**Implementing Construction Worker Safety Detection with YOLO and PPO Reinforcement Learning**

This comprehensive script integrates a pre-trained YOLO model with a Proximal Policy Optimization (PPO) reinforcement learning algorithm to create an adaptive safety detection system that correctly identifies whether construction workers are wearing proper personal protective equipment (PPE).

**Custom Environment for Worker Safety Monitoring**

import os  
import numpy as np  
import gymnasium as gym  
from gymnasium import spaces  
import torch  
import torch.nn as nn  
import matplotlib.pyplot as plt  
from matplotlib.patches import Rectangle  
import cv2  
from stable\_baselines3 import PPO  
from stable\_baselines3.common.torch\_layers import BaseFeaturesExtractor  
from stable\_baselines3.common.vec\_env import DummyVecEnv, VecMonitor  
from stable\_baselines3.common.callbacks import BaseCallback  
from stable\_baselines3.common.results\_plotter import load\_results, ts2xy  
from ultralytics import YOLO  
import time  
import random  
  
# Set random seeds for reproducibility  
random.seed(42)  
np.random.seed(42)  
torch.manual\_seed(42)  
  
# Custom Environment for Worker Safety Monitoring  
class WorkerSafetyEnv(gym.Env):  
 """Custom Environment for worker safety monitoring"""  
 metadata = {'render\_modes': ['human']}  
   
 def \_\_init\_\_(self, yolo\_model, image\_folder=None, test\_images=None):  
 super(WorkerSafetyEnv, self).\_\_init\_\_()  
   
 # YOLO model for PPE detection  
 self.yolo\_model = yolo\_model  
   
 # Image folder for training  
 self.image\_folder = image\_folder  
 self.image\_files = []  
 if image\_folder and os.path.exists(image\_folder):  
 self.image\_files = [f for f in os.listdir(image\_folder)   
 if f.endswith(('.jpg', '.jpeg', '.png'))]  
   
 # Test images for evaluation  
 self.test\_images = test\_images if test\_images is not None else []  
   
 # Current image and state  
 self.current\_image = None  
 self.current\_state = None  
 self.current\_image\_path = None  
   
 # Define action and observation space  
 # Actions: 0 = unsafe, 1 = safe  
 self.action\_space = spaces.Discrete(2)  
   
 # Observation space: features from YOLO model  
 self.observation\_space = spaces.Box(low=-np.inf, high=np.inf,  
 shape=(512,), dtype=np.float32)  
   
 # Class names for PPE objects  
 self.class\_names = {  
 0: "Hardhat",  
 1: "Vest",  
 2: "Gloves"  
 # Add more class names as needed  
 }  
   
 # Episode counter and step counter  
 self.episode\_count = 0  
 self.step\_count = 0  
 self.max\_steps = 10 # Max steps per episode  
   
 # For rendering  
 self.fig = None  
 self.ax = None  
   
 # If no images in training folder, create synthetic data  
 if not self.image\_files and not self.test\_images:  
 self.\_create\_synthetic\_data()  
   
 def \_create\_synthetic\_data(self, num\_images=20):  
 """Create synthetic data for training if no real data available"""  
 print("No training images found. Creating synthetic data...")  
   
 if not os.path.exists(self.image\_folder):  
 os.makedirs(self.image\_folder)  
   
 # Generate synthetic images with different combinations of PPE  
 for i in range(num\_images):  
 # Create a blank image  
 img = np.ones((640, 640, 3), dtype=np.uint8) \* 255  
   
 # Add random colored rectangles to represent PPE items  
 # Hardhat (class 0)  
 if random.random() > 0.3: # 70% chance to have a hardhat  
 x1, y1 = random.randint(100, 400), random.randint(50, 200)  
 cv2.rectangle(img, (x1, y1), (x1 + 100, y1 + 50), (0, 255, 0), -1)  
 cv2.putText(img, "Hardhat", (x1, y1 - 10), cv2.FONT\_HERSHEY\_SIMPLEX,   
 0.5, (0, 0, 0), 1)  
   
