# ДОДАТОК А

Код навчання моделі класифікації

```
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, transforms, models
from torch.utils.data import DataLoader
import os

# === Parameters ===
BATCH_SIZE = 16
EPOCHS = 15
IMG_SIZE = 224
NUM_CLASSES = 2

# === Directories ===
train_dir = 'dataset/train'
val_dir = 'dataset/val'
onnx_path = 'm1_model.onnx'
weights_path = 'm1_model_weights.pth'

# === Aurmentation and normalization ===
train_transform = transforms.Compose([
    transforms.Resize((IMG_SIZE, IMG_SIZE)),
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(15),
    transforms.ColorJitter(brightness=0.2, contrast=0.2,
saturation=0.2),
    transforms.RandomAffine(degrees=0, translate=(0.1, 0.1)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
std=[0.229, 0.224, 0.225])
])

val_transform = transforms.Compose([
```

```
    transforms.Resize((IMG_SIZE, IMG_SIZE)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
std=[0.229, 0.224, 0.225])
])

train_data = datasets.ImageFolder(train_dir,
transform=train_transform)
val_data = datasets.ImageFolder(val_dir,
transform=val_transform)

train_loader = DataLoader(train_data, batch_size=BATCH_SIZE,
shuffle=True)
val_loader = DataLoader(val_data, batch_size=BATCH_SIZE,
shuffle=False)

print("Класи:", train_data.classes)

# === Model ===
model =
models.resnet18(weights=models.ResNet18_Weights.IMAGENET1K_V1)
model.fc = nn.Sequential(
    nn.Dropout(0.5),  # Add Dropout
    nn.Linear(model.fc.in_features, NUM_CLASSES)
)
device = torch.device("cuda" if torch.cuda.is_available() else
"cpu")
model.to(device)

criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.0001,
weight_decay=1e-4)

# === Learning with Early Stopping ===
best_val_loss = float('inf')
patience = 3
```

```
counter = 0

for epoch in range(EPOCHS):
    model.train()
    running_loss = 0.0
    for images, labels in train_loader:
        images, labels = images.to(device), labels.to(device)
        optimizer.zero_grad()
        output = model(images)
        loss = criterion(output, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item()
    avg_train_loss = running_loss / len(train_loader)
    print(f"Epoch {epoch+1}/{EPOCHS}, Train Loss:
{avg_train_loss:.4f}")

    model.eval()
    val_loss = 0.0
    correct = 0
    total = 0
    with torch.no_grad():
        for images, labels in val_loader:
            images, labels = images.to(device),
labels.to(device)
            output = model(images)
            loss = criterion(output, labels)
            val_loss += loss.item()
            _, predicted = torch.max(output, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()
    avg_val_loss = val_loss / len(val_loader)
    val_accuracy = 100 * correct / total
    print(f"Val Loss: {avg_val_loss:.4f}, Val Accuracy:
{val_accuracy:.2f}%")
```

```
    # Early Stopping
    if avg_val_loss < best_val_loss:
        best_val_loss = avg_val_loss
        counter = 0
        torch.save(model.state_dict(), weights_path)
        print(f"Збережено кращі ваги в {weights_path}")
    else:
        counter += 1
        if counter >= patience:
            print("Early stopping triggered!")
            break

# === Export to ONNX ===
dummy_input = torch.randn(1, 3, IMG_SIZE, IMG_SIZE).to(device)
torch.onnx.export(
    model,
    dummy_input,
    onnx_path,
    input_names=['input'],
    output_names=['output'],
    dynamic_axes={'input': {0: 'batch_size'}, 'output': {0:
'batch_size'}},
    opset_version=11
)
print(f"Модель експортовано в {onnx_path}")
```

