**Project Title:**

**"AI-Driven Satellite Imagery Intelligence System (SAT-AI): Deep Learning for Crop Yield, Deforestation, Disaster Detection & Land Segmentation"**

**🧱 Phase-Wise Plan to Build the Project**

| **Phase** | **Description** | **Deliverables** |
| --- | --- | --- |
| 🔹 Phase 1 | **Project Planning & Dataset Setup** | Objective, dataset sources, tools, repo structure |
| 🔹 Phase 2 | **Data Collection & Preprocessing** | Fetch satellite images, preprocess them for training |
| 🔹 Phase 3 | **Model Development** | Train CNNs, UNet, SegNet, CNN-LSTM for each module |
| 🔹 Phase 4 | **Evaluation & Visualization** | Evaluate using metrics (IoU, Accuracy), visualize maps |
| 🔹 Phase 5 | **Web Dashboard (Optional)** | Display predictions (e.g., Streamlit or Flask) |
| 🔹 Phase 6 | **Documentation & GitHub** | README, final report, PPT, GitHub upload |

**🔍 Final Modules & Use Cases**

**✅ Module 1: Crop Yield Prediction**

* Input: NDVI time-series + weather data
* Model: CNN + LSTM
* Output: Predicted yield (kg/ha) map per region

**✅ Module 2: Deforestation Detection**

* Input: Time-separated satellite images
* Model: ResNet or custom CNN classifier
* Output: Binary forest loss map

**✅ Module 3: Disaster/Crisis Management**

* Input: Before-after images of disasters (flood/fire/earthquake)
* Model: UNet for segmentation or YOLO for detection
* Output: Damage area detection

**✅ Module 4: Temporal Change Detection**

* Input: Multi-temporal images of same region
* Model: 3D CNN or CNN-LSTM
* Output: Change masks (urban growth, water loss)

**✅ Module 5: Land Cover Segmentation**

* Input: Sentinel-2 / ISPRS Potsdam imagery
* Model: UNet and SegNet
* Output: Per-pixel segmentation (forest, water, building, road)

**🛠️ Tech Stack**

* **Language**: Python
* **DL Frameworks**: TensorFlow/Keras or PyTorch
* **Satellite Tools**: Google Earth Engine, Rasterio, OpenCV
* **Visualization**: Matplotlib, Seaborn, Folium (for maps)
* **Deployment**: Streamlit (optional dashboard)
* **Documentation**: Jupyter Notebooks + Final Report + PPT

**🧪 Dataset Sources**

| **Dataset** | **Use Case** | **Source** |
| --- | --- | --- |
| **Sentinel-2 / Landsat-8** | All tasks | Google Earth Engine |
| **Crop Yield Data (e.g., India/Rwanda)** | Yield Prediction | [Radiant Earth](https://radiant.earth), Kaggle |
| **ISPRS Potsdam / DeepGlobe** | Land Segmentation | ISPRS, Kaggle |
| **Forest Loss (Global Forest Watch)** | Deforestation | <https://www.globalforestwatch.org> |
| **Disaster Imagery (xBD, UNOSAT)** | Crisis Mgmt | [xView2](https://xview2.org/), [UNOSAT](https://unosat.org/) |

**Today’s Action Plan (Day 1: Setup Phase)**

Let’s begin with:

**🔹 Step 1: Set up Project Directory**

satellite-ai-analysis/

│

├── data/ # Datasets

│ ├── raw/ # Unprocessed satellite images

│ ├── processed/ # Preprocessed or labeled images

│ ├── external/ # Public datasets (e.g., ISPRS, Sentinel)

│

├── notebooks/ # Jupyter Notebooks for EDA, modeling, etc.

│ ├── 01\_data\_exploration.ipynb

│ ├── 02\_model\_training\_unet.ipynb

│ └── 03\_temporal\_analysis.ipynb

│

├── scripts/ # Python scripts

│ ├── data\_loader.py # Load & preprocess satellite images

│ ├── unet\_model.py # UNet architecture

│ ├── train.py # Training script

│ └── evaluate.py # Evaluation script

│

├── models/ # Trained model files (.h5 or .pth)

│ └── unet\_landcover.h5

│

├── reports/ # Project reports and images

│ ├── figures/ # Model output visualizations

│ └── SAT\_AI\_Report.pdf

│

├── dashboard/ # (Optional) Streamlit app

│ └── app.py

│

├── .gitignore # Ignore checkpoints, models, etc.