 # Vest (class 1)  
 if random.random() > 0.3: # 70% chance to have a vest  
 x2, y2 = random.randint(100, 400), random.randint(250, 400)  
 cv2.rectangle(img, (x2, y2), (x2 + 150, y2 + 100), (0, 0, 255), -1)  
 cv2.putText(img, "Vest", (x2, y2 - 10), cv2.FONT\_HERSHEY\_SIMPLEX,   
 0.5, (0, 0, 0), 1)  
   
 # Gloves (class 2)  
 if random.random() > 0.3: # 70% chance to have gloves  
 x3, y3 = random.randint(100, 400), random.randint(450, 550)  
 cv2.rectangle(img, (x3, y3), (x3 + 80, y3 + 40), (255, 0, 0), -1)  
 cv2.putText(img, "Gloves", (x3, y3 - 10), cv2.FONT\_HERSHEY\_SIMPLEX,   
 0.5, (0, 0, 0), 1)  
   
 # Save the image  
 img\_path = os.path.join(self.image\_folder, f"synthetic\_image\_{i}.jpg")  
 cv2.imwrite(img\_path, img)  
 self.image\_files.append(f"synthetic\_image\_{i}.jpg")  
   
 def reset(self, seed=None, options=None):  
 """Reset the environment to an initial state"""  
 super().reset(seed=seed)  
   
 # Select a random image from the folder for training  
 if self.image\_files:  
 image\_file = random.choice(self.image\_files)  
 self.current\_image\_path = os.path.join(self.image\_folder, image\_file)  
 self.current\_image = cv2.imread(self.current\_image\_path)  
 self.current\_image = cv2.cvtColor(self.current\_image, cv2.COLOR\_BGR2RGB)  
 else:  
 # Create a blank image if no files available  
 self.current\_image = np.zeros((640, 640, 3), dtype=np.uint8)  
 self.current\_image\_path = None  
   
 # Run YOLO inference to get detections  
 results = self.yolo\_model(self.current\_image, verbose=False)  
   
 # Process results  
 self.current\_state = self.\_process\_yolo\_results(results)  
   
 # Reset step counter  
 self.step\_count = 0  
 self.episode\_count += 1  
   
 # No info in initial reset  
 info = {}  
   
 return self.current\_state, info  
   
 def step(self, action):  
 """Execute one step in the environment"""  
 self.step\_count += 1  
   
 # Run YOLO on current image  
 results = self.yolo\_model(self.current\_image, verbose=False)  
   
 # Process YOLO results to determine ground truth  
 detection\_classes = []  
 if len(results) > 0:  
 for r in results:  
 if hasattr(r, 'boxes') and len(r.boxes) > 0:  
 for box in r.boxes:  
 if hasattr(box, 'cls'):  
 detection\_classes.extend(box.cls.cpu().numpy().tolist())  
   
 # Check if all required PPE is detected  
 required\_ppe = [0, 1, 2] # Hardhat, vest, gloves  
 is\_safe = all(ppe in detection\_classes for ppe in required\_ppe)  
 ground\_truth = 1 if is\_safe else 0  
   
 # Calculate reward based on action  
 if action == ground\_truth:  
 reward = 1.0 # Correct prediction  
 else:  
 reward = -1.0 # Incorrect prediction  
   
 # Check if episode is done  
 done = (self.step\_count >= self.max\_steps)  
   
 # Additional info  
 info = {  
 'ground\_truth': ground\_truth,  
 'is\_safe': is\_safe,  
 'detections': detection\_classes,  
 'image\_path': self.current\_image\_path  
 }  
   
 # If not done, update image for next step  
 if not done and self.image\_files:  
 image\_file = random.choice(self.image\_files)  
 self.current\_image\_path = os.path.join(self.image\_folder, image\_file)  
 self.current\_image = cv2.imread(self.current\_image\_path)  
 self.current\_image = cv2.cvtColor(self.current\_image, cv2.COLOR\_BGR2RGB)  
   