# ДОДАТОК Б

Код реалізації моделі класифікації

```
using Microsoft.ML.OnnxRuntime;
using Microsoft.ML.OnnxRuntime.Tensors;
using System;
using System.Collections.Generic;
using System.Drawing;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

public class OnnxModelRunner
{
    private readonly InferenceSession _session;
    private readonly int _inputWidth;
    private readonly int _inputHeight;

    public OnnxModelRunner(string modelPath)
    {
        _session = new InferenceSession(modelPath);

        var inputMeta = _session.InputMetadata["input"];
        _inputHeight = inputMeta.Dimensions[2];
        _inputWidth = inputMeta.Dimensions[3];
    }

    public string Predict(string imagePath)
    {
        var input = PreprocessImage(imagePath);
        var inputs = new List<NamedOnnxValue>
        {
            NamedOnnxValue.CreateFromTensor("input", input)
        };
```

```
        using var results = _session.Run(inputs);
        var output = 
results.First().AsEnumerable<float>().ToArray();

        int predictedIndex = Array.IndexOf(output, 
output.Max());
        return predictedIndex == 0 ? "⚠ Уражена ділянка" : 
"✅ Неуражена ділянка";
    }

    private DenseTensor<float> PreprocessImage(string 
imagePath)
    {
        using (var original = new Bitmap(imagePath))
        using (var resized = new Bitmap(original, new 
Size(_inputWidth, _inputHeight)))
        {
            var tensor = new DenseTensor<float>(new[] { 1, 3, 
_inputHeight, _inputWidth });

            for (int y = 0; y < _inputHeight; y++)
            {
                for (int x = 0; x < _inputWidth; x++)
                {
                    Color pixel = resized.GetPixel(x, y);
                    tensor[0, 0, y, x] = pixel.R / 255f;
                    tensor[0, 1, y, x] = pixel.G / 255f;
                    tensor[0, 2, y, x] = pixel.B / 255f;
                }
            }
            return tensor;
        }
    }
}
```

# ДОДАТОК В

Код навчання моделі сегментації

```
import os
import cv2
import torch
import torch.nn as nn
from torch.utils.data import Dataset, DataLoader
from torchvision import transforms
from tqdm import tqdm
import numpy as np

# ==== Parameters ====
IMG_SIZE = 128
BATCH_SIZE = 4
EPOCHS = 10
DEVICE = torch.device("cuda" if torch.cuda.is_available() else
"cpu")


# ==== Dataset class ====
class SegmentationDataset(Dataset):
    def __init__(self, image_dir, mask_dir, transform=None):
        self.image_dir = image_dir
        self.mask_dir = mask_dir
        self.images = os.listdir(image_dir)
        self.transform = transform

    def __len__(self):
        return len(self.images)

    def __getitem__(self, idx):
        img_path = os.path.join(self.image_dir,
self.images[idx])
        mask_path = os.path.join(self.mask_dir,
self.images[idx].replace(".jpg", "_mask.png"))
```

```
        image = cv2.imread(img_path)
        mask = cv2.imread(mask_path, cv2.IMREAD_GRAYSCALE)

