**Problem Overview**

The task is to design a computer vision system capable of detecting objects (e.g., "Person", "Car") and their associated sub-objects (e.g., "Helmet", "Tire") in a hierarchical structure, from a video stream. Additionally, the system should output results in a specified hierarchical **JSON** format and enable image retrieval for sub-objects. The solution must optimize for real-time video processing on CPU.

**Goals and Requirements**

1. **Object and Sub-Object Detection**:
   * Detect objects (e.g., Person, Car) and their sub-objects (e.g., Helmet, Tires).
   * Maintain a hierarchical structure linking sub-objects to their parent objects.
2. **JSON Output Format**:
   * Structure the detection results as JSON with a hierarchical structure, including objects, sub-objects, their bounding boxes, and unique identifiers.
3. **Sub-Object Image Retrieval**:
   * Allow retrieval and saving of cropped images corresponding to detected sub-objects.
4. **Inference Speed Optimization**:
   * Optimize the system to process video frames in real-time (10-30 FPS) on a CPU.
5. **Modularity and Extensibility**:
   * The system should be modular to allow easy extension for additional object-sub-object pairs.

**Approach**

**1. Object and Sub-Object Detection**

The solution involves detecting objects and sub-objects using a deep learning-based object detection model. We can use pre-trained models such as **YOLOv5** or **SSD** for object detection. Here's how the system can be structured:

* **Object Detection**: The object detection model will be trained or fine-tuned on the dataset containing various objects like "Person", "Car", etc. The model will output bounding boxes and class labels for the objects detected.
* **Sub-Object Detection**: This can be done either by:
  1. **Multi-class Detection**: Train a single model to detect both objects and sub-objects (e.g., Helmet as a sub-object of Person).
  2. **Two-Stage Detection**: First, detect the main object (e.g., Person) and then run a second detection model (or fine-tuned classifier) to detect sub-objects (e.g., Helmet within Person's bounding box).
* **Hierarchical Association**: Once objects and sub-objects are detected, they should be uniquely indexed and linked. A sub-object will be associated with its parent object through their unique identifiers.

**2. JSON Output Format**

The output will be in a hierarchical JSON structure, where:

* **"object"**: The primary object’s name (e.g., "Person").
* **"id"**: A unique identifier for each detected object.
* **"bbox"**: Bounding box coordinates [x1, y1, x2, y2].
* **"subobject"**: A nested dictionary for the associated sub-object(s), including its "object": "Car",

**3. Sub-Object Image Retrieval**

For each detected sub-object, the system will:

* Use the bounding box coordinates of the sub-object to crop the image from the original frame.
* Save the cropped image with a unique identifier corresponding to the sub-object (e.g., subobject\_1.png).

**4. Inference Speed Optimization**

Real-time performance is crucial for this system. Optimization strategies include:

* **Model Optimization**: Use efficient models like **YOLOv5 small** (yolov5s) or **MobileNetV2**, which are lightweight and optimized for real-time performance.
* **Resolution Scaling**: Downscale the input resolution for faster processing, especially if high resolution is not required for detection.
* **Batch Processing**: Process multiple frames in parallel (if possible) to improve throughput and overall performance.
* **Inference Framework Optimization**: Leverage frameworks like **ONNX** (Open Neural Network Exchange) or **TensorRT** to accelerate inference on CPU.
* **Multi-threading**: Use multi-threading to concurrently process video frames, making full use of the CPU.

**5. Modularity and Extensibility**

The system should be designed to allow easy addition of new object-sub-object pairs. This can be achieved through:

* **Configuration Files**: Use external configuration files (e.g., JSON or YAML) to define which objects and sub-objects should be detected. This allows easy adaptation for new detection scenarios without modifying the core code.

**Key Concepts and Frameworks Used**

**Deep Learning Models**: The system utilizes deep learning models such as **YOLOv5** for real-time object detection, leveraging PyTorch for model inference.

**Computer Vision Techniques**:

**Bounding Box Detection**: Used to identify the locations of objects and sub-objects in the image.

**Non-Maximum Suppression (NMS)**: Applied to handle overlapping detections and filter redundant bounding boxes.

**Image Cropping**: Used for extracting sub-object images based on bounding box coordinates.

**Optimizations**:

* + Real-time performance on CPU is achieved by using smaller models, such as **YOLOv5 small**, and reducing frame resolution.
  + **Multi-threading** and parallel processing techniques can be applied to improve frame processing speed.

**Assumptions and Limitations**

* **Assumptions**:
  + Objects are large enough and within the model's detection capability.
  + Video streams are of sufficient quality to detect objects accurately.
  + The system is designed to detect a predefined set of objects and sub-objects.
* **Limitations**:
  + Small or heavily occluded objects may not be detected accurately.
  + The system's performance could degrade with very high-resolution videos or highly complex scenes.
  + Detection accuracy is limited by the model's pre-trained capabilities; additional fine-tuning may be required for specialized scenarios.

**Conclusion**

This approach demonstrates the structured thought process to solve the problem. The solution leverages modern deep learning techniques (YOLOv5) for object and sub-object detection, structured data output in JSON format, and real-time video processing capabilities. The system is designed with modularity in mind, ensuring easy expansion for new object-sub-object detection tasks while maintaining performance and accuracy.