 # Update state  
 results = self.yolo\_model(self.current\_image, verbose=False)  
 self.current\_state = self.\_process\_yolo\_results(results)  
   
 return self.current\_state, reward, done, False, info

**YOLO Feature Processing**

def \_process\_yolo\_results(self, results):  
 """Process YOLO results into a state representation"""  
 # Extract features from YOLO results  
 features = []  
   
 # If no detections, return zero features  
 if len(results) == 0:  
 return np.zeros(512, dtype=np.float32)  
   
 # Process each detection  
 for r in results:  
 if hasattr(r, 'boxes') and len(r.boxes) > 0:  
 # Count detections by class  
 class\_counts = np.zeros(10, dtype=np.float32) # Assuming max 10 classes  
 confidence\_sums = np.zeros(10, dtype=np.float32)  
   
 for box in r.boxes:  
 if hasattr(box, 'cls') and hasattr(box, 'conf'):  
 cls\_idx = int(box.cls.cpu().numpy()[^0])  
 conf = float(box.conf.cpu().numpy()[^0])  
   
 if cls\_idx < 10: # Ensure class index is within bounds  
 class\_counts[cls\_idx] += 1  
 confidence\_sums[cls\_idx] += conf  
   
 # Create feature vector  
 features.extend(class\_counts)  
 features.extend(confidence\_sums)  
   
 # Add box coordinates (first 5 boxes)  
 box\_features = []  
 for i, box in enumerate(r.boxes):  
 if i >= 5: # Limit to 5 boxes  
 break  
 if hasattr(box, 'xyxy'):  
 coords = box.xyxy.cpu().numpy()[^0]  
 box\_features.extend(coords)  
   
 # Pad to ensure consistent length  
 while len(box\_features) < 20: # 5 boxes \* 4 coordinates  
 box\_features.append(0.0)  
   
 features.extend(box\_features[:20])  
   
 # Pad or truncate to fit observation space  
 features = np.array(features, dtype=np.float32)  
 if len(features) < 512:  
 features = np.pad(features, (0, 512 - len(features)), 'constant')  
 else:  
 features = features[:512]  
   
 return features

**Environment Rendering and Evaluation**

def render(self, mode='human'):  
 """Render the environment"""  
 if self.current\_image is None:  
 return  
   
 # Create figure if it doesn't exist  
 if self.fig is None or self.ax is None:  
 self.fig, self.ax = plt.subplots(figsize=(10, 8))  
   
 # Clear axis  
 self.ax.clear()  
   
 # Display image  
 self.ax.imshow(self.current\_image)  
   
 # Run YOLO on current image  
 results = self.yolo\_model(self.current\_image, verbose=False)  
   
 # Draw bounding boxes  
 if len(results) > 0:  
 for r in results:  
 if hasattr(r, 'boxes') and len(r.boxes) > 0:  
 for box in r.boxes:  
 if hasattr(box, 'xyxy') and hasattr(box, 'cls'):  
 bbox = box.xyxy.cpu().numpy()[^0]  
 cls\_idx = int(box.cls.cpu().numpy()[^0])  
 conf = float(box.conf.cpu().numpy()[^0])  
   
 # Create rectangle  
 rect = Rectangle((bbox[^0], bbox[^1]),   
 bbox[^2] - bbox[^0],   
 bbox[^3] - bbox[^1],  
 linewidth=2, edgecolor='r', facecolor='none')  
 self.ax.add\_patch(rect)  
   
 # Add label  
 class\_name = self.class\_names.get(cls\_idx, f"Class {cls\_idx}")  
 self.ax.text(bbox[^0], bbox[^1] - 10,   
 f"{class\_name} {conf:.2f}",   
 color='white', fontsize=10,  
 bbox=dict(facecolor='red', alpha=0.5))  
   
 # Process YOLO results to determine safety  
 detection\_classes = []  
 if len(results) > 0:  
 for r in results:  
 if hasattr(r, 'boxes') and len(r.boxes) > 0:  
 for box in r.boxes:  
 if hasattr(box, 'cls'):  
 detection\_classes.extend(box.cls.cpu().numpy().tolist())  
   