        image = cv2.resize(image, (IMG_SIZE, IMG_SIZE))
        mask = cv2.resize(mask, (IMG_SIZE, IMG_SIZE))
        mask = np.expand_dims(mask, axis=0) / 255.0  # [1, H,
W]

        if self.transform:
            image = self.transform(image)
        else:
            image = torch.from_numpy(image.transpose(2, 0,
1)).float() / 255.0

        return image, torch.from_numpy(mask).float()


# ==== U-Net model====
class UNet(nn.Module):
    def __init__(self):
        super(UNet, self).__init__()

        def conv_block(in_channels, out_channels):
            return nn.Sequential(
                nn.Conv2d(in_channels, out_channels, 3,
padding=1),
                nn.ReLU(),
                nn.Conv2d(out_channels, out_channels, 3,
padding=1),
                nn.ReLU()
            )

        self.encoder1 = conv_block(3, 64)
        self.pool1 = nn.MaxPool2d(2)
        self.encoder2 = conv_block(64, 128)
```

```
        self.pool2 = nn.MaxPool2d(2)

        self.bottleneck = conv_block(128, 256)

        self.upconv2 = nn.ConvTranspose2d(256, 128, 2,
stride=2)
        self.decoder2 = conv_block(256, 128)
        self.upconv1 = nn.ConvTranspose2d(128, 64, 2,
stride=2)
        self.decoder1 = conv_block(128, 64)

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

    def forward(self, x):
        e1 = self.encoder1(x)
        e2 = self.encoder2(self.pool1(e1))
        b = self.bottleneck(self.pool2(e2))
        d2 = self.upconv2(b)
        d2 = torch.cat((d2, e2), dim=1)
        d2 = self.decoder2(d2)
        d1 = self.upconv1(d2)
        d1 = torch.cat((d1, e1), dim=1)
        d1 = self.decoder1(d1)
        return torch.sigmoid(self.final_conv(d1))


# ==== Learning ====
def train_model():
    dataset = SegmentationDataset("dataset/images",
"dataset/masks")
    dataloader = DataLoader(dataset, batch_size=BATCH_SIZE,
shuffle=True)

    model = UNet().to(DEVICE)
    criterion = nn.BCELoss()
    optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
```

```
    print("Навчання моделі U-Net...")
    for epoch in range(EPOCHS):
        model.train()
        epoch_loss = 0
        for images, masks in tqdm(dataloader):
            images, masks = images.to(DEVICE),
masks.to(DEVICE)
            outputs = model(images)
            loss = criterion(outputs, masks)
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
            epoch_loss += loss.item()
        print(f"Епоха {epoch + 1}, Втрата: {epoch_loss:.4f}")
    # ==== Експорт в ONNX ====
    dummy_input = torch.randn(1, 3, IMG_SIZE,
IMG_SIZE).to(DEVICE)
    torch.onnx.export(model, dummy_input, "unet_model.onnx",
                      input_names=["input"],
output_names=["output"],
                      dynamic_axes={"input": {0: "batch"},
"output": {0: "batch"}},
                      opset_version=11)
    print("Модель експортовано у файл: unet_model.onnx")


if __name__ == "__main__":
    train_model()
```

