Alma Mater Studiorum – Università di Bologna

School of Engineering and Architecture

Master's Degree in Artificial Intelligence

Machine Learning for Computer Vision (ML4CV)

3 CFU Project

Improving Real-Time Cone Detection for Autonomous Formula SAE Racing with Modern YOLO Architectures

Student:

Nicolas Cridlig

Supervisor:

Prof. Samuele Salti

Academic Year 2024–2025

February 2026

Abstract

Autonomous Formula SAE race cars must detect colored traffic cones in real time to navigate an unknown track. We improve the cone detection pipeline for the Unibo Motorsport race car by evaluating three YOLO architectures, YOLO11n, YOLO12n, and YOLO26n, on the FSOCO-12 benchmark under identical training conditions. YOLO26n achieves mAP_{50}=0.763, a 14.6% improvement over the YOLO11n model previously deployed on the car (0.666), while being smaller and faster. A Bayesian hyperparameter sweep over 13 parameters found no improvement over the Ultralytics defaults, confirming that architecture selection dominates tuning for this task. A two-stage training strategy, pre-training on a larger 22,725-image dataset before fine-tuning on FSOCO-12, matched the single-stage benchmark but improved recall by 1.4 percentage points. To validate beyond internet benchmarks, we created fsoco-ubm, a 96-image test set from the car's own stereo camera, which revealed a 22–27% accuracy drop across all models. The selected model was deployed via TensorRT FP16 on the onboard RTX 4060. Integrating YOLO26n required modifying the C++ inference node to handle its end-to-end output format, which eliminated NMS postprocessing entirely. The full perception pipeline runs at 15.6 ms, within the 16.7 ms budget for 60 fps operation.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Context: Formula Student Driverless

This report addresses an element of the work undertaken by Unibo Motorsport (UBM) to succeed in the Formula SAE (FSAE) competition. Since 2017, the competition requires cars to autonomously navigate an unknown track delineated by colored traffic cones at maximum speed. To do so, the software and hardware stack must be efficient, reliable, and well integrated. The perception of the colored cones is vital due to the garbage-in, garbage-out moniker: even the best written code cannot produce good results on bad data. Therefore, in this report we increase successful cone detections by utilizing the newest generation of YOLO models alongside a new dataset. This research is vital, last summer, UBM competed at Formula Student Germany, and this year we will compete again from the 11-16th of August 2026.

1.2 Problem Statement

For perception, the software stack relies upon a custom trained YOLO object detector to identify track cones from a stereocamera so that they may be reconstructed with a multi-stage matching algorithm. The baseline model, YOLO11, has room for improvement. It is trained on base hyperparameters and any tuning or analysis was lost. The mAP_{50}=0.666 and while this figure is our quantitative benchmark, qualitative issues registered by the team include exposure sensitivity and false positives. The model is constrained by both accuracy and real time speed. The stereocamera operates at 60fps which allows for a maximum control loop time of 16.7 ms. Any time perception takes out of this hard limit, is less time for the decision and control algorithms to operate.

Figure 1.1: Example false positive: a person misclassified as a yellow cone by the baseline YOLO11n detector.

1.3 Objectives

The project objectives are:

1. Reproduce and establish a reproducible baseline
2. Evaluate more modern YOLO architectures (YOLO12n, YOLO26n)
3. Explore two-stage training (pre-training on larger dataset + fine-tuning)
4. Assess hyperparameter optimization via Bayesian sweep
5. Deploy optimized model on RTX 4060 via TensorRT
6. Create real-world validation dataset (fsoco-ubm)

1.4 Contributions

We evaluated three YOLO architectures and found that YOLO26n, the most recent and lightest, reaches mAP_{50}=0.763 on the FSOCO-12 benchmark—a 14.6% improvement over the model currently running on the car—while achieving 2.63 ms inference on the onboard RTX 4060. We also created fsoco-ubm, a real-world test set from the car's own camera, which revealed a 22–27% accuracy gap between internet benchmarks and deployment conditions across all models tested.