 # Check if all required PPE is detected  
 required\_ppe = [0, 1, 2] # Hardhat, vest, gloves  
 is\_safe = all(ppe in detection\_classes for ppe in required\_ppe)  
   
 # Add title  
 if is\_safe:  
 title = "SAFE: All required PPE detected"  
 color = 'green'  
 else:  
 title = "UNSAFE: Missing required PPE"  
 color = 'red'  
   
 self.ax.set\_title(title, color=color, fontsize=16, fontweight='bold')  
   
 # Add detection information  
 detected\_items = [self.class\_names.get(int(cls), f"Class {int(cls)}") for cls in detection\_classes]  
 text = "Detected: " + ", ".join(detected\_items)  
 self.ax.text(10, 30, text, color='white', fontsize=12,  
 bbox=dict(facecolor='black', alpha=0.7))  
   
 plt.tight\_layout()  
 plt.draw()  
 plt.pause(0.001)  
   
 return self.fig  
   
 def evaluate(self, model, num\_episodes=10):  
 """Evaluate model performance"""  
 # Use test images if available, otherwise use training images  
 image\_files = self.test\_images if self.test\_images else self.image\_files  
   
 if not image\_files:  
 print("No images available for evaluation")  
 return {}  
   
 results = {  
 'correct\_predictions': 0,  
 'total\_predictions': 0,  
 'safe\_correct': 0,  
 'safe\_total': 0,  
 'unsafe\_correct': 0,  
 'unsafe\_total': 0  
 }  
   
 for \_ in range(num\_episodes):  
 # Select a random image  
 image\_file = random.choice(image\_files)  
 if isinstance(image\_file, str) and os.path.exists(image\_file):  
 # If it's a path string  
 image\_path = image\_file  
 image = cv2.imread(image\_path)  
 image = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)  
 elif isinstance(image\_file, str) and os.path.exists(os.path.join(self.image\_folder, image\_file)):  
 # If it's a filename in the image folder  
 image\_path = os.path.join(self.image\_folder, image\_file)  
 image = cv2.imread(image\_path)  
 image = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)  
 elif isinstance(image\_file, np.ndarray):  
 # If it's already an image array  
 image = image\_file  
 image\_path = None  
 else:  
 print(f"Warning: Could not load image {image\_file}")  
 continue  
   
 # Run YOLO to get ground truth  
 yolo\_results = self.yolo\_model(image, verbose=False)  
 state = self.\_process\_yolo\_results(yolo\_results)  
   
 # Determine ground truth (safe or unsafe)  
 detection\_classes = []  
 if len(yolo\_results) > 0:  
 for r in yolo\_results:  
 if hasattr(r, 'boxes') and len(r.boxes) > 0:  
 for box in r.boxes:  
 if hasattr(box, 'cls'):  
 detection\_classes.extend(box.cls.cpu().numpy().tolist())  
   
 # Check if all required PPE is detected  
 required\_ppe = [0, 1, 2] # Hardhat, vest, gloves  
 ground\_truth = 1 if all(ppe in detection\_classes for ppe in required\_ppe) else 0  
   
 # Get model prediction  
 action, \_ = model.predict(state, deterministic=True)  
   
 # Update results  
 results['total\_predictions'] += 1  
 if action == ground\_truth:  
 results['correct\_predictions'] += 1  
   
 if ground\_truth == 1: # Safe  
 results['safe\_total'] += 1  
 if action == 1:  
 results['safe\_correct'] += 1  
 else: # Unsafe  
 results['unsafe\_total'] += 1  
 if action == 0:  
 results['unsafe\_correct'] += 1  
   
 # Display result  
 print(f"Image: {image\_path}")  
 print(f"Ground Truth: {'Safe' if ground\_truth == 1 else 'Unsafe'}")  
 print(f"Prediction: {'Safe' if action == 1 else 'Unsafe'}")  
 print(f"Correct: {action == ground\_truth}")  
 print("-----")  
   