# ДОДАТОК Г

Код навчання моделі для генерації масок

```
import cv2
import os
import numpy as np
from tqdm import tqdm

images_path = "dataset/images"
masks_path = "dataset/masks"
os.makedirs(masks_path, exist_ok=True)

# === Images processing ===
for file in tqdm(os.listdir(images_path)):
    if not file.lower().endswith(".jpg"):
        continue
    image = cv2.imread(os.path.join(images_path, file))

    # --- Converting in grey + improvement ---
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    blurred = cv2.GaussianBlur(gray, (5, 5), 0)

    # --- Threshold segmentation ---
    _, thresh = cv2.threshold(blurred, 100, 255,
cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)

    # --- Morphology for purification ---
    kernel = np.ones((3, 3), np.uint8)
    cleaned = cv2.morphologyEx(thresh, cv2.MORPH_OPEN, kernel,
iterations=2)

    # --- Mask saving ---
    mask_filename = os.path.join(masks_path,
file.replace(".jpg", "_mask.png"))
    cv2.imwrite(mask_filename, cleaned)
NX завершено!")
```

# ДОДАТОК Д

Код реалізації моделі сегментації

```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Drawing.Imaging;
using System.Linq;
using Microsoft.ML.OnnxRuntime;
using Microsoft.ML.OnnxRuntime.Tensors;

public class UnetModelRunner : IDisposable
{
    private readonly InferenceSession session;
    private readonly int modelWidth;
    private readonly int modelHeight;
    private readonly int outputSize;

    public UnetModelRunner(string modelPath)
    {
        session = new InferenceSession(modelPath);

        var inputMeta = session.InputMetadata["input"];
        var dims = inputMeta.Dimensions;

        modelHeight = dims[2];
        modelWidth = dims[3];

        outputSize = 512;
    }


    public Bitmap Segment(string imagePath)
    {
        using Bitmap bitmap = new Bitmap(imagePath);
```

```
        using Bitmap resized = new Bitmap(bitmap, modelWidth,
modelHeight);

        var inputTensor = ImageToTensor(resized);

        var inputs = new List<NamedOnnxValue>
    {
        NamedOnnxValue.CreateFromTensor("input", inputTensor)
    };

        using
IDisposableReadOnlyCollection<DisposableNamedOnnxValue>
results = session.Run(inputs);

        var output = results.First().AsTensor<float>();

        int affectedPixelCount = 0;
        foreach (var val in output.ToArray())
        {
            if (val > 0.5f)
                affectedPixelCount++;
        }

        if (affectedPixelCount == 0)
        {
            throw new Exception("На зображенні не виявлено
уражених ділянок.");
        }

        Bitmap mask = TensorToMask(output, modelWidth,
modelHeight);
        Bitmap scaledMask = new Bitmap(mask, new
Size(outputSize, outputSize));
        Bitmap finalImage = AddLegendBelow(scaledMask,
            "Темно-сіре — неуражене",
            "Біле — уражене");
```

```
        return finalImage;
    }



    private static Tensor<float> ImageToTensor(Bitmap image)
    {
        int width = image.Width;
        int height = image.Height;

        var tensor = new DenseTensor<float>(new[] { 1, 3,
height, width });

        for (int y = 0; y < height; y++)
        {
            for (int x = 0; x < width; x++)
            {
                Color pixel = image.GetPixel(x, y);

                tensor[0, 0, y, x] = pixel.R / 255f;
                tensor[0, 1, y, x] = pixel.G / 255f;
                tensor[0, 2, y, x] = pixel.B / 255f;
            }
        }

        return tensor;
    }

    private static Bitmap TensorToMask(Tensor<float> output,
int width, int height)
    {
        Bitmap mask = new Bitmap(width, height,
PixelFormat.Format32bppArgb);

        Color affectedColor = Color.FromArgb(255, 64, 64, 64);
// Темно-сірий — уражене
```

```
        Color healthyColor = Color.White; // Біле — неуражене

        for (int y = 0; y < height; y++)
        {
            for (int x = 0; x < width; x++)
            {
                float value = output[0, 0, y, x];
                bool isAffected = value > 0.5f;

                mask.SetPixel(x, y, isAffected ? affectedColor
: healthyColor);
            }
        }

        return mask;
    }

    private static Bitmap AddLegendBelow(Bitmap image, params
string[] legends)
    {
        int padding = 10;
        int fontSize = 24;
        int lineHeight = fontSize + 8;

        int legendHeight = padding * 2 + lineHeight *
legends.Length;
        int width = image.Width;
        int height = image.Height + legendHeight;

        Bitmap result = new Bitmap(width, height);

        using (Graphics g = Graphics.FromImage(result))
        {
            g.Clear(Color.White);
```

```
            g.DrawImage(image, 0, 0, image.Width,
image.Height);

            using (Font font = new Font("Arial", fontSize,
FontStyle.Bold))
            using (Brush textBrush = new
SolidBrush(Color.Black))
            {
                float yText = image.Height + padding;
                foreach (var legend in legends)
                {
                    var textSize = g.MeasureString(legend,
font);
                    float xText = (width - textSize.Width) /
2;

                    g.DrawString(legend, font, textBrush,
xText, yText);
                    yText += lineHeight;
                }
            }
        }

        return result;
    }

    public void Dispose()
    {
        session.Dispose();
    }
}
```

