*"""*

*Parking Space Detection System using YOLO Object Detection*

*ATTRIBUTION & ACKNOWLEDGMENTS:*

*- YOLO (You Only Look Once) algorithm: Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016)*

*- YOLOv5 implementation: Ultralytics (https://github.com/ultralytics/yolov5)*

*- Pre-trained COCO weights: Microsoft COCO Dataset (https://cocodataset.org)*

*- OpenCV library: https://opencv.org*

*- This implementation combines YOLO object detection with custom parking space management*

*ACADEMIC INTEGRITY NOTE:*

*This code uses publicly available pre-trained models and properly attributes all sources.*

*The parking space detection logic and ROI management is original implementation.*

*"""*

*import cv2*

*import numpy as np*

*import pandas as pd*

*from datetime import datetime*

*import json*

*import os*

*import torch*

*import csv*

*class YOLOParkingDetector:*

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

*self.****parking\_spaces*** *= []*

*self.****model*** *= None*

*self.****device*** *= 'cpu'*

*self.****prev\_status*** *= {}  # smoothing memory*

*self.****status\_consistency*** *= {}  # track consistency over frames*

*self.****current\_frame*** *= None*

*self.****frame\_buffer*** *= []  # for temporal smoothing*

*def download\_yolo\_model(self):*

*"""Load YOLO model with enhanced error handling"""*

*print("Setting up YOLO model...")*

*try:*

*from ultralytics import YOLO*

*print("Using Ultralytics YOLOv8s")*

*self.****model*** *= YOLO("yolov8s.pt")*

*# Warm up the model*

*dummy\_frame = np.****zeros****((640, 640, 3), dtype=np.****uint8****)*

*\_ = self.****model****(dummy\_frame, verbose=False)*

*print("✓ Model loaded and warmed up successfully")*

*return True*

*except Exception as e:*

*print(f"Error loading YOLO: {e}")*

*print("Run: pip install ultralytics torch torchvision")*

*return False*

*def detect\_vehicles\_enhanced(self, frame, confidence\_threshold=0.25):*

*"""Enhanced vehicle detection with multiple preprocessing techniques"""*

*if self.****model*** *is None:*

*return []*

*vehicles = []*

*# Method 1: Standard detection*

*results = self.****model****(frame, imgsz=1280, verbose=False, conf=confidence\_threshold)*

*vehicles.extend(self.\_extract\_vehicles\_from\_results(results, "standard"))*

*# Method 2: Enhanced contrast detection (for poorly lit vehicles)*

*enhanced\_frame = self.\_enhance\_contrast(frame)*

*results\_enhanced = self.****model****(enhanced\_frame, imgsz=1280, verbose=False, conf=confidence\_threshold-0.05)*

*vehicles.extend(self.\_extract\_vehicles\_from\_results(results\_enhanced, "enhanced"))*

*# Method 3: Histogram equalized detection (for shadow areas)*

*eq\_frame = self.\_histogram\_equalize(frame)*

*results\_eq = self.****model****(eq\_frame, imgsz=1280, verbose=False, conf=confidence\_threshold-0.05)*

*vehicles.extend(self.\_extract\_vehicles\_from\_results(results\_eq, "histogram\_eq"))*