Chapter 2

Background

2.1 Object Detection with YOLO

YOLO (You Only Look Once) is a family of real-time object detection models first introduced by Redmon et al. (2016) and now maintained by Ultralytics (Jocher et al., 2023). Unlike two-stage detectors that first propose regions and then classify them, YOLO performs detection in a single forward pass, which is what makes it fast enough for our use case. We evaluated three successive generations of nano-sized YOLO models.

2.1.1 YOLO11

YOLO11 was released in 2024 by Ultralytics (Ultralytics, 2024) and is the architecture currently deployed on the UBM race car. It introduces C3k2 backbone blocks, a Spatial Pyramid Pooling Fast (SPPF) layer, and C2PSA attention modules. The nano variant has 2.59M parameters and 6.4 GFLOPs. This is our baseline: every other model in this report is measured against it.

2.1.2 YOLO12

YOLO12, published in January 2025 by Tian et al. (2025), replaces the CSP backbone with R-ELAN (Residual Efficient Layer Aggregation Network) and introduces an area-attention mechanism that captures global context without the quadratic cost of standard self-attention. The nano variant sits at 2.56M parameters and 6.3 GFLOPs, marginally smaller than YOLO11n.

2.1.3 YOLO26

YOLO26 is the most recent release from Ultralytics at the time of writing (Sapkota et al., 2026). Its nano variant is the smallest of the three we evaluated: 2.51M parameters and

5.8 GFLOPs. It also eliminates the non-maximum suppression (NMS) post-processing step, further reducing latency.

2.2 Transfer Learning and Catastrophic Forgetting

Transfer learning is the practice of pre-training a model on a large dataset and then fine-tuning it on a smaller, task-specific one (FSB Driverless, 2024; FMDV, 2024). The main risk is catastrophic forgetting (French, 1999): if the learning rate is too high during fine-tuning, the model overwrites the useful features it learned during pre-training. Standard mitigation strategies include freezing early layers, using a low learning rate, and warming up gradually. We encountered this problem during our two-stage training and document the diagnosis and fix in sections A.3 and 3.6.

2.3 Model Optimization for Edge Deployment

The models are trained in PyTorch but deployed through a multi-stage optimization pipeline. First, the trained weights are exported to ONNX (Open Neural Network Exchange) (ONNX Community, 2019), a hardware-agnostic intermediate representation. Then, NVIDIA TensorRT (NVIDIA Corporation, 2024) compiles the ONNX graph into an optimized engine for the target GPU. TensorRT applies several automatic optimizations: kernel fusion combines sequences of operations (e.g. Conv+BatchNorm+ReLU) into a single GPU kernel to reduce memory bandwidth and kernel launch overhead, and precision reduction converts FP32 weights to FP16 or INT8 to exploit the GPU's dedicated Tensor Cores. INT8 quantization requires a calibration dataset of 500–1,000 representative images to determine the optimal scaling factors, and typically incurs a 2–3% mAP_{50} drop compared to FP16.

One practical constraint is that TensorRT selects kernels and memory layouts optimized for the specific GPU it builds on. While GPUs within the same generation (e.g. RTX 4080 Super and RTX 4060, both Ada Lovelace) share compute capability 8.9, an engine built on one cannot run on the other, since only building on the target hardware ensures optimal kernel selection. We therefore built the final engine directly on the race car's RTX 4060.

2.4 Formula Student Driverless Perception

The perception pipeline, as designed by Fusa (2025), is built around a ZED 2i stereo camera (Stereolabs, 2021) operating at 1280×720 resolution and 30 fps but targeting 60 fps in the future. The YOLO detector runs on each frame to produce 2D bounding boxes for five cone classes defined by the FSAE Driverless rules (SAE International, 2025): small

blue cones mark the left track border, small yellow cones mark the right border, small orange cones delineate entry and exit lanes, and large orange cones are placed before and after start, finish, and timekeeping lines. A fifth class, "unknown," is used for ambiguous or occluded cones that cannot be confidently classified. These detections are then matched across the stereo pair and triangulated to produce 3D positions relative to the car. Any excess time taken by detection directly reduces the budget available for planning and control downstream.