# ДОДАТОК Е

Код навчання моделі прогнозування

```
import pandas as pd
import torch
import torch.nn as nn
import torch.onnx
from sklearn.preprocessing import LabelEncoder
import numpy as np

df = pd.read_csv(
    "dataset/damaged_metadata.csv",
    encoding="windows-1251",
    quotechar='"',
    sep=",",
    on_bad_lines='skip'
)

# Clean and fill missing
df.fillna("unknown", inplace=True)

# === Helper: convert time range like '5-10 years' to number
===
def parse_time_range(val):
    if isinstance(val, str) and '-' in val:
        numbers = [s for s in val.replace("years",
"").replace("months", "").split('-') if s.strip().isdigit()]
        if numbers:
            return sum(map(int, numbers)) / len(numbers)
    elif isinstance(val, str) and val.strip().isdigit():
        return int(val)
    return 0

df["recovery_time_1_val"] =
df["recovery_time_1"].apply(parse_time_range)
```

```
# === Encode categorical columns ===
categorical_cols = ["equipment_type", "recovery_time_1_desc"]
encoders = {}
for col in categorical_cols:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])
    encoders[col] = le

# === Features and target ===
features = ["equipment_type", "recovery_time_1_desc"]
X = df[features].values.astype(np.float32)
y = df["recovery_time_1_val"].values.astype(np.float32)

# === Create artificial sequences of shape (batch, seq,
feature) ===
X_seq = np.expand_dims(X, axis=1)
y = np.expand_dims(y, axis=1)

X_tensor = torch.tensor(X_seq)
y_tensor = torch.tensor(y)

# === Define LSTM+TFT model ===
class SimpleLSTMTransformer(nn.Module):
    def __init__(self, input_size, hidden_size,
output_size=1):
        super(SimpleLSTMTransformer, self).__init__()
        self.lstm = nn.LSTM(input_size, hidden_size,
batch_first=True)
        self.linear = nn.Linear(hidden_size, hidden_size)
        self.transformer = nn.TransformerEncoder(
            nn.TransformerEncoderLayer(d_model=hidden_size,
nhead=2, batch_first=True),
            num_layers=1
        )
        self.output = nn.Linear(hidden_size, output_size)
```

```
    def forward(self, x):
        x, _ = self.lstm(x)
        x = self.linear(x)
        x = self.transformer(x)
        return self.output(x[:, -1, :])

# === Instantiate and train ===
input_size = X_tensor.shape[2]
model = SimpleLSTMTransformer(input_size=input_size,
hidden_size=32)
criterion = nn.MSELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=1e-3)

# Simple training loop
model.train()
for epoch in range(150):
    optimizer.zero_grad()
    output = model(X_tensor)
    loss = criterion(output, y_tensor)
    loss.backward()
    optimizer.step()
    if epoch % 10 == 0:
        print(f"Epoch {epoch}, Loss: {loss.item():.4f}")

# === Export to ONNX ===
model.eval()
dummy_input = torch.randn(1, 1, input_size)
torch.onnx.export(
    model, dummy_input, "model_lstm_tft.onnx",
    input_names=["input"], output_names=["output"],
    dynamic_axes={"input": {0: "batch"}, "output": {0:
"batch"}},
    opset_version=14
)

print("Експорт до ONNX завершено!")
```

# ДОДАТОК Ж

Код реалізації моделі прогнозування

```
using Microsoft.ML.OnnxRuntime.Tensors;
using Microsoft.ML.OnnxRuntime;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace AgroAI.Services
{
    public class LstmOnnxRunner
    {
        private readonly InferenceSession _session;
        private readonly int _sequenceLength;
        private readonly int _featureDim;

        public LstmOnnxRunner(string modelPath, int
sequenceLength, int featureDim)
        {
            _session = new InferenceSession(modelPath);
            _sequenceLength = sequenceLength;
            _featureDim = featureDim;
        }

        public float[] Predict(float[,] inputSequence)
        {
            int batch = inputSequence.GetLength(0);
            int featureDim = inputSequence.GetLength(1);

            var tensor = new DenseTensor<float>(new[] { batch,
1, featureDim });

            for (int i = 0; i < batch; i++)
```

```
            {
                for (int j = 0; j < featureDim; j++)
                {
                    tensor[i, 0, j] = inputSequence[i, j];
                }
            }

            var inputs = new List<NamedOnnxValue>
            {
                NamedOnnxValue.CreateFromTensor("input",
tensor)
            };

            using var results = _session.Run(inputs);
            var outputTensor =
results.First().AsTensor<float>();

            return outputTensor.ToArray();
        }



    }
}
```