*# Remove duplicates using NMS*

*vehicles = self.\_remove\_duplicate\_detections(vehicles)*

*return vehicles*

*def \_extract\_vehicles\_from\_results(self, results, method\_name):*

*"""Extract vehicle information from YOLO results"""*

*vehicles = []*

*for result in results:*

*if result.boxes is not None:*

*for box in result.boxes:*

*conf = float(box.conf[0])*

*cid = int(box.cls[0])*

*# Vehicle class IDs in COCO: car=2, motorcycle=3, bus=5, truck=7*

*if cid in [2, 3, 5, 7]:*

*x1, y1, x2, y2 = box.xyxy[0].tolist()*

*vehicles.append({*

*"bbox": [int(x1), int(y1), int(x2), int(y2)],*

*"confidence": conf,*

*"class": result.names[cid],*

*"method": method\_name*

*})*

*return vehicles*

*def \_enhance\_contrast(self, frame):*

*"""Enhance frame contrast using CLAHE"""*

*lab = cv2.cvtColor(frame, cv2.COLOR\_BGR2LAB)*

*clahe = cv2.createCLAHE(clipLimit=3.0, tileGridSize=(8, 8))*

*lab[:, :, 0] = clahe.apply(lab[:, :, 0])*

*return cv2.cvtColor(lab, cv2.COLOR\_LAB2BGR)*

*def \_histogram\_equalize(self, frame):*

*"""Apply histogram equalization"""*

*yuv = cv2.cvtColor(frame, cv2.COLOR\_BGR2YUV)*

*yuv[:, :, 0] = cv2.equalizeHist(yuv[:, :, 0])*

*return cv2.cvtColor(yuv, cv2.COLOR\_YUV2BGR)*

*def \_remove\_duplicate\_detections(self, vehicles, iou\_threshold=0.5):*

*"""Remove duplicate detections using Non-Maximum Suppression"""*

*if len(vehicles) <= 1:*

*return vehicles*

*boxes = np.array([v["bbox"] for v in vehicles], dtype=np.****float32****)*

*scores = np.array([v["confidence"] for v in vehicles], dtype=np.****float32****)*

*# Apply NMS*

*indices = cv2.dnn.NMSBoxes(boxes.tolist(), scores.tolist(), 0.1, iou\_threshold)*

*if len(indices) > 0:*

*indices = indices.flatten()*

*return [vehicles[i] for i in indices]*

*return vehicles*

*def select\_parking\_spaces(self, image\_path):*

*"""Interactive parking space selection with enhanced UI"""*

*print("=== PARKING SPACE SELECTION MODE ===")*

*print("Instructions:")*

*print("• Click and drag to select each parking space")*

*print("• Press 'n' to confirm current selection")*

*print("• Press 'r' to reset current selection")*

*print("• Press 'd' to delete last space")*

*print("• Press 'q' to finish selection")*

*image = cv2.imread(image\_path)*

*if image is None:*

*print(f"Error: Could not load image {image\_path}")*

*return False*

*clone = image.copy()*

*drawing = False*

*start\_point = None*

*current\_rect = None*

*def mouse\_callback(event, x, y, flags, param):*

*nonlocal drawing, start\_point, current\_rect, image*

*if event == cv2.EVENT\_LBUTTONDOWN:*

*drawing = True*

*start\_point = (x, y)*

*elif event == cv2.EVENT\_MOUSEMOVE and drawing:*

*image = clone.copy()*

*self.\_draw\_existing\_spaces(image)*

*current\_rect = (start\_point[0], start\_point[1], x - start\_point[0], y - start\_point[1])*

*cv2.rectangle(image, start\_point, (x, y), (255, 0, 0), 2)*

*cv2.putText(image, f'Space {len(self.****parking\_spaces****)+1} (selecting...)',*

*(start\_point[0], start\_point[1]-5), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (255, 0, 0), 1)*

*elif event == cv2.EVENT\_LBUTTONUP and drawing:*

*drawing = False*

*cv2.namedWindow('Select Parking Spaces', cv2.WINDOW\_NORMAL)*

*cv2.resizeWindow('Select Parking Spaces', 1200, 800)*

*cv2.setMouseCallback('Select Parking Spaces', mouse\_callback)*

*while True:*

*cv2.imshow('Select Parking Spaces', image)*

*key = cv2.waitKey(1) & 0xFF*

*if key == ord('n'):  # Confirm space*

*if current\_rect and abs(current\_rect[2]) > 30 and abs(current\_rect[3]) > 30:*

*space = {*

*'id': len(self.****parking\_spaces****) + 1,*

*'x': min(current\_rect[0], current\_rect[0] + current\_rect[2]),*

*'y': min(current\_rect[1], current\_rect[1] + current\_rect[3]),*

*'w': abs(current\_rect[2]),*

*'h': abs(current\_rect[3])*

*}*

*self.****parking\_spaces****.append(space)*

*print(f"✓ Space {len(self.****parking\_spaces****)} added: {space['w']}x{space['h']} at ({space['x']},{space['y']})")*