├── requirements.txt # All required pip packages

├── README.md # Project overview, setup, and usage

└── LICENSE # MIT or Apache 2.0

**🔹 Step 2: Install Key Libraries**

bash

CopyEdit

pip install tensorflow keras opencv-python matplotlib seaborn scikit-learn rasterio geopandas earthengine-api

**🔹 Step 3: Set up Google Earth Engine (GEE)**

* Sign up at: <https://earthengine.google.com>
* Install GEE Python API: pip install earthengine-api
* Authenticate: earthengine authenticate

**🔹 Step 4: Create GitHub Repo**

* Name: satellite-ai-analysis
* Add sections for:
  + 📁 Project Overview
  + 🛰️ Dataset Sources
  + 🧠 Models Used
  + 📊 Results & Visualizations
  + 🚀 Run Instructions

**🧾 Step: Create scripts/data\_loader.py**

**🔧 What This Script Will Do:**

* Load images from /data/raw/
* Read satellite image bands (e.g., RGB, NIR)
* Resize them to a fixed shape (e.g., 256x256)
* Normalize pixel values
* Convert to NumPy arrays or tensors
* Save preprocessed images to /data/processed/

**✅ scripts/data\_loader.py Template Code (Using rasterio + cv2)**

python

CopyEdit

import os

import numpy as np

import rasterio

import cv2

RAW\_DIR = "data/raw/"

PROCESSED\_DIR = "data/processed/"

IMAGE\_SIZE = (256, 256)

def normalize(img):

"""Normalize image to [0, 1] range"""

return (img - np.min(img)) / (np.max(img) - np.min(img) + 1e-6)

def load\_and\_preprocess\_image(file\_path):

"""Load a multi-band satellite image and preprocess"""

with rasterio.open(file\_path) as src:

image = src.read([1, 2, 3]) # Load RGB bands

image = np.transpose(image, (1, 2, 0)) # Convert to HWC

image = cv2.resize(image, IMAGE\_SIZE)

image = normalize(image)

return image

def process\_dataset():

files = [f for f in os.listdir(RAW\_DIR) if f.endswith(".tif")]

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

for file in files:

input\_path = os.path.join(RAW\_DIR, file)

output\_path = os.path.join(PROCESSED\_DIR, file.replace(".tif", ".npy"))

print(f"Processing {file}...")

image = load\_and\_preprocess\_image(input\_path)

np.save(output\_path, image)

print("✅ All images preprocessed and saved.")

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

process\_dataset()

**✅ Step-by-Step: Fetch Sentinel-2 Imagery from Google Earth Engine**

**🛰️ What You’ll Get**

* GeoTIFF .tif files
* Real Sentinel-2 images (RGB or NDVI)
* Custom region and time range (e.g., crop fields, forests, disaster zones)

**🔧 Step 1: Sign up & Setup Earth Engine**

**✅ 1.1. Sign up for Earth Engine (FREE):**

👉 https://signup.earthengine.google.com/

**✅ 1.2. Install Earth Engine Python API:**

bash

CopyEdit

pip install earthengine-api

**✅ 1.3. Authenticate:**

bash

CopyEdit

earthengine authenticate

A browser will open — sign in with your Google account and copy the token.

**📍 Step 2: Create Earth Engine Python Script**

Create a file:  
📁 scripts/earth\_engine\_export.py

Paste the following code:

import ee

# Initialize EE with your approved GCP project

ee.Initialize(project="majestic-poetry-465305-d6")

# Define your area of interest

region = ee.Geometry.Rectangle([77.4, 12.8, 77.8, 13.2])

# Use the updated Sentinel-2 Harmonized collection

image = (

ee.ImageCollection("COPERNICUS/S2\_SR\_HARMONIZED")

.filterDate("2022-01-01", "2022-01-15")

.filterBounds(region)

.sort("CLOUDY\_PIXEL\_PERCENTAGE")

.first()

.select(["B4", "B3", "B2"]) # Red, Green, Blue

)

# Start an export to Google Drive

task = ee.batch.Export.image.toDrive(

image=image,

description="Sentinel\_RGB\_Export",

folder="satellite\_ai",

fileNamePrefix="bangalore\_rgb",

region=region,

scale=10,

maxPixels=1e9,

fileFormat="GeoTIFF"

)

task.start()

print("✅ Export task started. Check your Google Drive shortly.")