2.5 Related Work

The FSOCO (Formula Student Objects in Context) dataset (FSOCO Contributors, 2020) is a community-maintained benchmark for cone detection, contributed by multiple Formula Student teams. It provides a shared evaluation standard, though the images come from a variety of cameras and conditions that may not match any single team's deployment setup. Most teams in the Driverless competition use some variant of YOLO, trained on FSOCO or private datasets, and evaluate using the standard COCO metrics (Lin et al., 2014): mAP_{50} (intersection-over-union threshold of 0.5) and $\text{mAP}_{50\text{-}95}$ (averaged over thresholds from 0.5 to 0.95). The prior work at UBM is documented in the thesis by Fusa (2025), which established the YOLO11n baseline and the stereo matching pipeline we build upon.

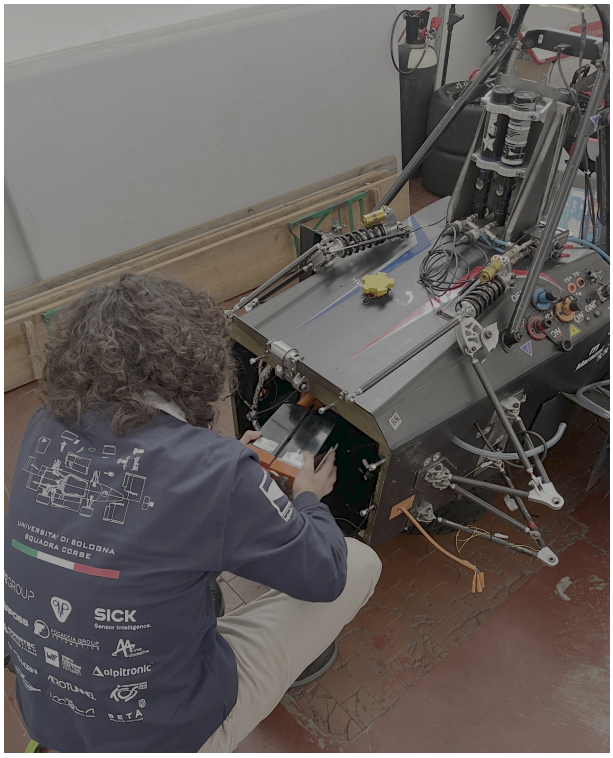

Figure 2.1: The rear of the racecar inside the UBM Workshop while removing the 12v battery which powers the computers.

Chapter 3

Methods

3.1 Experimental Setup

All training scripts, configurations, and evaluation code are publicly available (Cridlig, 2026). We used a M1 MacBook Air for code development and an Ubuntu workstation with a RTX 4080 Super for training. The existing inference pipeline (`ubm-yolo-detector`) enforced the use of Ultralytics models, which we kept to slot the new model into the existing modular framework. Weights and Biases (Weights & Biases, Inc., 2024) provided experiment tracking. The Autonomous System Unit (ASU) on the race car is a water-cooled desktop running ROS2 (Fusa, 2025); for our purposes we care only about its GPU, a RTX 4060.