*current\_rect = None*

*image = clone.copy()*

*self.\_draw\_existing\_spaces(image)*

*else:*

*print("⚠ Please select a larger area (minimum 30x30 pixels)")*

*elif key == ord('r'):  # Reset current selection*

*image = clone.copy()*

*self.\_draw\_existing\_spaces(image)*

*current\_rect = None*

*print("Current selection reset")*

*elif key == ord('d'):  # Delete last space*

*if self.****parking\_spaces****:*

*deleted = self.****parking\_spaces****.pop()*

*print(f"✗ Deleted space {deleted['id']}")*

*image = clone.copy()*

*self.\_draw\_existing\_spaces(image)*

*elif key == ord('q'):  # Quit*

*break*

*cv2.destroyAllWindows()*

*if self.****parking\_spaces****:*

*with open('parking\_spaces\_config.json', 'w') as f:*

*json.dump(self.****parking\_spaces****, f, indent=2)*

*print(f"\n✓ Saved {len(self.****parking\_spaces****)} parking spaces to config file")*

*return True*

*else:*

*print("No parking spaces selected!")*

*return False*

*def \_draw\_existing\_spaces(self, image):*

*"""Draw existing parking spaces on image"""*

*for i, space in enumerate(self.****parking\_spaces****):*

*cv2.rectangle(image, (space['x'], space['y']),*

*(space['x'] + space['w'], space['y'] + space['h']),*

*(0, 255, 0), 2)*

*cv2.putText(image, f'Space {i+1}',*

*(space['x'], space['y']-5),*

*cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (0, 255, 0), 1)*

*def load\_parking\_config(self, config\_file="parking\_spaces\_config.json"):*

*"""Load parking configuration with validation"""*

*try:*

*with open(config\_file, 'r') as f:*

*self.****parking\_spaces*** *= json.load(f)*

*print(f"✓ Loaded {len(self.****parking\_spaces****)} parking spaces from config")*

*return True*

*except FileNotFoundError:*

*print(f"⚠ Config file {config\_file} not found")*

*return False*

*except json.JSONDecodeError:*

*print(f"⚠ Invalid JSON in {config\_file}")*

*return False*

*def is\_parking\_space\_occupied\_enhanced(self, vehicles, space, overlap\_threshold=0.2):*

*"""Enhanced occupancy detection with temporal smoothing"""*

*space\_id = space['id']*

*x1, y1 = space['x'], space['y']*

*x2, y2 = x1 + space['w'], y1 + space['h']*

*area = space['w'] \* space['h']*

*# Method 1: Vehicle overlap detection*

*vehicle\_detected = False*

*best\_vehicle = None*

*best\_overlap = 0*

*for v in vehicles:*

*vx1, vy1, vx2, vy2 = v['bbox']*

*ix1, iy1 = max(x1, vx1), max(y1, vy1)*

*ix2, iy2 = min(x2, vx2), min(y2, vy2)*

*if ix1 < ix2 and iy1 < iy2:*

*inter = (ix2 - ix1) \* (iy2 - iy1)*

*overlap\_ratio = inter / area*

*if overlap\_ratio > overlap\_threshold and overlap\_ratio > best\_overlap:*

*vehicle\_detected = True*

*best\_vehicle = v*

*best\_overlap = overlap\_ratio*

*# Method 2: Visual analysis fallback*

*visual\_occupied = False*

*if self.****current\_frame*** *is not None:*

*roi = self.****current\_frame****[y1:y2, x1:x2]*

*if roi.size > 0:*

*visual\_occupied = self.\_analyze\_roi\_occupancy(roi, area)*

*# Method 3: Temporal consistency check*

*current\_decision = vehicle\_detected or visual\_occupied*

*# Initialize consistency tracker*

*if space\_id not in self.****status\_consistency****:*

*self.****status\_consistency****[space\_id] = {'occupied\_count': 0, 'free\_count': 0, 'history': []}*