**✅ Step 3: Run the Script**

python scripts/earth\_engine\_export.py

⚠️ This only *starts* the export — Earth Engine will process it in the background.

**📁 Step 4: Download from Google Drive**

Once processed:

* Go to your Google Drive → satellite\_ai folder
* Download the file: bangalore\_rgb.tif
* Move it into:

satellite-ai-analysis/data/raw/

Now you're ready to run your data\_loader.py on real .tif files!

**📌 Want a Different Location?**

Just replace the coordinates in:

region = ee.Geometry.Rectangle([min\_lon, min\_lat, max\_lon, max\_lat])

**✅ Preprocessing Pipeline Goals**

✔️ Load the satellite .tif image  
✔️ Normalize pixel values  
✔️ Convert to NumPy arrays  
✔️ Create fixed-size image **patches** (e.g., 128×128)  
✔️ Save patches for training (and masks, if available)

**📁 Recommended Directory Structure**

We'll use this:

AI-DRIVEN-SATELLITE-ANALYSIS/

│

├── data/

│ ├── raw/ ← Original .tif

│ ├── processed/ ← Normalized full image

│ └── patches/ ← Training patches

│ ├── images/

│ └── masks/ ← (optional for segmentation)

│

├── scripts/

│ └── preprocess.py ← Full preprocessing pipeline

**🐍 scripts/preprocess.py — Complete Preprocessing Pipeline**

import os

import numpy as np

import rasterio

from rasterio.windows import Window

from pathlib import Path

from tqdm import tqdm

# === CONFIG ===

RGB\_PATH = "data/raw/bangalore\_rgb.tif"

MASK\_PATH = "data/raw/bangalore\_mask.tif"

PATCH\_SIZE = 128

STRIDE = 128

IMAGE\_SAVE\_DIR = "data/patches/images"

MASK\_SAVE\_DIR = "data/patches/masks"

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

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

# === LOAD TIF IMAGES ===

def load\_image(path, is\_mask=False):

with rasterio.open(path) as src:

data = src.read()

return data[0] if is\_mask else data.transpose(1, 2, 0)

# === NORMALIZE RGB TO [0, 1] ===

def normalize(image):

return image / 10000.0

# === RESIZE MASK TO MATCH RGB IF SHAPE DIFFERS ===

def resize\_if\_needed(mask, target\_shape):

if mask.shape != target\_shape[:2]:

print("⚠️ Mask shape doesn't match RGB. Resizing...")

from skimage.transform import resize

return resize(mask, target\_shape[:2], order=0, preserve\_range=True).astype(mask.dtype)

return mask

# === EXTRACT PATCHES ===

def extract\_image\_mask\_patches(image, mask, patch\_size, stride):

H, W = image.shape[:2]

patch\_id = 0

for i in tqdm(range(0, H - patch\_size + 1, stride), desc="Extracting patches"):

for j in range(0, W - patch\_size + 1, stride):

img\_patch = image[i:i+patch\_size, j:j+patch\_size, :]

mask\_patch = mask[i:i+patch\_size, j:j+patch\_size]

np.save(f"{IMAGE\_SAVE\_DIR}/patch\_{patch\_id:04d}.npy", img\_patch)

np.save(f"{MASK\_SAVE\_DIR}/patch\_{patch\_id:04d}.npy", mask\_patch)

patch\_id += 1

print(f"✅ Extracted {patch\_id} (image, mask) patch pairs.")

# === MAIN PIPELINE ===

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

image = load\_image(RGB\_PATH, is\_mask=False)

mask = load\_image(MASK\_PATH, is\_mask=True)

mask = resize\_if\_needed(mask, image.shape)

image = normalize(image)

extract\_image\_mask\_patches(image, mask, PATCH\_SIZE, STRIDE)

**🔧 Optional: Add Mask Preprocessing (for Segmentation Tasks)**

If you have masks (e.g., land cover, water, forest), you can:

1. Load the mask .tif (as single-channel)
2. Resize/crop to same shape as RGB
3. Save patches into data/patches/masks/

I'll help you with that if you're doing segmentation like UNet.