# ДОДАТОК И

Код функціональності інтерфейсу

```
using Microsoft.Win32;
using System;
using System.IO;
using System.Linq;
using System.Windows;
using System.Windows.Media.Imaging;
using System.Windows.Interop;
using System.Threading.Tasks;
using System.Windows.Controls;
using IOPath = System.IO.Path;
using AgroAI.Services;

namespace AgroAI
{
    public partial class MainWindow : Window
    {
        private string imagePath;
        private string currentPrediction = "";
        private string currentForecast = "";
        private BitmapSource currentMask;

        public MainWindow()
        {
            InitializeComponent();
        }

        private async void LoadImage_Click(object sender,
RoutedEventArgs e)
        {
            OpenFileDialog dialog = new OpenFileDialog();
            dialog.Filter = "Image files
(*.png;*.jpg)|*.png;*.jpg";
```

```
            if (dialog.ShowDialog() == true)
            {
                imagePath = dialog.FileName;
                OriginalImage.Source = new BitmapImage(new
Uri(imagePath));
                MaskImage.Source = null;
                ForecastTextBlock.Text = "";
                ResultTextBlock.Text = "";
                SegmentButton.Visibility =
Visibility.Collapsed;
                PredictButton.Visibility =
Visibility.Collapsed;
                currentPrediction = "";
                currentForecast = "";
                currentMask = null;
            }
        }

        private async void Classify_Click(object sender,
RoutedEventArgs e)
        {
            if (string.IsNullOrEmpty(imagePath))
            {
                ResultTextBlock.Text = "❗ Спочатку завантажте
зображення!";
                return;
            }

            LoadingOverlay.Visibility = Visibility.Visible;
            await Task.Delay(300);

            try
            {
                var classifier = new
OnnxModelRunner("Models/m1_model.onnx");
```

```
                var result = await Task.Run(() =>
classifier.Predict(imagePath));
                currentPrediction = result;

                ResultTextBlock.Text = result;

                if (result.Contains("Уражена"))
                {
                    SegmentButton.Visibility =
Visibility.Visible;
                    PredictButton.Visibility =
Visibility.Visible;
                }
                else
                {
                    SegmentButton.Visibility =
Visibility.Collapsed;
                    PredictButton.Visibility =
Visibility.Collapsed;
                    MaskImage.Source = null;
                    ForecastTextBlock.Text = "";
                }
            }
            catch (Exception ex)
            {
                ResultTextBlock.Text = $"❗ Помилка
класифікації: {ex.Message}";
            }
            finally
            {
                LoadingOverlay.Visibility =
Visibility.Collapsed;
            }
        }
```

```
        private async void Segment_Click(object sender,
RoutedEventArgs e)
        {
            if (string.IsNullOrEmpty(imagePath))
            {
                ResultTextBlock.Text = "❗ Спочатку завантажте
зображення!";
                return;
            }

            LoadingOverlay.Visibility = Visibility.Visible;
            await Task.Delay(300);

            try
            {
                var segmenter = new
UnetModelRunner("Models/unet_model.onnx");
                var mask = await Task.Run(() =>
segmenter.Segment(imagePath));

                var outputPath =
IOPath.Combine(IOPath.GetDirectoryName(imagePath),
"segmented_mask.png");
                mask.Save(outputPath);

                currentMask = BitmapFromBitmap(mask);
                MaskImage.Source = currentMask;
            }
            catch (Exception ex)
            {
                ResultTextBlock.Text = $"❗ Помилка сегментації:
{ex.Message}";
            }
            finally
            {
```

```
                LoadingOverlay.Visibility =
Visibility.Collapsed;
            }
        }

        private async void PredictLstm_Click(object sender,
RoutedEventArgs e)
        {
            LoadingOverlay.Visibility = Visibility.Visible;
            await Task.Delay(300);

            try
            {
                float[,] inputSequence =
LoadYourTimeSeriesData();
                var lstmRunner = new
LstmOnnxRunner("Models/model_lstm_tft.onnx", 50, 2);
                var prediction = await Task.Run(() =>
lstmRunner.Predict(inputSequence));

                float totalArea = prediction.Sum();
                string weapon = totalArea > 50 ? "2S1 з ZY-23"
:
                                totalArea > 20 ? "Grad" :
"Стрілецька зброя";
                float recoveryTime = totalArea * 0.5f;

                currentForecast =
                    $"Загальна площа ураження: {totalArea:F2}
м²\n" +
                    $"Можлива зброя: {weapon}\n" +
                    $"Орієнтовний час відновлення:
{recoveryTime:F1} місяців";

                ForecastTextBlock.Text = currentForecast;
            }
```

```
            catch (Exception ex)
            {
                ForecastTextBlock.Text = $"❗ Помилка прогнозу: {ex.Message}";
            }
            finally
            {
                LoadingOverlay.Visibility = Visibility.Collapsed;
            }
        }

        private void SaveReport_Click(object sender, RoutedEventArgs e)
        {
            if (string.IsNullOrEmpty(currentPrediction) || string.IsNullOrEmpty(currentForecast))
            {
                MessageBox.Show("Спочатку виконайте класифікацію та прогноз.");
                return;
            }

            string selectedFormat = (ReportFormatCombo.SelectedItem as ComboBoxItem)?.Content.ToString();

            SaveFileDialog dialog = new SaveFileDialog();
            if (selectedFormat == "PDF")
                dialog.Filter = "PDF файл (*.pdf)|*.pdf";
            else
                dialog.Filter = "Word документ (*.docx)|*.docx";

            if (dialog.ShowDialog() == true)
```

```
            {
                try
                {
                    if (selectedFormat == "PDF")

ReportGenerator.GeneratePdf(dialog.FileName,
currentPrediction, currentForecast, currentMask);
                    else

ReportGenerator.GenerateDocx(dialog.FileName,
currentPrediction, currentForecast, currentMask);

                    MessageBox.Show("Звіт успішно
збережено.");
                }
                catch (Exception ex)
                {
                    MessageBox.Show($"❗ Помилка збереження
звіту:\n{ex.Message}");
                }
            }
        }

        private float[,] LoadYourTimeSeriesData()
        {
            int batch = 50;
            int featureDim = 2;
            float[,] data = new float[batch, featureDim];
            for (int i = 0; i < batch; i++)
            {
                data[i, 0] = (float)Math.Sin(i * 0.1);
                data[i, 1] = (float)Math.Cos(i * 0.1);
            }
            return data;
        }
```

```
        private BitmapSource
BitmapFromBitmap(System.Drawing.Bitmap bitmap)
        {
            var hBitmap = bitmap.GetHbitmap();
            try
            {
                return
Imaging.CreateBitmapSourceFromHBitmap(hBitmap, IntPtr.Zero,
Int32Rect.Empty, BitmapSizeOptions.FromEmptyOptions());
            }
            finally
            {
                DeleteObject(hBitmap);
            }
        }


[System.Runtime.InteropServices.DllImport("gdi32.dll")]
        [return:
System.Runtime.InteropServices.MarshalAs(System.Runtime.Intero
pServices.UnmanagedType.Bool)]
        private static extern bool DeleteObject(IntPtr
hObject);
    }
}
```