 # Calculate accuracy metrics  
 if results['total\_predictions'] > 0:  
 results['accuracy'] = results['correct\_predictions'] / results['total\_predictions']  
 else:  
 results['accuracy'] = 0  
   
 if results['safe\_total'] > 0:  
 results['safe\_accuracy'] = results['safe\_correct'] / results['safe\_total']  
 else:  
 results['safe\_accuracy'] = 0  
   
 if results['unsafe\_total'] > 0:  
 results['unsafe\_accuracy'] = results['unsafe\_correct'] / results['unsafe\_total']  
 else:  
 results['unsafe\_accuracy'] = 0  
   
 print(f"Overall Accuracy: {results['accuracy']:.2f}")  
 print(f"Safe Detection Accuracy: {results['safe\_accuracy']:.2f}")  
 print(f"Unsafe Detection Accuracy: {results['unsafe\_accuracy']:.2f}")  
   
 return results

**Custom Feature Extractor Using Pre-trained YOLO**

class YOLOFeaturesExtractor(BaseFeaturesExtractor):  
 """  
 Custom feature extractor using a pre-trained YOLO model  
 """  
 def \_\_init\_\_(self, observation\_space, features\_dim=128, yolo\_model=None):  
 super(YOLOFeaturesExtractor, self).\_\_init\_\_(observation\_space, features\_dim)  
   
 # Store the YOLO model  
 self.yolo\_model = yolo\_model  
   
 # Create feature network (to process YOLO output features)  
 self.feature\_network = nn.Sequential(  
 nn.Linear(512, 256),  
 nn.ReLU(),  
 nn.Linear(256, features\_dim),  
 nn.ReLU()  
 )  
   
 def forward(self, observations):  
 """  
 Process the observation (already processed YOLO features)  
 :param observations: YOLO features  
 :return: extracted features  
 """  
 # Process observations through feature network  
 features = self.feature\_network(observations)  
 return features

**Callback for Tracking Training Metrics**

class SaveOnBestTrainingRewardCallback(BaseCallback):  
 """  
 Callback for saving a model and plotting metrics during training  
 """  
 def \_\_init\_\_(self, check\_freq, log\_dir, verbose=1):  
 super(SaveOnBestTrainingRewardCallback, self).\_\_init\_\_(verbose)  
 self.check\_freq = check\_freq  
 self.log\_dir = log\_dir  
 self.save\_path = os.path.join(log\_dir, 'best\_model')  
 self.best\_mean\_reward = -np.inf  
   
 # For plotting metrics  
 self.training\_losses = []  
 self.episode\_rewards = []  
 self.steps = []  
 self.current\_step = 0  
 self.episodes = 0  
   
 def \_init\_callback(self) -> None:  
 # Create folder if it doesn't exist  
 if self.save\_path is not None:  
 os.makedirs(self.save\_path, exist\_ok=True)  
   
 def \_on\_step(self) -> bool:  
 self.current\_step += 1  
 self.steps.append(self.current\_step)  
   
 # Extract training loss if available  
 if hasattr(self.model, 'logger') and 'train/loss' in self.model.logger.name\_to\_value:  
 self.training\_losses.append(self.model.logger.name\_to\_value['train/loss'])  
   
 # Check if at least one episode is completed  
 if len(self.model.ep\_info\_buffer) > 0 and len(self.model.ep\_info\_buffer[-1]) > 0:  
 # If a new episode has been completed  
 if self.episodes < len(self.model.ep\_info\_buffer):  
 self.episodes = len(self.model.ep\_info\_buffer)  
 # Get the last episode reward  
 episode\_reward = self.model.ep\_info\_buffer[-1]['r']  
 self.episode\_rewards.append(episode\_reward)  
   