**✅ After This Pipeline**

You'll have:

* .npy image patches saved as patch\_0001.npy, etc.
* All ready for CNN or UNet training

**✅ Step-by-Step: Create Mask from NDVI using Google Earth Engine**

**📌 What we’ll do:**

1. Export Sentinel-2 bands: **B4 (Red)** and **B8 (NIR)**
2. Compute **NDVI**
3. Threshold NDVI to make a **vegetation mask**
4. Export both NDVI and mask as .tif to **Google Drive**

**✅ Step 1: Open Google Earth Engine Code Editor**

Go to 👉 <https://code.earthengine.google.com>

**✅ Step 2: Copy-Paste the Script Below**

var region = ee.Geometry.Rectangle([77.4, 12.8, 77.8, 13.2]);

var start = '2023-12-01';

var end = '2024-03-01';

var s2 = ee.ImageCollection('COPERNICUS/S2\_SR')

.filterBounds(region)

.filterDate(start, end)

.filter(ee.Filter.lt('CLOUDY\_PIXEL\_PERCENTAGE', 10))

.select(['B4', 'B8']) // ✅ Select only required bands

.median(); // ✅ Median now works

var red = s2.select('B4');

var nir = s2.select('B8');

var ndvi = nir.subtract(red).divide(nir.add(red)).rename('NDVI');

var mask = ndvi.gt(0.3).rename('mask');

Map.centerObject(region, 10);

Map.addLayer(ndvi, {min: 0, max: 1, palette: ['white', 'green']}, 'NDVI');

Map.addLayer(mask, {min: 0, max: 1, palette: ['black', 'lime']}, 'Vegetation Mask');

Export.image.toDrive({

image: ndvi.clip(region),

description: 'bangalore\_ndvi',

fileNamePrefix: 'bangalore\_ndvi',

region: region,

scale: 10,

crs: 'EPSG:4326',

maxPixels: 1e13

});

Export.image.toDrive({

image: mask.clip(region),

description: 'bangalore\_mask',

fileNamePrefix: 'bangalore\_mask',

region: region,

scale: 10,

crs: 'EPSG:4326',

maxPixels: 1e13

});

**✅ Step 3: Run the Script**

Click the **"Run"** button ▶️ in the top bar.

Earth Engine will ask you to **authorize** Google Drive access the first time. Allow it.

**✅ Step 4: Wait for Exports**

1. Go to the **Tasks** tab (right panel).
2. Click **Run** for both tasks: bangalore\_ndvi and bangalore\_mask.
3. Wait 3–5 mins.
4. Files will be saved to **Google Drive → root** as:
   * bangalore\_ndvi.tif
   * bangalore\_mask.tif

**✅ Step 5: Download .tif files to your data/raw/ folder**

Save them into:

bash

CopyEdit

PROJECT\_ROOT/data/raw/bangalore\_mask.tif

PROJECT\_ROOT/data/raw/bangalore\_ndvi.tif

**✅ Final Step: Load in Python**

Now in your preprocessing script:

python

CopyEdit

import rasterio

import matplotlib.pyplot as plt

mask\_path = r"your\_path\data\raw\bangalore\_mask.tif"

with rasterio.open(mask\_path) as src:

mask = src.read(1)

plt.imshow(mask, cmap='gray')

plt.title("Vegetation Mask from NDVI")

plt.axis('off')

plt.show()

Perfect! 🎉 Now that you've downloaded:

* ✅ bangalore\_ndvi.tif
* ✅ bangalore\_mask.tif

Let’s **integrate them into your project** and move toward **model training**.

**✅ Step 1: Move Files to data/raw/**

Ensure these files are placed inside your project:

AI-DRIVEN SATELLITE ANALYSIS/

└── data/

└── raw/

├── bangalore\_ndvi.tif

└── bangalore\_mask.tif

**✅ Step 2: Visualize & Validate in Python**

Create a new notebook (or use scripts/visualize\_ndvi.py) and run:

import rasterio

import matplotlib.pyplot as plt

ndvi\_path = r"data/raw/bangalore\_ndvi.tif"

mask\_path = r"data/raw/bangalore\_mask.tif"

# Load NDVI

with rasterio.open(ndvi\_path) as src:

ndvi = src.read(1)

# Load Mask

with rasterio.open(mask\_path) as src:

mask = src.read(1)

# Show side by side

plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)

plt.imshow(ndvi, cmap='YlGn')

plt.title("NDVI")

plt.colorbar()

plt.axis('off')

plt.subplot(1, 2, 2)

plt.imshow(mask, cmap='gray')

plt.title("Vegetation Mask")

plt.colorbar()

plt.axis('off')

plt.tight\_layout()

plt.show()

**✅ Step 3: Extract Patches (for UNet Training)**

Perfect! 🚀 Let’s now **extract patches** from the full NDVI image and vegetation mask so we can train the UNet model.