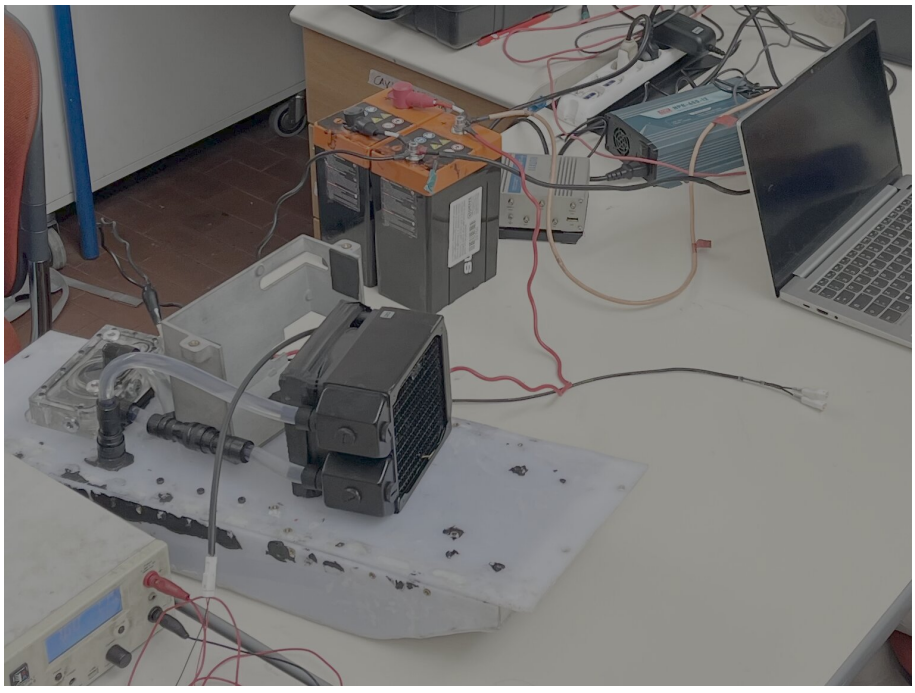

Figure 3.1: The Autonomous System Unit (white box) benchmarking YOLO models.

Table 3.1: Hardware used for training and deployment.

Machine	Purpose	GPU	OS
M1 MacBook	Code development	CPU only	macOS
Ubuntu Workstation	Training & testing	RTX 4080 Super (CUDA)	Ubuntu
ASU (Race Car)	Deployment	RTX 4060 (CUDA)	Ubuntu (ROS2)

3.2 Datasets

We used three datasets, summarized in table 3.2.

Table 3.2: Datasets used in this project.

Dataset	Images	Instances	Split	Purpose
FSOCO-12 (train)	7,120	∼78,000	Train	Primary training
FSOCO-12 (val)	1,968	∼36,000	Validation	Model selection
FSOCO-12 (test)	689	12,054	Test	Standard benchmark
cone-detector	22,725	∼200,000	Train	Stage 1 pre-training
fsoco-ubm	96	1,426	Test	Real-world validation

3.2.1 FSOCO-12

FSOCO-12 (Formula Student Objects in Context, version 12) is a community-curated dataset hosted on Roboflow (FMDV, 2024; FSOCO Contributors, 2020). It contains 9,777 images split into train (7,120), validation (1,968), and test (689) sets, with approximately 126,000 cone instances across the five FSAE classes. The images come from multiple Formula Student teams and cover diverse lighting, weather, and track conditions. The class distribution is imbalanced: yellow and blue cones dominate (∼77% of test instances), while unknown cones account for only 5.6% and are the hardest to detect. We use the test set (689 images, 12,054 instances) as our primary benchmark throughout this report.

3.2.2 cone-detector

The cone-detector dataset (FSB Driverless, 2024) contains 22,725 images with the same five cone classes as FSOCO-12, making it 2.3× larger. It is also a community dataset from Roboflow, contributed by a different Formula Student team. Training a model from scratch on this dataset alone plateaus at mAP_{50}≈0.68, which is worse than our FSOCO-12 baseline. However, the volume of data makes it useful for pre-training before fine-tuning on the curated FSOCO-12 set (section 3.6).

(a) blue (b) yellow (c) small orange (d) large orange

Figure 2: FSOCO supports five object classes. The four main classes are shown here, the fifth class *other* includes all cones that are not rules compliant.