*# Update consistency tracker*

*consistency = self.****status\_consistency****[space\_id]*

*consistency['history'].append(current\_decision)*

*# Keep only last 5 frames for smoothing*

*if len(consistency['history']) > 5:*

*consistency['history'].pop(0)*

*# Calculate consistency score*

*occupied\_frames = sum(consistency['history'])*

*total\_frames = len(consistency['history'])*

*# Decision logic: require majority vote for status change*

*if total\_frames >= 3:*

*final\_decision = occupied\_frames >= (total\_frames // 2 + 1)*

*else:*

*final\_decision = current\_decision*

*return final\_decision, best\_vehicle, vehicle\_detected, visual\_occupied*

*def \_analyze\_roi\_occupancy(self, roi, area):*

*"""Analyze ROI for visual occupancy indicators"""*

*if roi.size == 0:*

*return False*

*gray = cv2.cvtColor(roi, cv2.COLOR\_BGR2GRAY)*

*# Edge detection for car outlines*

*edges = cv2.Canny(gray, 50, 150)*

*edge\_density = np.sum(edges > 0) / area*

*# Intensity analysis (darker = more likely occupied)*

*mean\_intensity = np.mean(gray)*

*intensity\_factor = max(0, (120 - mean\_intensity) / 120)*

*# Color variance (cars have more varied colors)*

*color\_variance = np.var(roi)*

*variance\_factor = min(color\_variance / 5000, 1.0)*

*# Texture analysis using standard deviation*

*texture\_score = np.std(gray) / 255.0*

*# Combined occupancy score*

*occupancy\_score = (*

*min(edge\_density \* 15, 0.35) +    # Edge contribution*

*intensity\_factor \* 0.25 +         # Shadow contribution*

*variance\_factor \* 0.25 +          # Color variance contribution*

*texture\_score \* 0.15              # Texture contribution*

*)*

*return occupancy\_score > 0.42*

*def process\_video(self, video\_path, output\_video="enhanced\_yolo\_parking.mp4", csv\_path="parking\_analysis.csv"):*

*"""Enhanced video processing with improved detection"""*

*if not self.****parking\_spaces****:*

*print("❌ No parking spaces defined! Run space selection first.")*

*return*

*if self.****model*** *is None:*

*if not self.download\_yolo\_model():*

*return*

*cap = cv2.VideoCapture(video\_path)*

*if not cap.isOpened():*

*print(f"❌ Could not open video: {video\_path}")*

*return*

*fps = int(cap.get(cv2.CAP\_PROP\_FPS))*

*w, h = int(cap.get(cv2.CAP\_PROP\_FRAME\_WIDTH)), int(cap.get(cv2.CAP\_PROP\_FRAME\_HEIGHT))*

*total\_frames = int(cap.get(cv2.CAP\_PROP\_FRAME\_COUNT))*

*print(f"📹 Processing video: {w}x{h} @ {fps}fps, {total\_frames} frames")*

*# Try multiple codec options for best compatibility*

*codec\_options = [*

*('X264', cv2.VideoWriter\_fourcc(\*'X264')),  # H.264 alternative*

*('XVID', cv2.VideoWriter\_fourcc(\*'XVID')),  # XVID codec*

*('MJPG', cv2.VideoWriter\_fourcc(\*'MJPG')),  # Motion JPEG*

*('mp4v', cv2.VideoWriter\_fourcc(\*'mp4v'))   # Original fallback*

*]*

*out = None*

*for codec\_name, fourcc in codec\_options:*

*print(f"Trying codec: {codec\_name}")*

*out = cv2.VideoWriter(output\_video, fourcc, fps, (w, h))*

*if out.isOpened():*

*print(f"✓ Successfully using {codec\_name} codec")*

*break*

*else:*

*out.release()*

*if out is None or not out.isOpened():*

*print("❌ Could not initialize video writer with any codec")*

*return*

*# Setup enhanced CSV output*

*csv\_file = open(csv\_path, 'w', newline='')*

*fieldnames = ['frame', 'timestamp', 'vehicles\_detected', 'vehicles\_standard', 'vehicles\_enhanced',*

*'total\_spaces', 'occupied', 'free', 'occupancy\_rate']*

*fieldnames += [f"space\_{s['id']}" for s in self.****parking\_spaces****]*

*fieldnames += [f"space\_{s['id']}\_method" for s in self.****parking\_spaces****]*

*writer = csv.DictWriter(csv\_file, fieldnames=fieldnames)*

*writer.writeheader()*

*frame\_count = 0*

*start\_time = datetime.now()*

*while True:*

*ret, frame = cap.read()*

*if not ret:*

*break*

*frame\_count += 1*

*timestamp = frame\_count / fps*

*self.****current\_frame*** *= frame.copy()*

*# Enhanced vehicle detection*

*vehicles = self.detect\_vehicles\_enhanced(frame)*

*# Count vehicles by detection method*

*standard\_vehicles = len([v for v in vehicles if v.get('method') == 'standard'])*

