**BÁO CÁO CÔNG VIỆC**

**Công việc số:** 71

**Mô tả công việc:** Chuyển mã nguồn tính diện tích chiếm dụng lòng đường từ Python qua C#

**Người thực hiện:** Hạ Quang Dũng

**Ngày bắt đầu:** 01/03/2025

**Ngày kết thúc:** 05/03/2025

***Ghi chú:***

**NỘI DUNG TÀI LIỆU**

[**1. Xây dựng lại thư viện yolo\_grpc 2**](#_vp4jlkkiz7zy)

[1.1. File image\_service.proto 2](#_4rbpwvn9vgij)

[1.2. File vehicle\_density\_analysis.py 3](#_rfzamhor7gaf)

[1.3. File server.py 4](#_1skdrmd8fj0m)

[1.4. File Setup.py 6](#_zc1za2zbzu49)

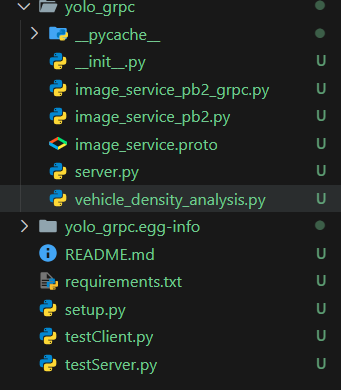
[1.5. Hướng dẫn cách build và cài đặt thư viện 7](#_aht2k73avkwi)

[**2. Chỉnh sửa lại File csproj 7**](#_eo0lozj7guiz)

[**3. Chỉnh sửa lại mã nguồn C# 8**](#_q2t9qpihxm1c)

# 1. Xây dựng lại thư viện yolo\_grpc

Cấu trúc thư mục như sau:



## 1.1. File image\_service.proto

| syntax = "proto3";  service ImageTransfer {  rpc SendImage (ImageRequest) returns (ImageResponse);  rpc CalculateDensity (DensityRequest) returns (DensityResponse); }  message ImageRequest {  string path = 1;  string model\_path = 2; }  message ImageResponse {  bool success = 1;  string message = 2;  string data = 3; }  message DensityRequest {  string model\_path = 1;  string pathImageProcessed = 2;  string pathImage = 3; }  message DensityResponse {  bool success = 1;  string message = 2;  float density = 3; } |
| --- |

## 1.2. File vehicle\_density\_analysis.py

| import cv2 import numpy as np from ultralytics import YOLO  def load\_and\_process\_road\_mask(processed\_image\_path):  processed\_image = cv2.imread(processed\_image\_path)  if processed\_image is None:  raise FileNotFoundError(f"Không thể đọc ảnh: {processed\_image\_path}")   gray\_road\_mask = cv2.cvtColor(processed\_image, cv2.COLOR\_BGR2GRAY)  \_, road\_mask = cv2.threshold(gray\_road\_mask, 200, 255, cv2.THRESH\_BINARY)   return np.count\_nonzero(road\_mask == 255)  def detect\_vehicles(original\_image\_path, model\_path):  model = YOLO(model\_path)  original\_image = cv2.imread(original\_image\_path)  if original\_image is None:  raise FileNotFoundError(f"Không thể đọc ảnh: {original\_image\_path}")   results = model(original\_image)   vehicle\_list = ["car", "truck", "motorcycle", "bicycle", "bus", "person"]  sum\_vehicle\_area = 0   for i in range(len(results[0].masks)):  class\_id = int(results[0].boxes.cls[i])  if model.names[class\_id] in vehicle\_list:  mask = results[0].masks.data[i]  sum\_vehicle\_area += np.count\_nonzero(mask)   return sum\_vehicle\_area  def calculate\_occupancy(road\_area, vehicle\_area):  return (vehicle\_area / road\_area) \* 100 if road\_area > 0 else 0  def process\_image\_data(model\_path, pathImageProcessed, pathImage):  road\_area = load\_and\_process\_road\_mask(pathImageProcessed)  vehicle\_area = detect\_vehicles(pathImage, model\_path)  return calculate\_occupancy(road\_area, vehicle\_area) |
| --- |

