End-to-End Object Segmentation and Tracking with Labeller

Executive summary

This project delivers an end-to-end workflow for vehicle and pedestrian segmentation and multi-object tracking, covering data collection, AI-assisted labeling, YOLO-Seg fine-tuning, evaluation, review in Labeller, and a Streamlit demo app that runs tracking and exports the tracked video and it's results as JSON. The pipeline was implemented on Colab for training and on Streamlit Cloud for deployment, emphasizing reproducibility, practical debugging, and compliant licensing via a root-level sources.md.

Problem and goals

The objective was to build a compact but realistic workflow that learns segmentation for vehicles and pedestrians from a custom image set and then tracks those categories in video with persistent IDs. The target experience includes a small dataset curated from permissibly licensed images, AI-assisted annotation to accelerate labeling, a trained YOLO-Seg model, and a web demo that lets a user upload a clip, run tracking, download outputs, and obtain results.json.

Data collection

Directly downloadable images were required to avoid pre-labeled datasets and video-only sources common in traffic-light feeds. Approximately 150 images were collected from web search using the Unsplash developer API with focused queries such as "busy crosswalk" and "city street pedestrians," saving files by photo ID for later attribution and metadata recovery.

Labeling workflow

A project was created in Labellerr using the UI to start labeling while SDK credentials (Client_Id) was pending; objects were defined as vehicles and pedestrians with polygon outlines. Manual polygon labeling proved slow, so a second Labellerr project was created using the platform's AI-assisted click-to-outline feature, which substantially accelerated

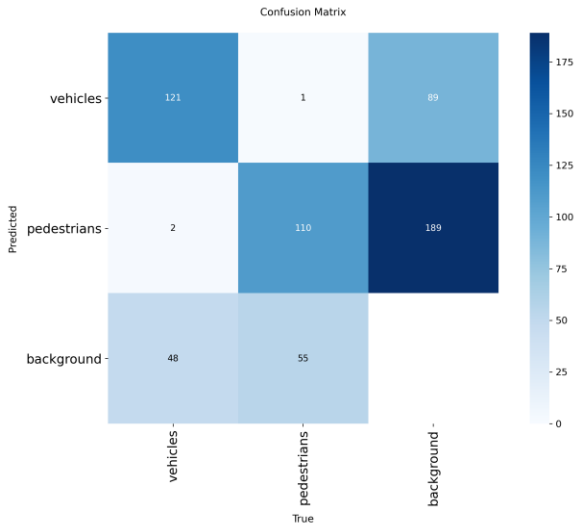
annotation throughput. Roughly 100 images were labeled for training and about 40 images were labeled for validation using the AI-assisted workflow.

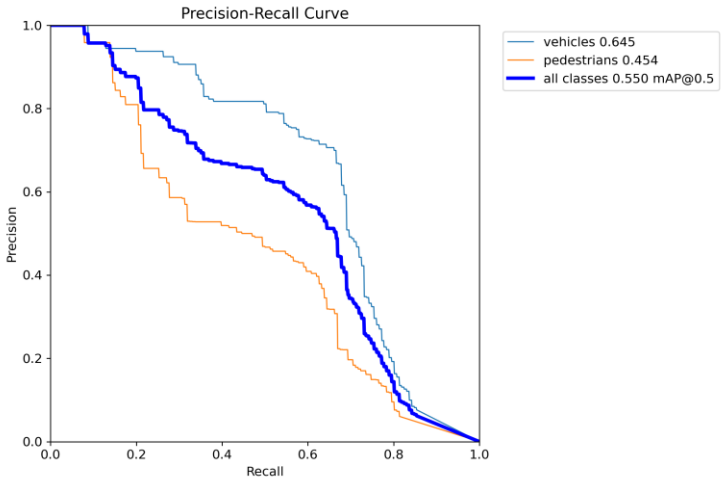
Dataset splits and export

The working dataset comprised ~150 collected images, with ~100 assigned to train and ~40 to validation after labeling; any remaining images could be reserved for exploratory checks. Labeled data were exported from Labeller in YOLOv11 format suitable for YOLO-Seg training without additional conversion.

Model training and Evaluation

A YOLO-Seg model was fine-tuned on the labeled dataset to learn polygon-level masks for pedestrians and vehicles. Training was run on Colab and completed in roughly 15 minutes for the selected lightweight configuration, making it feasible to iterate on splits or parameters if needed. Resulting artifacts include trained weights, logs, and curves.