*enhanced\_vehicles = len([v for v in vehicles if v.get('method') in ['enhanced', 'histogram\_eq']])*

*occupied = 0*

*free = 0*

*row = {*

*"frame": frame\_count,*

*"timestamp": round(timestamp, 2),*

*"vehicles\_detected": len(vehicles),*

*"vehicles\_standard": standard\_vehicles,*

*"vehicles\_enhanced": enhanced\_vehicles,*

*"total\_spaces": len(self.****parking\_spaces****)*

*}*

*# Process each parking space*

*for space in self.****parking\_spaces****:*

*sid = space['id']*

*is\_occ, best\_vehicle, vehicle\_detected, visual\_detected = self.is\_parking\_space\_occupied\_enhanced(vehicles, space)*

*# Determine detection method*

*if vehicle\_detected and visual\_detected:*

*method = "vehicle+visual"*

*elif vehicle\_detected:*

*method = "vehicle\_only"*

*elif visual\_detected:*

*method = "visual\_only"*

*else:*

*method = "empty"*

*row[f"space\_{sid}\_method"] = method*

*if is\_occ:*

*occupied += 1*

*row[f"space\_{sid}"] = "occupied"*

*color = (0, 0, 255)  # Red*

*status = "OCCUPIED"*

*# Draw vehicle bounding box if detected*

*if best\_vehicle:*

*vx1, vy1, vx2, vy2 = best\_vehicle['bbox']*

*cv2.rectangle(frame, (vx1, vy1), (vx2, vy2), (255, 0, 255), 2)*

*cv2.putText(frame, f"{best\_vehicle['class']} {best\_vehicle['confidence']:.2f}",*

*(vx1, vy1-5), cv2.FONT\_HERSHEY\_SIMPLEX, 0.4, (255, 0, 255), 1)*

*else:*

*free += 1*

*row[f"space\_{sid}"] = "free"*

*color = (0, 255, 0)  # Green*

*status = "FREE"*

*# Draw parking space with enhanced visualization*

*thickness = 3 if is\_occ else 2*

*cv2.rectangle(frame, (space['x'], space['y']),*

*(space['x'] + space['w'], space['y'] + space['h']), color, thickness)*

*# Enhanced status text*

*status\_text = f"P{sid}: {status}"*

*if method != "empty":*

*method\_short = {"vehicle+visual": "V+V", "vehicle\_only": "VEH", "visual\_only": "VIS"}*

*status\_text += f" ({method\_short.get(method, method)})"*

*cv2.putText(frame, status\_text, (space['x'], space['y'] - 5),*

*cv2.FONT\_HERSHEY\_SIMPLEX, 0.45, color, 1)*

*# Update row data*

*row["occupied"] = occupied*

*row["free"] = free*

*row["occupancy\_rate"] = round((occupied / len(self.****parking\_spaces****)) \* 100, 1)*

*writer.writerow(row)*

*# Enhanced UI display*

*cv2.putText(frame, f"🅿️ ENHANCED PARKING DETECTION | Total: {len(self.****parking\_spaces****)} | Occupied: {occupied} | Free: {free}",*

*(10, 30), cv2.FONT\_HERSHEY\_SIMPLEX, 0.6, (255, 255, 255), 2)*

*cv2.putText(frame, f"🚗 Vehicles: {len(vehicles)} (Std:{standard\_vehicles} Enh:{enhanced\_vehicles}) | Frame: {frame\_count}/{total\_frames}",*

*(10, 60), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (200, 200, 200), 1)*

*cv2.putText(frame, f"⏱️ Time: {timestamp:.1f}s | Occupancy: {row['occupancy\_rate']}%",*

*(10, h - 20), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (255, 255, 255), 1)*