Figure 3.2: Class distribution in the FSOCO-12 dataset.

3.2.3 fsoco-ubm: Real-World Test Set

FSOCO-12 is an internet dataset assembled from many teams and cameras, so good performance on it does not guarantee good performance on our car. To address this, we created fsoco-ubm: a 96-image test set extracted directly from the ZED 2i stereo camera on the UBM race car. The images were recorded on November 20, 2025 at the Rioveggio test track during driving runs at 30–50 km/h. We extracted one frame every 60 frames from the stereo video (one sample every two seconds of real-world time), split the 2560×720 stereo pairs into left and right images of 1280×720, and annotated them using Roboflow Label Assist with manual review.

The dataset contains 1,426 cone instances across the five classes, dominated by yellow and blue cones with few orange instances. The real-world conditions introduce challenges absent from the internet benchmark: motion blur, variable outdoor lighting, small pixel area for distant cones, and color desaturation. We use fsoco-ubm strictly as a held-out test set and never for training.

3.3 Baseline Reproduction

Our first step was to reproduce the baseline from Fusa's thesis (Fusa, 2025), which reports mAP_{50}=0.824 for YOLO11n on FSOCO. We trained YOLO11n for 300 epochs on FSOCO-12 using Ultralytics default hyperparameters (AdamW optimizer, lr0=0.01, batch 64, 640×640 input, default augmentation with mosaic=1.0, mixup=0.0). The model converged around epoch 200 and plateaued at mAP_{50}=0.714 on the validation set, 13.4% below the thesis claim.

We contacted the thesis author, who confirmed that the reported results were produced by other team members (Gabriele and Patta) using an unknown "particular dataset" and

training configuration that was subsequently lost. We then evaluated the production weights currently deployed on the car—a separate YOLO11n checkpoint trained with that lost configuration—on the FSOCO-12 test set: it achieves only mAP_{50}=0.666. Our retrained model scores mAP_{50}=0.707 on the same test set, already 6.2% above production, and we adopt this as the reproducible baseline for all subsequent experiments.

3.4 Hyperparameter Sweep

To determine whether the baseline could be improved through hyperparameter tuning, we ran a Bayesian optimization sweep using Weights & Biases (Weights & Biases, Inc., 2024; Snoek et al., 2012). The sweep explored 13 hyperparameters: learning rate (lr0, lrf), momentum, weight decay, warmup epochs, close mosaic epoch, dropout, and six augmentation parameters (hsv_h, hsv_s, hsv_v, mosaic, mixup, copy_paste). Each run trained for 100 epochs on FSOCO-12 with YOLO11n.

Of 21 planned runs, 10 completed successfully and 11 crashed. The crashes were strongly correlated with aggressive augmentation: crashed runs had 4× higher mixup (0.197 vs 0.049) and nearly 2× higher dropout (0.156 vs 0.081) on average. High mixup creates heavily blended training images that, combined with high dropout, introduce too much regularization for the model to converge. Among the 10 completed runs, the best achieved mAP_{50}=0.709, 0.7% *worse* than our baseline (table 3.3). The mean was 0.703 with a standard deviation of 0.019, confirming that YOLO11n on FSOCO-12 is insensitive to these hyperparameters. We stopped the sweep and pivoted to architecture changes.

Table 3.3: Hyperparameter sweep summary (W&B Bayesian optimization).