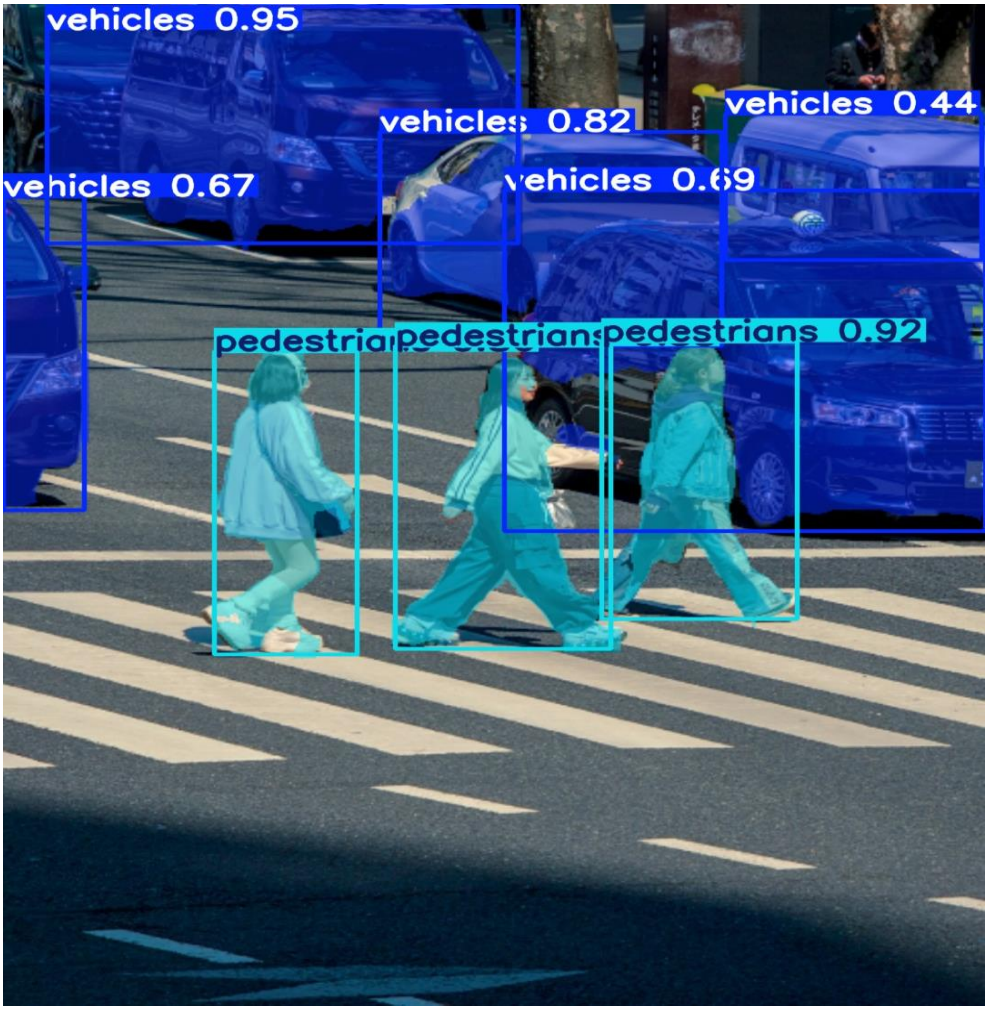


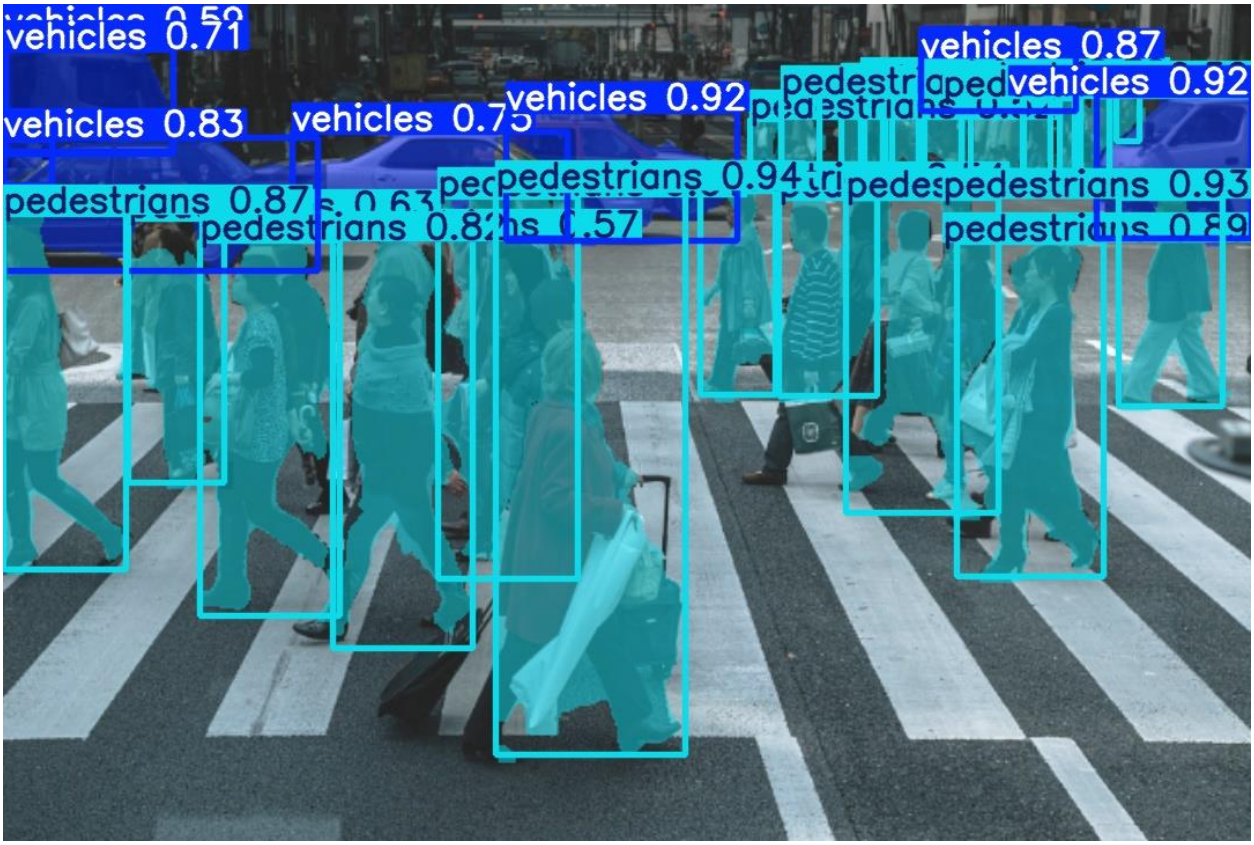
Model-Metrics

mAP50: 0.5498994976107534
mAP50-95: 0.34375703322322826
Precision: 0.5611873725221825
Recall: 0.6164658634538153
F1-Score: 0.5875292447603246

Test inference