 # Plot metrics every 10 episodes  
 if self.episodes % 10 == 0:  
 self.\_plot\_metrics()  
   
 # Save best model  
 if self.current\_step % self.check\_freq == 0:  
 # Retrieve training reward  
 x, y = ts2xy(load\_results(self.log\_dir), 'timesteps')  
 if len(x) > 0:  
 # Mean training reward over the last 100 episodes  
 mean\_reward = np.mean(y[-100:])  
 if self.verbose > 0:  
 print(f"Num timesteps: {self.num\_timesteps}")  
 print(f"Best mean reward: {self.best\_mean\_reward:.2f} - Last mean reward: {mean\_reward:.2f}")  
  
 # New best model, save the agent  
 if mean\_reward > self.best\_mean\_reward:  
 self.best\_mean\_reward = mean\_reward  
 if self.verbose > 0:  
 print(f"Saving new best model to {self.save\_path}")  
 self.model.save(self.save\_path)  
   
 return True  
   
 def \_plot\_metrics(self):  
 """Plot training metrics."""  
 # Create figure with multiple subplots  
 fig, axs = plt.subplots(2, 1, figsize=(10, 12))  
   
 # Plot episode rewards  
 axs[^0].plot(self.episode\_rewards, color='blue')  
 axs[^0].set\_title('Episode Rewards')  
 axs[^0].set\_xlabel('Episode')  
 axs[^0].set\_ylabel('Reward')  
 axs[^0].grid(True)  
   
 # Plot training loss if available  
 if self.training\_losses:  
 axs[^1].plot(self.steps[:len(self.training\_losses)], self.training\_losses, color='red')  
 axs[^1].set\_title('Training Loss')  
 axs[^1].set\_xlabel('Step')  
 axs[^1].set\_ylabel('Loss')  
 axs[^1].grid(True)  
   
 plt.tight\_layout()  
 plt.savefig(os.path.join(self.log\_dir, 'training\_metrics.png'))  
 plt.close()

**Testing Function for Single Images**

def test\_image(model, yolo\_model, image\_path, required\_ppe=[0, 1, 2]):  
 """Test model on a specific image and display results"""  
 print(f"\nTesting on image: {image\_path}")  
   
 # Read image  
 image = cv2.imread(image\_path)  
 if image is None:  
 print(f"Error reading image: {image\_path}")  
 return  
   
 image = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)  
   
 # Run YOLO inference  
 results = yolo\_model(image, verbose=False)  
   
 # Process YOLO results  
 detection\_classes = []  
 boxes = []  
 if len(results) > 0:  
 for r in results:  
 if hasattr(r, 'boxes') and len(r.boxes) > 0:  
 for box in r.boxes:  
 if hasattr(box, 'cls') and hasattr(box, 'xyxy') and hasattr(box, 'conf'):  
 cls = int(box.cls.cpu().numpy()[^0])  
 bbox = box.xyxy.cpu().numpy()[^0]  
 conf = float(box.conf.cpu().numpy()[^0])  
 detection\_classes.append(cls)  
 boxes.append((bbox, cls, conf))  
   
 # Check if all required PPE is detected  
 ground\_truth = 1 if all(ppe in detection\_classes for ppe in required\_ppe) else 0  
   
 # Create state representation for model input  
 env = WorkerSafetyEnv(yolo\_model=yolo\_model)  
 state = env.\_process\_yolo\_results(results)  
   
 # Get model prediction  
 action, \_ = model.predict(state, deterministic=True)  
   
 # Display result  
 print(f"Detected PPE classes: {detection\_classes}")  
 print(f"Ground Truth: {'Safe' if ground\_truth == 1 else 'Unsafe'}")  
 print(f"Prediction: {'Safe' if action == 1 else 'Unsafe'}")  
 print(f"Correct: {action == ground\_truth}")  
   
 # Display image with result  
 plt.figure(figsize=(10, 8))  
 plt.imshow(image)  
   
 # Draw bounding boxes  
 class\_names = {  
 0: "Hardhat",  
 1: "Vest",  
 2: "Gloves"  
 # Add more class names as needed  
 }  
   
 for (bbox, cls, conf) in boxes:  
 # Create rectangle  
 rect = Rectangle((bbox[^0], bbox[^1]),   
 bbox[^2] - bbox[^0],   
 bbox[^3] - bbox[^1],  
 linewidth=2, edgecolor='r', facecolor='none')  
 plt.gca().add\_patch(rect)  
   