**🎯 Objective**

We'll extract **128×128 patches** from:

* ✅ bangalore\_ndvi.tif → as **input images**
* ✅ bangalore\_mask.tif → as **segmentation masks**

**📁 Directory Structure for Patches**

We'll save them here:

javascript

CopyEdit

data/

└── patches/

├── images/ → input NDVI patches (as `.npy` or `.png`)

└── masks/ → mask patches

Let’s build the script.

**✅ Python Script: Patch Extraction (scripts/extract\_patches.py)**

python

CopyEdit

import os

import numpy as np

import rasterio

from patchify import patchify

from tqdm import tqdm

# 🔁 Paths (update to your full path)

ndvi\_path = r"C:\Users\khuza\OneDrive\Desktop\Data science course\DATA SCIENCE COURSE\PROJECTS\AI-DRIVEN SATELLITE ANALYSIS\data\raw\bangalore\_ndvi.tif"

mask\_path = r"C:\Users\khuza\OneDrive\Desktop\Data science course\DATA SCIENCE COURSE\PROJECTS\AI-DRIVEN SATELLITE ANALYSIS\data\raw\bangalore\_mask.tif"

patch\_size = 128

stride = 128

# 📁 Output directories

image\_dir = r"C:\Users\khuza\OneDrive\Desktop\Data science course\DATA SCIENCE COURSE\PROJECTS\AI-DRIVEN SATELLITE ANALYSIS\data\patches\images"

mask\_dir = r"C:\Users\khuza\OneDrive\Desktop\Data science course\DATA SCIENCE COURSE\PROJECTS\AI-DRIVEN SATELLITE ANALYSIS\data\patches\masks"

os.makedirs(image\_dir, exist\_ok=True)

os.makedirs(mask\_dir, exist\_ok=True)

# 🧾 Read NDVI and Mask

with rasterio.open(ndvi\_path) as src:

ndvi = src.read(1)

with rasterio.open(mask\_path) as src:

mask = src.read(1)

# 🧩 Patchify both arrays

ndvi\_patches = patchify(ndvi, (patch\_size, patch\_size), step=stride)

mask\_patches = patchify(mask, (patch\_size, patch\_size), step=stride)

# 📝 Save patches

count = 0

for i in tqdm(range(ndvi\_patches.shape[0])):

for j in range(ndvi\_patches.shape[1]):

image\_patch = ndvi\_patches[i, j, :, :]

mask\_patch = mask\_patches[i, j, :, :]

# Save as .npy (or change to .png if needed)

np.save(os.path.join(image\_dir, f"img\_{count}.npy"), image\_patch)

np.save(os.path.join(mask\_dir, f"mask\_{count}.npy"), mask\_patch)

count += 1

print(f"✅ Extracted and saved {count} image-mask patches.")

**✅ Run the Script**

From your project root (where scripts/ is), run:

bash

CopyEdit

python scripts/extract\_patches.py

Or open it in Jupyter and run each cell.

Great! 🎉 Now that your NDVI and mask patches are extracted and stored in:

bash

CopyEdit

data/patches/images/ → NDVI input patches

data/patches/masks/ → binary vegetation mask patches

Let’s move to the **model training phase** using **UNet**!

**🧠 Step-by-Step Plan for Training UNet**

**✅ 1. Dataset Loader**

We’ll load .npy patches and convert them into PyTorch tensors.

**✅ 2. UNet Model Definition**

Build a standard UNet architecture (or import a ready one).

**✅ 3. Training Loop**

Train the model to predict vegetation masks from NDVI inputs.