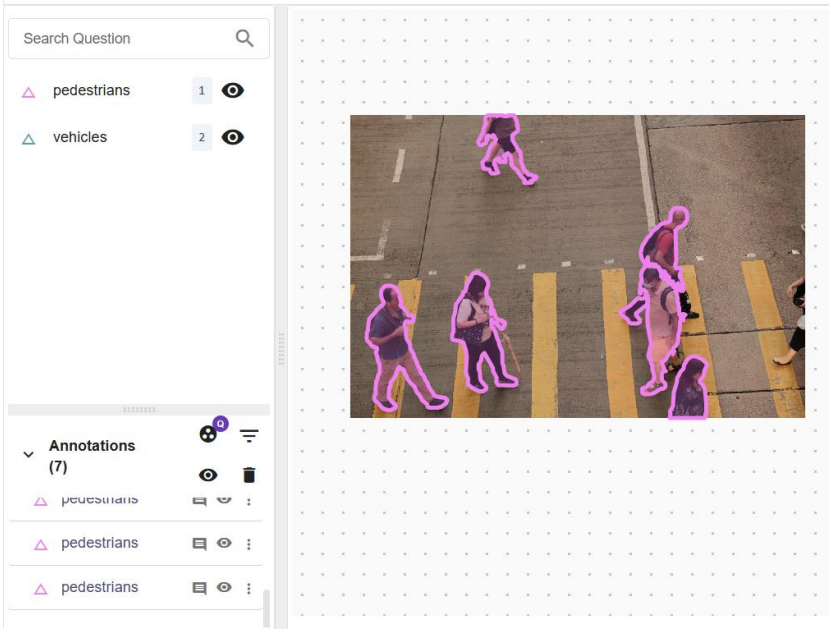
A separate batch of ~40 Unsplash images was collected and kept outside the training and validation sets for qualitative and quick quantitative checks. The trained model produced consistent predictions across these held-out images.