# ДОДАТОК К

Код візуальної частини інтерфейсу

```
<Window x:Class="AgroAI.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentati
on"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="AgroAI Monitoring System"
        Height="800" Width="1200"
        Background="#e6f2e6"
        ResizeMode="CanResize"
        WindowStartupLocation="CenterScreen">

    <Grid Margin="20">
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="*"/>
        </Grid.RowDefinitions>

        <StackPanel Grid.Row="0" Margin="0,0,0,20">
            <TextBlock Text="Проєкт AgroAI"
                       FontSize="28" FontWeight="Bold"
                       Foreground="#2e7d32"
HorizontalAlignment="Center"/>
            <TextBlock Text="Моніторинг та аналіз пошкоджень
сільськогосподарських земель засобами ШІ"
                       FontSize="16" Foreground="#4d4d4d"
                       HorizontalAlignment="Center"/>
        </StackPanel>

        <Grid Grid.Row="1">
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="220"/>
                <ColumnDefinition Width="*"/>
            </Grid.ColumnDefinitions>
```

```
            <Border Grid.Column="0" Background="#d0e6d0"
Margin="0,0,20,0" Padding="10">
                <StackPanel>
                    <Button Content="Завантажити зображення"
Click="LoadImage_Click" Height="40" Margin="0,0,0,10"
Background="#81c784" Foreground="White"/>
                    <Button Content="Класифікувати (M1)"
Click="Classify_Click" Height="40" Margin="0,0,0,10"
Background="#66bb6a" Foreground="White"/>
                    <Button x:Name="SegmentButton"
Content="Сегментація (M2)" Click="Segment_Click" Height="40"
Margin="0,0,0,10" Background="#4caf50" Foreground="White"
Visibility="Collapsed"/>
                    <Button x:Name="PredictButton"
Content="Прогнозування (M3)" Click="PredictLstm_Click"
Height="40" Margin="0,0,0,10" Background="#388e3c"
Foreground="White" Visibility="Collapsed"/>
                    <Button x:Name="SaveReportButton"
Content="Зберегти звіт" Click="SaveReport_Click" Height="40"
Margin="0,0,0,10" Background="#2e7d32" Foreground="White"/>
                    <ComboBox x:Name="ReportFormatCombo"
Height="20" Margin="0,0,0,5">
                        <ComboBoxItem Content="PDF"
IsSelected="True"/>
                        <ComboBoxItem Content="DOCX"/>
                    </ComboBox>
                </StackPanel>
            </Border>

            <Grid Grid.Column="1">
                <Grid.ColumnDefinitions>
                    <ColumnDefinition Width="*"/>
                    <ColumnDefinition Width="*"/>
                </Grid.ColumnDefinitions>
```

```
                <Border Grid.Column="0" BorderBrush="#2e7d32"
BorderThickness="1" Margin="5">
                    <Image x:Name="OriginalImage"
Stretch="Uniform" Margin="10"/>
                </Border>

                <StackPanel Grid.Column="1" Margin="5"
VerticalAlignment="Stretch">
                    <Border BorderBrush="#388e3c"
BorderThickness="1" Margin="0,0,0,5">
                        <Image x:Name="MaskImage"
Stretch="Uniform" Margin="10" Height="300"/>
                    </Border>

                    <Border BorderBrush="#2e7d32"
BorderThickness="1">
                        <ScrollViewer
VerticalScrollBarVisibility="Auto" Margin="10">
                            <StackPanel>
                                <TextBlock
x:Name="ResultTextBlock" FontSize="16" TextWrapping="Wrap"
Foreground="#2e7d32"/>
                                <TextBlock
x:Name="ForecastTextBlock" FontSize="16" TextWrapping="Wrap"
Foreground="#2e7d32" Margin="0,10,0,0"/>
                            </StackPanel>
                        </ScrollViewer>
                    </Border>
                </StackPanel>
            </Grid>
        </Grid>

        <Grid x:Name="LoadingOverlay" Background="#AAFFFFFF"
Visibility="Collapsed">
```

```
            <TextBlock Text="Обробка..." FontSize="24"
FontWeight="Bold" Foreground="#2e7d32"
HorizontalAlignment="Center" VerticalAlignment="Center"/>
            <ProgressBar IsIndeterminate="True" Height="20"
Width="200" HorizontalAlignment="Center"
VerticalAlignment="Center" Margin="0,40,0,0"/>
        </Grid>
    </Grid>
</Window>
```

# ДОДАТОК Л

Відомість кваліфікаційної роботи

| Позначення | Найменування | Дод. відомості |
|---|---|---|
| | Текстові документи | |
| **1.** | Пояснювальна записка | 95 с. |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | Інші документи | |
| | | |
| **2.** | Презентаційні матеріали | 13 плакатів |
| | | |
| | | |
| | | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | | | |
| | | | | | | | |
| . | . | Прізвище та ініціали. | Підп. | Дата | | | |
| Розробив | | Сільванович К.В. | | | Моніторинг та аналіз пошкоджень сільськогосподарських земель засобами ШІ | Шифр групи | Код напр./спец. |
| Перевірив | | Гриньова О.Є. | | | | ІТШІ-21-3 | 122 |
| Н.контр. | | Малєєва І.А. | | | | ХНУРЕ кафедра ШІ | |
| Затв. | | Золотухін О.В. | | | | | |