CSGY 6613 Artificial Intelligence I
Sarah Chamoun
Spring 2026
Professor Pantelis Monogioudiis
Assignment 2 Paper

This project builds a simple system that can find parts of a car in a YouTube video using a query image. The goal is not to train a new model, but to create a clear and logical pipeline that connects object detection results to specific times in the video. The system works step by step. First, it downloads a public Toyota RAV4 review video from YouTube. Then it extracts frames from the video at fixed time intervals. Next, it runs an object detection model on each frame. All detected objects are saved in a structured Parquet file. Finally, when a query image of a car part is given, the system returns the time ranges in the video where that part appears.

The video was automatically downloaded using yt-dlp in the notebook, which makes the process repeatable and organized. After downloading the video, frames were extracted using ffmpeg. Instead of extracting every frame, the system extracts one every 5 seconds with the setting fps=1/5. This choice greatly reduces the total number of frames that need to be processed. For example, a 46-minute video contains about 165,000 frames because 46 minutes × 60 seconds × 60 frames per second equals approximately 165,000. By sampling one frame every five seconds, this number is reduced to about 552 frames. If frames were extracted every second, the detection process would take much longer and create a much larger index. If frames were extracted less often, short appearances of car parts might be missed. Sampling every five seconds provides a good balance between efficiency and coverage. Each extracted frame is also assigned a timestamp. If a frame has a number called frame_index and frames are sampled every five seconds, then the timestamp is calculated as timestamp_sec = frame_index × 5. This simple rule ensures that every detection can be connected back to the correct second in the original video.

For object detection, a pre-trained YOLO model from the Ultralytics library was used. YOLO was chosen because it is fast, reliable, and easy to use in Python. The confidence threshold was set to 0.25. This means that only detections with at least 25% confidence are saved. A lower threshold keeps more detections but may include mistakes. A higher threshold removes more mistakes but may miss real objects. YOLO stands for “You Only Look Once.” It is an object detection model that looks at an image, draws bounding boxes around objects, and identifies each object.

For each frame, all objects that are detected with confidence above the set threshold are saved in a Parquet file. Each row in this file includes the video ID, the frame number, the time in seconds, the detected object label, the bounding box location, and the confidence score. The Parquet file serves as an index of everything in the video. It allows us to filter by object type and sort the results by time. Parquet was chosen because it is fast and efficient to store and load. The query images were loaded from the Hugging Face dataset aegean-ai/rav4-exterior-images. These images show different outside views of the same car model. When a query image is given, the same YOLO model is used to detect objects in it. The object with the highest confidence score is selected as the main label for retrieval.

For example, if the model detects “wheel” in the query image, the system searches the detection index for all rows where the class label is “wheel.” This ensures that both the video frames and the query images use the same detection logic. However, returning individual frames is not very useful. Instead, the system groups detections into time segments. After filtering by class label, detections are sorted by timestamp. If two detections happen within two seconds of each other, they are considered part of the same segment. If the timestamp variance exceeds 2 seconds, a new segment is created.

Each segment stores the start time, end time, object label, and the number of supporting detections. This grouping step turns single-frame detections into useful video clips. Several query images were tested. In many cases, the model correctly retrieved the time ranges where the correct car part appears. These results were checked by opening YouTube links and recording their start and end times. The results were mostly accurate for parts that stay visible for several seconds, such as wheels or headlights.

There are some limitations to this approach. First, the object detector is not perfect. It can confuse similar car parts, such as bumpers and body panels. If the detector makes a mistake, the retrieval results will be incorrect as well. Second, frames are sampled every five seconds, so short appearances of car parts may be missed. Sampling frames more often would improve accuracy, but it would also increase computation time. Third, some query images show multiple car parts. The system only uses the label with the highest confidence score, potentially ignoring other important objects. Finally, differences in lighting and camera angles between the video and the query images can lower detection accuracy.

Even with these limitations, the system shows how object detection can be used to build a searchable video index. By combining video download, frame extraction, object detection, structured data storage, and time grouping, the system can find video segments using only an image as input. In the future, the system could be improved by fine-tuning the detector on a car parts dataset, sampling frames more frequently, using multiple labels per query image, or using visual features rather than only class labels. These improvements could increase accuracy and make the system more reliable.

This project demonstrates a clear, organized way to connect images to video clips. The pipeline is simple, repeatable, and easy to follow. The system depends on how accurate the object detector is and how often frames are sampled, but it can connect detected objects to specific time ranges in the video.