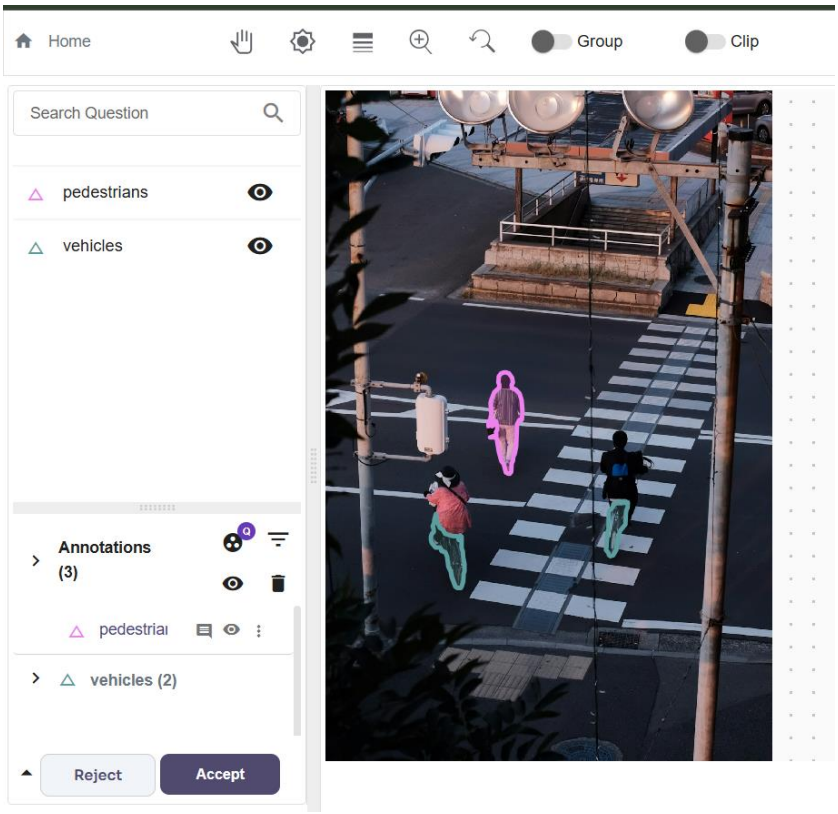

vehicles 0.95
vehicles 0.82
vehicles 0.44
vehicles 0.67
vehicles 0.69
pedestrians 0.92



Pre-annotation round-trip via SDK

After receiving client credentials, test predictions were prepared for Labeller review according to the guideline requiring pre-annotations in a test project. Pre-annotations were uploaded using Labeller's sdk. The asynchronous upload path returned an internal server error during trial, so the synchronous upload method was used, which completed successfully. In the Labellerr UI, test files in Unlabelled status displayed the model's suggested masks for visual verification.