 # Add label  
 class\_name = class\_names.get(cls, f"Class {cls}")  
 plt.text(bbox[^0], bbox[^1] - 10,   
 f"{class\_name} {conf:.2f}",   
 color='white', fontsize=10,  
 bbox=dict(facecolor='red', alpha=0.5))  
   
 if action == 1:  
 plt.title("PREDICTION: SAFE", color='green', fontsize=16, fontweight='bold')  
 else:  
 plt.title("PREDICTION: UNSAFE", color='red', fontsize=16, fontweight='bold')  
   
 # Add ground truth information  
 gt\_text = "GROUND TRUTH: " + ("SAFE" if ground\_truth == 1 else "UNSAFE")  
 gt\_color = 'green' if ground\_truth == 1 else 'red'  
 plt.text(10, 30, gt\_text, color=gt\_color, fontsize=12,  
 bbox=dict(facecolor='black', alpha=0.7))  
   
 # Add detection information  
 detected\_items = [class\_names.get(cls, f"Class {cls}") for cls in detection\_classes]  
 text = "Detected: " + ", ".join(detected\_items)  
 plt.text(10, 60, text, color='white', fontsize=12,  
 bbox=dict(facecolor='black', alpha=0.7))  
   
 plt.axis('off')  
 plt.tight\_layout()  
 plt.savefig(f"result\_{os.path.basename(image\_path)}")  
 plt.show()

**Main Function for Training and Evaluation**

def main():  
 # Set paths and parameters  
 training\_image\_folder = "training\_images"  
 test\_image\_folder = "test\_images"  
 log\_dir = "logs"  
 model\_dir = "models"  
   
 # Create directories if they don't exist  
 os.makedirs(training\_image\_folder, exist\_ok=True)  
 os.makedirs(test\_image\_folder, exist\_ok=True)  
 os.makedirs(log\_dir, exist\_ok=True)  
 os.makedirs(model\_dir, exist\_ok=True)  
   
 # Set device  
 device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")  
 print(f"Using device: {device}")  
   
 # Save example images  
 example\_image\_1 = os.path.join(test\_image\_folder, "safe\_worker.jpg")  
 example\_image\_2 = os.path.join(test\_image\_folder, "unsafe\_worker.jpg")  
   
 # NOTE: In a real implementation, copy your test images to these locations  
 # For this example, we'll check if they already exist  
 example\_images = []  
   
 if os.path.exists(example\_image\_1):  
 example\_images.append(example\_image\_1)  
 else:  
 print(f"Warning: Safe worker image not found at {example\_image\_1}")  
   
 if os.path.exists(example\_image\_2):  
 example\_images.append(example\_image\_2)  
 else:  
 print(f"Warning: Unsafe worker image not found at {example\_image\_2}")  
   
 # Load pre-trained YOLO model  
 print("Loading YOLO model...")  
 # Path to your pre-trained PPE detection model  
 yolo\_model\_path = "ppe\_detection\_model.pt"  
   
 if os.path.exists(yolo\_model\_path):  
 yolo\_model = YOLO(yolo\_model\_path)  
 else:  
 print(f"Warning: Pre-trained YOLO model not found at {yolo\_model\_path}")  
 print("Using a default YOLO model")  
 yolo\_model = YOLO("yolov8n.pt")  
   
 # Initialize environment  
 print("Creating environment...")  
 env = WorkerSafetyEnv(yolo\_model=yolo\_model,   
 image\_folder=training\_image\_folder,   
 test\_images=example\_images)  
   
 # Wrap environment for monitoring and vectorization  
 env = DummyVecEnv([lambda: env])  
 env = VecMonitor(env, log\_dir)  
   