**🛠️ Step 1: Project Setup**

Create a file at:

bash

CopyEdit

scripts/train\_unet.py

Paste this complete working setup:

**✅ scripts/train\_unet.py (Starter Script)**

import os

import numpy as np

import torch

import torch.nn as nn

from torch.utils.data import Dataset, DataLoader

from torchvision import transforms

from tqdm import tqdm

import matplotlib.pyplot as plt

# ✅ Define dataset

class PatchDataset(Dataset):

def \_\_init\_\_(self, image\_dir, mask\_dir, transform=None):

self.image\_paths = sorted([os.path.join(image\_dir, f) for f in os.listdir(image\_dir) if f.endswith('.npy')])

self.mask\_paths = sorted([os.path.join(mask\_dir, f) for f in os.listdir(mask\_dir) if f.endswith('.npy')])

self.transform = transform

def \_\_len\_\_(self):

return len(self.image\_paths)

def \_\_getitem\_\_(self, idx):

image = np.load(self.image\_paths[idx]) # shape: [H, W]

mask = np.load(self.mask\_paths[idx]) # shape: [H, W]

# Normalize NDVI from [0,1] to [-1,1]

image = (image - 0.5) \* 2.0

image = np.expand\_dims(image, axis=0) # [1, H, W]

mask = np.expand\_dims(mask, axis=0)

return torch.tensor(image, dtype=torch.float32), torch.tensor(mask, dtype=torch.float32)

# ✅ UNet model (simple version)

class UNet(nn.Module):

def \_\_init\_\_(self):

super(UNet, self).\_\_init\_\_()

def CBR(in\_c, out\_c):

return nn.Sequential(

nn.Conv2d(in\_c, out\_c, 3, padding=1),

nn.ReLU(inplace=True),

nn.Conv2d(out\_c, out\_c, 3, padding=1),

nn.ReLU(inplace=True)

)

self.enc1 = CBR(1, 64)

self.enc2 = CBR(64, 128)

self.enc3 = CBR(128, 256)

self.pool = nn.MaxPool2d(2)

self.dec2 = CBR(256 + 128, 128)

self.dec1 = CBR(128 + 64, 64)

self.up2 = nn.ConvTranspose2d(256, 128, 2, stride=2)

self.up1 = nn.ConvTranspose2d(128, 64, 2, stride=2)

self.final = nn.Conv2d(64, 1, 1)

def forward(self, x):

e1 = self.enc1(x)

e2 = self.enc2(self.pool(e1))

e3 = self.enc3(self.pool(e2))

d2 = self.up2(e3)

d2 = self.dec2(torch.cat([d2, e2], dim=1))

d1 = self.up1(d2)

d1 = self.dec1(torch.cat([d1, e1], dim=1))

out = self.final(d1)

return torch.sigmoid(out)

# ✅ Paths

image\_dir = r"C:\Users\khuza\OneDrive\Desktop\Data science course\DATA SCIENCE COURSE\PROJECTS\AI-DRIVEN SATELLITE ANALYSIS\data\patches\images"

mask\_dir = r"C:\Users\khuza\OneDrive\Desktop\Data science course\DATA SCIENCE COURSE\PROJECTS\AI-DRIVEN SATELLITE ANALYSIS\data\patches\masks"

# ✅ Hyperparameters

device = torch.device('cuda' if torch.cuda.is\_available() else 'cpu')

batch\_size = 16

epochs = 10

lr = 1e-3

# ✅ Dataloaders

dataset = PatchDataset(image\_dir, mask\_dir)

train\_loader = DataLoader(dataset, batch\_size=batch\_size, shuffle=True)

# ✅ Model, Loss, Optimizer

model = UNet().to(device)

criterion = nn.BCELoss()

optimizer = torch.optim.Adam(model.parameters(), lr=lr)

# ✅ Training loop

for epoch in range(epochs):

model.train()

epoch\_loss = 0

for images, masks in tqdm(train\_loader, desc=f"Epoch {epoch+1}/{epochs}"):

images, masks = images.to(device), masks.to(device)

preds = model(images)

loss = criterion(preds, masks)

optimizer.zero\_grad()

loss.backward()

optimizer.step()

epoch\_loss += loss.item()

print(f"✅ Epoch {epoch+1} Loss: {epoch\_loss / len(train\_loader):.4f}")

# ✅ Save model

torch.save(model.state\_dict(), "unet\_model.pth")

print("🎉 Model saved as unet\_model.pth")

**✅ Before You Run**

1. Make sure you have these installed:

pip install torch torchvision tqdm numpy matplotlib patchify

1. You can run this via terminal or Jupyter:

python scripts/train\_unet.py