*# Attribution watermark*

*cv2.putText(frame, "YOLOv8: Ultralytics | Enhanced Detection",*

*(w - 280, h - 5), cv2.FONT\_HERSHEY\_SIMPLEX, 0.3, (150, 150, 150), 1)*

*out.write(frame)*

*# Progress update*

*if frame\_count % 30 == 0 or frame\_count == total\_frames:*

*elapsed = (datetime.now() - start\_time).total\_seconds()*

*fps\_actual = frame\_count / elapsed if elapsed > 0 else 0*

*progress = (frame\_count / total\_frames) \* 100*

*print(f"📊 Progress: {frame\_count}/{total\_frames} ({progress:.1f}%) | FPS: {fps\_actual:.1f}")*

*cap.release()*

*out.release()*

*csv\_file.close()*

*# Final summary*

*processing\_time = (datetime.now() - start\_time).total\_seconds()*

*print(f"\n✅ Processing complete!")*

*print(f"📹 Output video: {output\_video}")*

*print(f"📊 CSV results: {csv\_path}")*

*print(f"⏱️ Processing time: {processing\_time:.1f}s")*

*print(f"🔄 Average FPS: {frame\_count/processing\_time:.1f}")*

*def extract\_reference\_frame(video\_path, output\_path='reference\_frame.jpg', frame\_number=30):*

*"""Extract reference frame from video"""*

*cap = cv2.VideoCapture(video\_path)*

*if not cap.isOpened():*

*print(f"❌ Could not open video: {video\_path}")*

*return False*

*cap.set(cv2.CAP\_PROP\_POS\_FRAMES, frame\_number)*

*ret, frame = cap.read()*

*if ret:*

*cv2.imwrite(output\_path, frame)*

*print(f"✅ Reference frame saved: {output\_path}")*

*cap.release()*

*return True*

*else:*

*print("❌ Could not extract frame")*

*cap.release()*

*return False*

*def main():*

*"""Enhanced main function with better user experience"""*

*print("="\*70)*

*print("🅿️  ENHANCED YOLO PARKING DETECTION SYSTEM")*

*print("="\*70)*

*print("\n🔧 Features:")*

*print("• Multi-method vehicle detection (Standard + Enhanced + Histogram Equalization)")*

*print("• Visual analysis fallback for missed vehicles")*

*print("• Temporal smoothing for stable results")*

*print("• Real-time CSV output with detection methods")*

*print("• Interactive parking space selection")*

*print("\n📚 Attributions:")*

*print("• YOLO Algorithm: Ultralytics YOLOv8")*

*print("• COCO Dataset: Microsoft")*

*print("• Enhanced detection logic: Original implementation")*

*print("="\*70)*

*detector = YOLOParkingDetector()*

*# Get video path*

*video\_path = "Cam1.mp4"*

*if not os.****path****.exists(video\_path):*

*print(f"❌ Video file not found: {video\_path}")*

*return*

*# Load or create parking configuration*

*if os.****path****.exists('parking\_spaces\_config.json'):*

*use\_existing = input("\n🔧 Found existing parking configuration. Use it? (y/n): ").lower()*

*if use\_existing == 'y':*

*if not detector.load\_parking\_config():*

*print("❌ Failed to load config")*

*return*

*else:*

*print("📸 Extracting reference frame...")*

*if extract\_reference\_frame(video\_path):*

*if not detector.select\_parking\_spaces('reference\_frame.jpg'):*

*return*

*else:*

*return*

*else:*

*print("📸 Extracting reference frame for parking space selection...")*

*if extract\_reference\_frame(video\_path):*

*if not detector.select\_parking\_spaces('reference\_frame.jpg'):*

*return*

*else:*

*return*

*if not detector.****parking\_spaces****:*

*print("❌ No parking spaces configured!")*

*return*

*# Process video*

*output\_video = input("\n📹 Output video name (default: enhanced\_yolo\_parking.mp4): ").strip()*

*if not output\_video:*

*output\_video = "enhanced\_yolo\_parking.mp4"*

*csv\_output = input("📊 CSV output name (default: parking\_analysis.csv): ").strip()*

*if not csv\_output:*

*csv\_output = "parking\_analysis.csv"*

*print(f"\n🚀 Starting enhanced processing...")*

*detector.process\_video(video\_path, output\_video, csv\_output)*

*print(f"\n🎉 All done! Check your output files:")*

*print(f"📹 Video: {output\_video}")*

*print(f"📊 Data: {csv\_output}")*

*print(f"⚙️ Config: parking\_spaces\_config.json")*

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

*main()*