Search Question
pedestrians
1
vehicles
2
Annotations
(7)
pedestrians
pedestrians

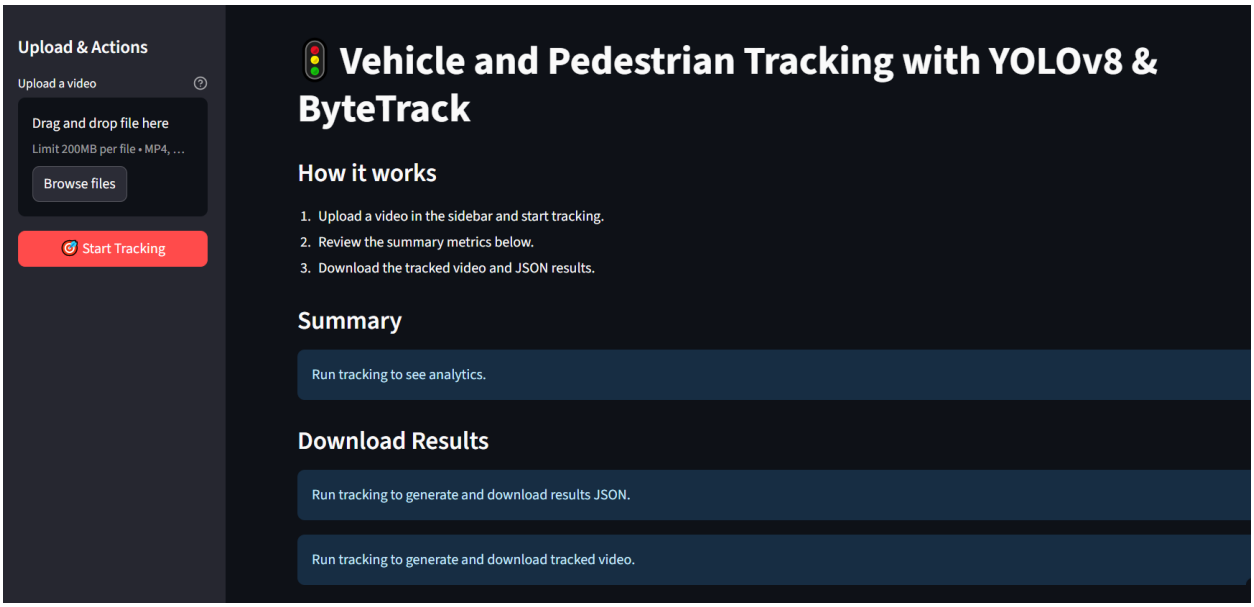

Home
Group
Clip
Search Question
pedestrians
vehicles
Annotations
(3)
vehicles (2)
Reject
Accept

Tracking pipeline and JSON export

For video tracking, the YOLO built-in tracking mode was used with ByteTrack to obtain consistent IDs across frames. The demo pipeline assigns track IDs to predicted objects frame-by-frame and exports a results.json that records id, class, confidence, bounding box, and frame index for downstream use. This approach avoided custom bindings and reduced integration errors while satisfying the requirement to track trained classes.

Web application

A Streamlit application was built to let a user upload a video, run tracking with the trained YOLO-Seg model and ByteTrack, and then download both the processed video and results.json. A preview feature was attempted but removed after encountering an H.264/codec rendering issue in the hosted environment; the app keeps a clear focus on upload, process, and download to preserve reliability. The app was deployed on Streamlit Cloud.



Deployment and dependencies

Initial deployment failed to import OpenCV and Ultralytics, which was resolved by adding requirements.txt (and, if needed, an additional packages file) at the project root. Once

dependencies were pinned and the environment rebuilt, the application deployed and ran successfully end-to-end. This configuration will be included verbatim to ensure one-click reproducibility on the hosting platform.

Issues debugged

- Manual labeling throughput: mitigated by adopting Labellerr's AI-assisted click-to-outline workflow and recreating the project with the same dataset.
- SDK integration: asynchronous pre-annotation upload returned a server error; switching to synchronous upload completed successfully, enabling the UI review step.
- Video preview codec: an H.264 rendering error in the hosted Streamlit environment led to removal of in-app playback to prioritize stable file download and inspection.
- Dependency resolution: missing cv2 and ultralytics on first deploy were fixed by explicit requirements and a clean rebuild.

Guide: build an object tracker with Labeller

- Define scope: list object classes, intended scenes, target metrics, and a simple success criterion.
- Collect data: source permissibly licensed images,avoid pre-labeled sets.
- Create train project: upload images to Labeller, define classes, and use polygon masks with AI-assisted outlining for speed and consistency.
- Labeling quality: set brief rules for edges/occlusions, review a small sample, and standardize class names before exporting.
- Export labels: choose a YOLO-Seg–compatible format from Labeller; keep a clear split of train, val, and test with non-overlapping files.
- Train baseline: fine-tune a lightweight YOLO-Seg model in a notebook; log key metrics and save weights and curves for the report.
- Validate on hold-out: run inference on a separate test set; capture qualitative examples and standard metrics.
- Close the loop: create a Labeller test project, upload predictions as pre-annotations via the SDK, and verify them in the UI.

- Tracking step: pair the trained model with a simple tracker (e.g., ByteTrack) to assign persistent IDs and export a results.json per video.
- Minimal app: provide a small UI to upload a video, run tracking, and download the processed video and results.json; prefer reliability over inline playback if codecs are limited.
- Pitfalls to anticipate: rate limits in collection, slow manual labeling, SDK upload modes, and deployment dependencies; adopt small, testable increments.

Final summary

- Built an end-to-end pipeline: curated data, AI-assisted labeling in Labellerr, YOLO-Seg training, evaluation on a held-out set, SDK-based pre-annotation review, and a demo app with tracking and JSON export.
- Prioritized pragmatism: small model for fast iteration, clear splits for hygiene, and simple UI/UX for dependable upload-process-download flows.
- Resolved common hurdles: accelerated labeling with AI assist, switched SDK modes when needed, and pinned dependencies for a clean deploy.