 # Create callback for saving best model and plotting  
 callback = SaveOnBestTrainingRewardCallback(check\_freq=1000, log\_dir=log\_dir)  
   
 # Define policy kwargs for custom feature extractor  
 policy\_kwargs = {  
 "features\_extractor\_class": YOLOFeaturesExtractor,  
 "features\_extractor\_kwargs": {  
 "features\_dim": 128,  
 "yolo\_model": yolo\_model  
 }  
 }  
   
 # Initialize PPO model  
 print("Creating PPO model...")  
 model = PPO("MlpPolicy",   
 env,   
 policy\_kwargs=policy\_kwargs,  
 tensorboard\_log=log\_dir,  
 verbose=1,  
 learning\_rate=0.0003)  
   
 # Train model  
 print("Training model...")  
 model.learn(total\_timesteps=100000, callback=callback)  
   
 # Save final model  
 final\_model\_path = os.path.join(model\_dir, "final\_model")  
 model.save(final\_model\_path)  
 print(f"Model saved to {final\_model\_path}")  
   
 # Evaluate model  
 print("Evaluating model...")  
 env.envs[^0].evaluate(model, num\_episodes=10)  
   
 # Test on example images  
 print("Testing on example images...")  
 for img\_path in example\_images:  
 test\_image(model, yolo\_model, img\_path)  
   
 # Plot final training metrics  
 if hasattr(callback, 'episode\_rewards') and callback.episode\_rewards:  
 plt.figure(figsize=(10, 6))  
 plt.plot(callback.episode\_rewards)  
 plt.title('Episode Rewards')  
 plt.xlabel('Episode')  
 plt.ylabel('Reward')  
 plt.grid(True)  
 plt.savefig(os.path.join(log\_dir, 'final\_rewards.png'))  
 plt.show()  
   
 if hasattr(callback, 'training\_losses') and callback.training\_losses:  
 plt.figure(figsize=(10, 6))  
 plt.plot(callback.steps[:len(callback.training\_losses)], callback.training\_losses)  
 plt.title('Training Loss')  
 plt.xlabel('Step')  
 plt.ylabel('Loss')  
 plt.grid(True)  
 plt.savefig(os.path.join(log\_dir, 'final\_loss.png'))  
 plt.show()  
   
 # Close environment  
 env.close()  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

**How the System Works**

This implementation follows the transfer learning approach where a pre-trained YOLO model is used to bootstrap the reinforcement learning process. The system has several key components:

1. **Pre-trained YOLO Model**: The script uses your existing YOLO model that was trained to detect PPE items (hardhat, vest, gloves).
2. **Custom Environment**: The WorkerSafetyEnv implements a standard Gymnasium environment that processes images, runs the YOLO model on them, and determines whether workers are safe based on detected PPE.
3. **Feature Extraction**: The YOLOFeaturesExtractor class takes the output from YOLO and processes it into a feature representation that can be used by the RL algorithm.
4. **PPO Reinforcement Learning**: The system uses the PPO algorithm to learn optimal decisions based on the processed features from YOLO.
5. **Performance Tracking**: The custom callback tracks training metrics (rewards, losses) and saves the best model during training.

The script automatically handles cases where test images or pre-trained models might not be available by providing fallback options.

**Conclusion**

This implementation demonstrates how to effectively combine computer vision (YOLO) with reinforcement learning (PPO) for construction worker safety monitoring. The system learns from the pre-trained YOLO model while allowing the RL agent to adapt its decision-making process over time, potentially improving performance beyond what the object detection system alone could achieve.

When you run this script, it will:

1. Train the PPO model using your pre-trained YOLO detector
2. Generate performance metrics and visualizations
3. Evaluate the model on test images
4. Provide detailed safety assessments based on detected PPE

To use this system with your specific use case, you'll need to:

1. Place your pre-trained YOLO model at "ppe\_detection\_model.pt"
2. Add training images to the "training\_images" folder
3. Place test images in the "test\_images" folder
4. Run the script to train and evaluate the model

⁂