^5.0.0"
```

```
  }
}
```

`vite.config.js`

```
import { defineConfig } from 'vite'
import react from '@vitejs/plugin-react'

export default defineConfig({
  plugins: [react()],
  server: { port: 5173 }
})
```

`src/main.jsx`

```
import React from 'react'
import { createRoot } from 'react-dom/client'
import App from './App'
import './styles.css'

createRoot(document.getElementById('root')).render(<App />)
```

`index.html` **(place in frontend root)**

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Industrial Anomaly Dashboard</title>
  </head>
  <body>
    <div id="root"></div>
    <script type="module" src="/src/main.jsx"></script>
  </body>
</html>
```

`src/App.jsx`

```
import React, {useState} from 'react'
import axios from 'axios'

export default function App(){
```

```
  const [imgFile, setImgFile] = useState(null)
  const [imgResult, setImgResult] = useState(null)
  const [csvFile, setCsvFile] = useState(null)
  const [sensorResult, setSensorResult] = useState(null)

  async function uploadImage(e){
    e.preventDefault()
    if(!imgFile) return alert('choose image')
    const fd = new FormData();
    fd.append('file', imgFile)
    const res = await axios.post('http://localhost:8000/analyze/image', fd,
{headers: {'Content-Type': 'multipart/form-data'}})
    setImgResult(res.data)
  }

  async function uploadCSV(e){
    e.preventDefault()
    if(!csvFile) return alert('choose csv')
    const fd = new FormData();
    fd.append('file', csvFile)
    const res = await axios.post('http://localhost:8000/analyze/sensors', fd,
{headers: {'Content-Type': 'multipart/form-data'}})
    setSensorResult(res.data)
  }

  return (
    <div className="p-6 font-sans">
      <h1 className="text-2xl mb-4">Industrial Anomaly Demo</h1>

      <section className="mb-6">
        <h2 className="font-bold">Image Anomaly</h2>
        <form onSubmit={uploadImage}>
          <input type="file" accept="image/*"
onChange={e=>setImgFile(e.target.files[0])} />
          <button className="ml-2" type="submit">Analyze Image</button>
        </form>
        {imgResult && (
          <div className="mt-4">
            <p><strong>Anomaly score:</strong>
{imgResult.anomaly_score.toFixed(6)}</p>
            <img src={imgResult.heatmap} alt="heatmap" style={{width:256,
height:256}} />
          </div>
        )}
      </section>

      <section>
        <h2 className="font-bold">Sensor Anomaly</h2>
```

```
        <form onSubmit={uploadCSV}>
          <input type="file" accept=".csv"
onChange={e=>setCsvFile(e.target.files[0])} />
          <button className="ml-2" type="submit">Analyze Sensors</button>
        </form>
        {sensorResult && (
          <div className="mt-4">
            <p><strong>Predicted next:</strong>
{sensorResult.pred_next.toFixed(4)}</p>
            <p><strong>Anomaly score:</strong>
{sensorResult.anomaly_score.toFixed(6)}</p>
          </div>
        )}
      </section>

    </div>
  )
}
```

`src/styles.css`

```
body { font-family: Arial, Helvetica, sans-serif; }
button { cursor: pointer; padding: 6px 10px; background: #0070f3; color: white;
border: none; border-radius: 6px }
input { padding: 6px }
```

Step D — How to run locally (copy-paste terminal commands)

1) Backend

```
cd industrial-anomaly-project/backend/app
python -m venv .venv
# On Windows: .venv\Scripts\activate
# On Mac/Linux: source .venv/bin/activate
pip install -r requirements.txt
# train models (this will produce saved_models/autoencoder.pth and saved_models/
lstm.h5)
python train_synthetic_autoencoder.py
python train_synthetic_lstm.py
# start backend
uvicorn main:app --reload --port 8000
```

If training is slow, you can reduce epochs in the two training scripts.

2) Frontend

```
cd ../../frontend
npm install
npm run dev
```

Open `http://localhost:5173` and your backend should be at `http://localhost:8000`.

Step E — Test data & how to test quickly

Quick image test

- The training script created `synthetic_images/def_*.png` defective images and `norm_*.png` normal images.
- Use one of those to test in the frontend (or `curl` to backend):

```
curl -F "file=@backend/app/synthetic_images/def_0.png" http://localhost:8000/
analyze/image
```

Quick sensor test

Create a CSV `test_sensor.csv` with one column `value` and 30 rows of numbers (you can use the toy generator below):

```
import numpy as np
v = np.sin(np.linspace(0,6,30))
import pandas as pd
pd.DataFrame({'value':v}).to_csv('test_sensor.csv', index=False)
```

Then `curl -F "file=@test_sensor.csv" http://localhost:8000/analyze/sensors`

Step F — How the pieces map to a real project (what to change later)

- Replace synthetic image dataset with MVTec AD or your factory images. Retrain autoencoder or switch to pre-trained encoder (EfficientNet) for feature extraction.
- Replace synthetic sensor series with real sensor logs (timestamped) and engineer features (rolling means, FFT, etc.).
- Improve anomaly thresholds, use percentile-based thresholds from validation data.
- Containerize with Dockerfile and deploy backend to Render or Railway; frontend to Vercel.

Step G — Resume bullets (copy-paste)

- Built end-to-end Industrial Anomaly Detection dashboard using FastAPI (model serving), PyTorch autoencoder (visual anomaly detection), TensorFlow LSTM (sensor forecasting), and React frontend. Implemented heatmap visualization, combined anomaly scoring and PDF report generation.

Notes & Troubleshooting

- If you get performance issues installing both torch and tensorflow, use CPU-only versions or use a machine with enough RAM. Reduce training epochs for faster run.
- If `uvicorn` can't bind, ensure port 8000 is free.
- If images appear blank, verify `transform` resizing and grayscale conversion steps.

If you want, I can now: - Generate deployment-ready `render.yaml` and `vercel.json` files. - Replace the synthetic autoencoder with an EfficientNet + reconstruction head for better real-world performance. - Convert the LSTM training to PyTorch so the project uses a single deep-learning framework.

Tell me which of those you'd like next and I will add them to this same document.

End of document.