In [1]:

```
import os
import cv2
import numpy as np
import tempfile
from PyQt5.QtCore import QThread, pyqtSignal, QObject
from PyQt5.QtGui import QImage, QPixmap
from PyQt5.QtWidgets import QApplication
from ultralytics import YOLO
import time
import pygame.mixer


class VideoThread(QThread):
        """Thread for capturing and processing video frames."""
        update_frame = pyqtSignal(np.ndarray)
        update_detections = pyqtSignal(list)
        update_fps = pyqtSignal(float)

        def __init__(self, model_path, target_classes, confidence_threshold):
                super().__init__()
                self.model_path = model_path
                self.target_classes = target_classes
                self.confidence_threshold = confidence_threshold
                self.running = False
                self.mode = "webcam"  # or "image"
                self.image = None
def load_model(self):
        try:
                self.model = YOLO(self.model_path)
                return True
        except Exception as e:
                print(f"Error loading model: {str(e)}")
                return False
def update_settings(self, target_classes, confidence_threshold):
        self.target_classes = target_classes
        self.confidence_threshold = confidence_threshold

def set_image(self, image):
        self.image = image
        self.mode = "image"
        if not self.running:
                self.start()

def set_webcam(self):
        self.mode = "webcam"
        if not self.running:
                self.start()

def run(self):
        if not self.load_model():
                return

        self.running = True
```

```
        if self.mode == "webcam":
                cap = cv2.VideoCapture(0)
                cap.set(cv2.CAP_PROP_FRAME_WIDTH, 640)
                cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 480)

                while self.running and self.mode == "webcam":
                        ret, frame = cap.read()
                        if not ret:
                                break

                        start_time = time.time()
                        processed_frame, detections = self.process_frame(frame)
                        fps = 1.0 / (time.time() - start_time)

                        self.update_frame.emit(processed_frame)
                        self.update_detections.emit(detections)
                        self.update_fps.emit(fps)

                        # Short delay to reduce CPU usage
                        time.sleep(0.01)

                if cap.isOpened():
                        cap.release()

        elif self.mode == "image" and self.image is not None:
                start_time = time.time()
                processed_frame, detections = self.process_frame(self.image)
                process_time = time.time() - start_time

                self.update_frame.emit(processed_frame)
                self.update_detections.emit(detections)
                # For images, we'll use FPS signal to show processing time
def process_frame(self, frame):
        """Process a single frame for object detection."""
        # Convert frame to RGB for YOLO
        frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

        start_time = time.time()
        results = self.model(frame_rgb)
        detections = []
        boxes = results[0].boxes if hasattr(results[0], 'boxes') else []

        for box in boxes:
                # Get detection confidence
                conf = float(box.conf[0])
                # Get class name
                cls_id = int(box.cls[0])
                cls_name = self.model.names[cls_id].lower()

                # Filter by confidence and target classes
                if conf > self.confidence_threshold and cls_name in self.target_cla
                        # Get bounding box coordinates
                        x1, y1, x2, y2 = box.xyxy[0].cpu().numpy()
                        x1, y1, x2, y2 = int(x1), int(y1), int(x2), int(y2)

                        detections.append({
```

```
                                'class': cls_name,
                                'confidence': conf,
                                'bbox': (x1, y1, x2, y2)
                        })

        # Draw bounding boxes and labels on the frame
        frame_copy = frame.copy()
        for det in detections:
                x1, y1, x2, y2 = det['bbox']
                cls_name = det['class']
                conf = det['confidence']

                # Assign different colors for different classes
                color_map = {
                        'car': (0, 255, 0),    # Green
                        'cat': (255, 0, 0),    # Blue
                        'person': (0, 0, 255), # Red
                        'dog': (255, 255, 0)   # Cyan
                }

                color = color_map.get(cls_name, (255, 255, 255))

                # Draw bounding box
                cv2.rectangle(frame_copy, (x1, y1), (x2, y2), color, 2)

                # Draw label with background
                label = f"{cls_name.title()}: {conf:.2f}"
                (text_width, text_height), _ = cv2.getTextSize(label, cv2.FONT_HERS
                cv2.rectangle(frame_copy, (x1, y1 - text_height - 10), (x1 + text_w
                cv2.putText(frame_copy, label, (x1, y1 - 5), cv2.FONT_HERSHEY_SIMPL

        return frame_copy, detections



def stop(self):
        self.running = False
        self.wait()


class SoundManager:
        """Handles alert sounds for object detection."""

        def __init__(self, target_classes=None, custom_alert_path=None):
                pygame.mixer.init()
                self.target_classes = target_classes or ["car", "cat", "person", "d
                self.target_classes = [cls.lower() for cls in self.target_classes]
                self.custom_alert_path = custom_alert_path
                self.last_alert_time = {cls: 0 for cls in self.target_classes}
                self.alert_cooldown = 3  # seconds between alerts for the same clas
                self.is_alert_playing = False
                self.temp_dir = tempfile.mkdtemp()
                self.setup_sounds()

        def setup_sounds(self):
                """Set up alert sounds for each target class."""
```

```
                self.sounds = {}

                # If custom alert sound is provided, use it for all classes
                if self.custom_alert_path and os.path.exists(self.custom_alert_path
                        try:
                                # Load the custom alert
                                custom_sound = pygame.mixer.Sound(self.custom_alert
                                for cls in self.target_classes:
                                        self.sounds[cls] = custom_sound

                                print(f"Custom alert sound loaded from: {self.custo
                                return
                        except Exception as e:
                                print(f"Error loading custom alert sound: {str(e)}"
```

```
pygame 2.6.1 (SDL 2.28.4, Python 3.12.1)
Hello from the pygame community. https://www.pygame.org/contribute.html
```