## 1.3. File server.py

| import grpc from concurrent import futures from yolo\_grpc import image\_service\_pb2 from yolo\_grpc import image\_service\_pb2\_grpc import cv2 import os import numpy as np from ultralytics import YOLO import json from .vehicle\_density\_analysis import process\_image\_data   # Hàm phát hiện đối tượng def detect\_objects(image, model):  results = model(image)  detections = []  for result in results:  for box in result.boxes:  x1, y1, x2, y2 = box.xyxy[0].cpu().numpy()  confidence = float(box.conf[0].cpu().numpy())  class\_id = int(box.cls[0].cpu().numpy())  label = model.names[class\_id]   detections.append({  "x": int(x1),  "y": int(y1),  "w": int(x2 - x1),  "h": int(y2 - y1),  "label": label,  "confidence": round(confidence, 2)  })  return detections   class ImageTransferServicer(image\_service\_pb2\_grpc.ImageTransferServicer):  def SendImage(self, request, context):  try:  image\_path = request.path  model\_path = request.model\_path    # Sử dụng model\_path để load model  model = YOLO(model\_path)    # Kiểm tra file tồn tại  if not os.path.exists(image\_path):  return image\_service\_pb2.ImageResponse(  success=False,   message=f"File not found: {image\_path}"  )    # Đọc ảnh  image = cv2.imread(image\_path)    if image is None:  return image\_service\_pb2.ImageResponse(  success=False,   message=f"Cannot read image: {image\_path}"  )    # Phát hiện đối tượng  detections = detect\_objects(image, model)    # Chuyển danh sách phát hiện đối tượng thành chuỗi JSON  detections\_json = json.dumps(detections)    return image\_service\_pb2.ImageResponse(  success=True,   message="Detection successful",  data=detections\_json # Trả về dữ liệu JSON  )    except Exception as e:  return image\_service\_pb2.ImageResponse(  success=False,   message=str(e)  )   def CalculateDensity(self, request, context):  try:  density = process\_image\_data(request.model\_path, request.pathImageProcessed, request.pathImage)   return image\_service\_pb2.DensityResponse(  success=True,  message="Density calculation successful",  density=density  )  except Exception as e:  return image\_service\_pb2.DensityResponse(success=False, message=str(e), density=0.0)    def serve():  server = grpc.server(futures.ThreadPoolExecutor(max\_workers=10))    image\_service\_pb2\_grpc.add\_ImageTransferServicer\_to\_server(  ImageTransferServicer(), server)    # Lắng nghe trên port 50051  server.add\_insecure\_port('[::]:50051')  server.start()    print("gRPC server started, listening on port 50051")  server.wait\_for\_termination() |
| --- |

## 1.4. File Setup.py

| from setuptools import setup, find\_packages  setup(  name='yolo\_grpc',  version='0.1.0',  packages=find\_packages(include=['yolo\_grpc', 'yolo\_grpc.\*']),  install\_requires=[  'protobuf',  'grpcio-tools',  'grpcio',  'opencv-python',  'ultralytics',  'numpy',  'torch',  'PyYAML',  ],  author='Ha Quang Dung',  author\_email='dungha.4work@gmail.com',  description='A gRPC service for object detection using YOLO',  long\_description=open('README.md', encoding='utf-8').read(),  long\_description\_content\_type='text/markdown',  url='https://github.com/<your-github-username>/yolo\_grpc',  classifiers=[  'Programming Language :: Python :: 3',  'License :: OSI Approved :: MIT License',  'Operating System :: OS Independent',  ],  python\_requires='>=3.7', ) |
| --- |

## 1.5. Hướng dẫn cách build và cài đặt thư viện

Chạy proto

| python -m grpc\_tools.protoc -I. --python\_out=. --grpc\_python\_out=. image\_service.proto |
| --- |

Build thư viện

| python setup.py sdist bdist\_wheel |
| --- |

# 2. Chỉnh sửa lại File csproj

Cấu hình thêm gRPC client trong dự án C#.

Thêm dependency Grpc.Net.Client vào .csproj:

| <ItemGroup>  <Protobuf Include="src/Python/image\_service.proto" GrpcServices="Both"/> </ItemGroup> |
| --- |

# 3. Chỉnh sửa lại mã nguồn C#

| using VehicleDetection.src.CSharp.Models;  using VehicleDetection.src.CSharp.Services;  using System.Text;  using Newtonsoft.Json;  using System.Diagnostics;  using Grpc.Net.Client;  using VehicleDetection\_8.\_0\_.src.CSharp.Models;  namespace VehicleDetection\_8.\_0\_  {  public partial class MainForm : Form  {  #region Thuộc tính  private PythonExecutor \_pythonExecutor;  private ImageExtractor \_imageExtractor;  private int \_frameSkipQuantity;  private Dictionary<string, DetectionResult> \_frameTimeExecute = new();  private Stopwatch \_stopwatch = new();  // Đường dẫn thư mục và file  private string \_rootDir;  private string \_extractImageFolder;  private string \_logFilePath;  private string \_modelPath;  private string \_videoPath;  private string \_modelSegmentPath;  private string \_imageProcessedPath;  #endregion  #region Nhóm hàm khởi tạo  public MainForm()  {  InitializeComponent();  InitializePaths();  }  private void InitializePaths()  {  \_rootDir = Path.GetFullPath(Path.Combine("..", "..", ".."));  \_extractImageFolder = Path.Combine(\_rootDir, "resources", "Image", "ExtractFromVideo");  \_logFilePath = Path.Combine(\_rootDir, "resources", "Logs", "Log.txt");  \_modelPath = Path.Combine(\_rootDir, "model", "yolov8n.pt");  \_imageExtractor = new ImageExtractor(\_extractImageFolder);  \_frameSkipQuantity = (int)nmrframeSkip.Value;  \_modelSegmentPath = Path.Combine(\_rootDir, "model", "yolov8n-seg.pt");  \_imageProcessedPath = Path.Combine(\_rootDir, "resources", "Image", "Processed", "img5\_mask.jpg");  }  #endregion  #region Nhóm hàm sự kiện  private async void Form1\_Load(object sender, EventArgs e)  {  StartPythonGRPCServer();  }  private void MainForm\_FormClosing(object sender, FormClosingEventArgs e)  {  //PortKiller.KillAllProcessesAndConnectionsOnPort(50051);  }  private void btnSelectFile\_Click(object sender, EventArgs e)  {  SelectVideoFile();  }  private void btnExtractImages\_Click(object sender, EventArgs e)  {  if (\_videoPath == null)  {  MessageBox.Show("Vui lòng chọn Video trước khi trích xuất!");  return;  }  \_stopwatch = Stopwatch.StartNew();  Task.Factory.StartNew(() => ProcessImage());  }  private void nmrframeSkip\_ValueChanged(object sender, EventArgs e)  {  \_frameSkipQuantity = (int)nmrframeSkip.Value;  }  #endregion  #region Nhóm hàm xử lý chính  private void SelectVideoFile()  {  using var openFileDialog = new OpenFileDialog  {  Title = "Chọn File Video",  Filter = "Video Files|\*.mp4;\*.avi;\*.mov;\*.mkv|All Files|\*.\*"  };  if (openFileDialog.ShowDialog() == DialogResult.OK)  {  \_videoPath = openFileDialog.FileName;  wmpVideo.URL = \_videoPath;  wmpVideo.Ctlcontrols.play();  }  }  private void StartPythonGRPCServer()  {  string scriptPath = Path.Combine(\_rootDir, "src", "Python", "GRPCServer.py");  \_pythonExecutor = new PythonExecutor("python", scriptPath);  \_pythonExecutor.Execute(string.Empty);  }  #endregion  #region Trích xuất Hình ảnh  private async Task ProcessImage()  {  ImageExtractor imageExtractor = new ImageExtractor(\_extractImageFolder);  imageExtractor.ExtractImages(\_videoPath, \_frameSkipQuantity);  try  {  DetectionResult detectionResult = new DetectionResult();  string[] allFiles = Directory.GetFiles(\_extractImageFolder);  foreach (string file in allFiles)  {  \_stopwatch = Stopwatch.StartNew();  \_stopwatch.Start();  var (results, density) = await gRpc(file);  \_stopwatch.Stop();  foreach (var box in results)  {  string label = box.label.ToString();  if (detectionResult.VehicleCounts.ContainsKey(label))  {  detectionResult.VehicleCounts[label]++;  }  else  {  detectionResult.VehicleCounts[label] = 1;  }  }  Invoke(new Action(() =>  {  pictureBox1.Image = Helper.DrawBoundingBoxes(file, results);  pictureBox1.SizeMode = PictureBoxSizeMode.Zoom;  pictureBox1.Refresh();  lbPercent.Text = $"Mật độ phương tiện chiếm dụng: {Math.Round(density, 2)}%";  }  )  );  detectionResult.TotalTime = (double)\_stopwatch.Elapsed.TotalSeconds;  detectionResult.TotalVehicles = (int)detectionResult.VehicleCounts.Values.Sum();  UpdateDataGridView(detectionResult.VehicleCounts);  File.Delete(file);  Invoke(new Action(() => DisplayDetectionResult(detectionResult)));  detectionResult.VehicleCounts.Clear();  }  }  catch (Exception ex)  {  Console.WriteLine($"Có lỗi xảy ra: {ex.Message}");  }  }  private async Task<(List<dynamic>, float)> gRpc(string imagePath)  {  try  {  // Địa chỉ server gRPC  using var channel = GrpcChannel.ForAddress("http://localhost:50051");  // Tạo client  var client = new ImageTransfer.ImageTransferClient(channel);  //// Gửi yêu cầu tới server  var request = new ImageRequest { Path = imagePath, ModelPath = \_modelPath };  var response = await client.SendImageAsync(request);  // Create the request object  var request2 = new DensityRequest  {  ModelPath = \_modelSegmentPath,  PathImageProcessed = \_imageProcessedPath,  PathImage = imagePath  };  var response2 = await client.CalculateDensityAsync(request2);  // Xử lý phản hồi  if (response2.Success)  {  Console.WriteLine("Message: " + response2.Message);  Console.WriteLine("Density: " + response2.Density);  // Phân tích JSON thành danh sách dynamic  List<dynamic> detections = JsonConvert.DeserializeObject<List<dynamic>>(response.Data);  // Trả về danh sách phát hiện  return (detections, response2.Density);  }  else  {  return (new List<dynamic>(), 0); // Trả về danh sách rỗng nếu có lỗi  }  }  catch (Exception ex)  {  Console.WriteLine($"An error occurred: {ex.Message}");  //return new List<dynamic>(); // Trả về danh sách rỗng nếu có ngoại lệ  return (new List<dynamic>(), 0); // Trả về danh sách rỗng nếu có ngoại lệ  }  }  #endregion  #region Cập nhật Giao diện Người dùng (UI)  private void DisplayDetectionResult(DetectionResult detectionResult)  {  if (detectionResult.VehicleCounts == null) return;  lbTotalTime.Text = $"Tổng thời gian thực hiện: {detectionResult.TotalTime} giây";  lbTotalVehicles.Text = $"Tổng số phương tiện: {detectionResult.TotalVehicles}";  dataGridView1.Columns.Clear();  dataGridView1.Columns.Add("VehicleType", "Loại Phương Tiện");  dataGridView1.Columns.Add("Count", "Số Lượng");  dataGridView1.Rows.Clear();  foreach (var result in detectionResult.VehicleCounts)  {  dataGridView1.Rows.Add(result.Key, result.Value);  }  }  private async void UpdateDataGridView(Dictionary<string, int> results)  {  Invoke((Action)(() =>  {  dataGridView1.Columns.Clear();  dataGridView1.Columns.Add("VehicleType", "Loại Phương Tiện");  dataGridView1.Columns.Add("Count", "Số Lượng");  dataGridView1.Rows.Clear();  foreach (var result in results)  {  dataGridView1.Rows.Add(result.Key, result.Value);  }  }));  }  #endregion  }  } |
| --- |