Metric	**Value**
Runs completed	10 / 21
Best sweep mAP_{50}	0.709
Baseline mAP_{50}	0.714
Mean of sweep runs	0.703
Standard deviation	0.019

3.5 Architecture Evaluation

Since hyperparameter tuning proved ineffective, we evaluated whether newer architectures could break through the YOLO11n performance ceiling. We trained YOLO12n (Tian et al., 2025) and YOLO26n (Sapkota et al., 2026) under identical conditions: FSOCO-12 dataset, 300 epochs, batch 64, Ultralytics default hyperparameters. All three models are nano variants with comparable parameter counts (2.51–2.59M) and GFLOPs (5.8–6.4), as shown

in table 3.4, making the comparison fair with respect to model capacity. YOLO12n finished at mAP_{50}=0.708 on the test set, a marginal gain over YOLO11n (0.707). YOLO26n reached mAP_{50}=0.763, a substantial 7.9% improvement over our baseline and 14.6% over the UBM production model. Every class improved, with the largest absolute gains on unknown cones (+23.4%) and orange cones (+6.8%).

Table 3.4: Architecture specifications for the three evaluated models.

Model	Params (M)	GFLOPs	Year
YOLO11n	2.59	6.4	2024
YOLO12n	2.56	6.3	2025
YOLO26n	2.51	5.8	2026

3.6 Two-Stage Training Strategy

Given that YOLO26n achieved the best single-stage result, we investigated whether pre-training on a larger dataset could push it further. The cone-detector dataset (FSB Driverless, 2024) contains 22,725 images with the same five classes as FSOCO-12, providing 2.3× more training data. Both datasets target the same task—cone detection—so the domain gap is small.

Stage 1 pre-trained YOLO26n on cone-detector starting from COCO-pretrained weights. Training was planned for 400 epochs but the loss converged early and we stopped at epoch 338, achieving mAP_{50}=0.734 on the cone-detector validation set. Stage 2 fine-tuned the Stage 1 checkpoint on FSOCO-12 for 300 additional epochs. Our first attempt at Stage 2 failed: the model's mAP_{50} dropped from 0.754 to 0.628 in just two epochs—catastrophic forgetting (French, 1999). The root cause was Ultralytics' `optimizer='auto'` default, which silently replaced our specified learning rate of 0.001 with a hardcoded 0.01, as detailed in section A.3.

We redesigned Stage 2 as a two-phase process using explicit AdamW (Loshchilov and Hutter, 2019). Phase 2A froze the first 10 backbone layers and trained only the detection head for 50 epochs with lr=0.001 and 20% warmup (10 epochs). Phase 2B then unfroze all layers for 250 epochs with an ultra-low learning rate of 0.00005 (100× lower than training from scratch) and 50 epochs of warmup with cosine annealing. This eliminated the forgetting entirely: validation mAP_{50} improved monotonically throughout Stage 2 and converged at 0.761 on the FSOCO-12 test set, comparable to the single-stage result (0.763) but with better recall and real-world precision, as we discuss in chapter 4.

3.7 Deployment Pipeline

The trained PyTorch model is deployed through a three-step export pipeline. First, the `best.pt` checkpoint is exported to ONNX (ONNX Community, 2019) with a fixed batch size of 2 (two images in a stereo pair). Then, on the ASU itself, the ONNX graph is compiled into a TensorRT FP16 engine (NVIDIA Corporation, 2024) using `trtexec`. Building the engine on the target hardware is mandatory because TensorRT optimizes kernel selection and memory layout for the specific GPU architecture (RTX 4060).

The C++ ROS2 inference node (`ros_yolo_detector_node`) receives stereo image pairs from the ZED 2i camera topic and runs YOLO inference on the left and right frames in parallel using a batch. The resulting bounding box vectors are sorted by center coordinates and matched across the stereo pair using a distance and similarity criterion, followed by feature matching for robust association. Matched detections are triangulated into 3D positions in the camera coordinate frame and then transformed into the vehicle frame (FLU convention) for downstream SLAM consumption. The node publishes both an annotated image with bounding box overlays and a `DetectionInfosArray` message containing 3D coordinates and confidence scores. We lowered the detection confidence threshold from the default 0.5 to 0.35, which improved mAP_{50} on fsoco-ubm by 10% by recovering additional true positives at the cost of a marginal increase in false detections. This parameter as well as the NMS threshold are modifiable without recompiling and can be tuned further on the track.

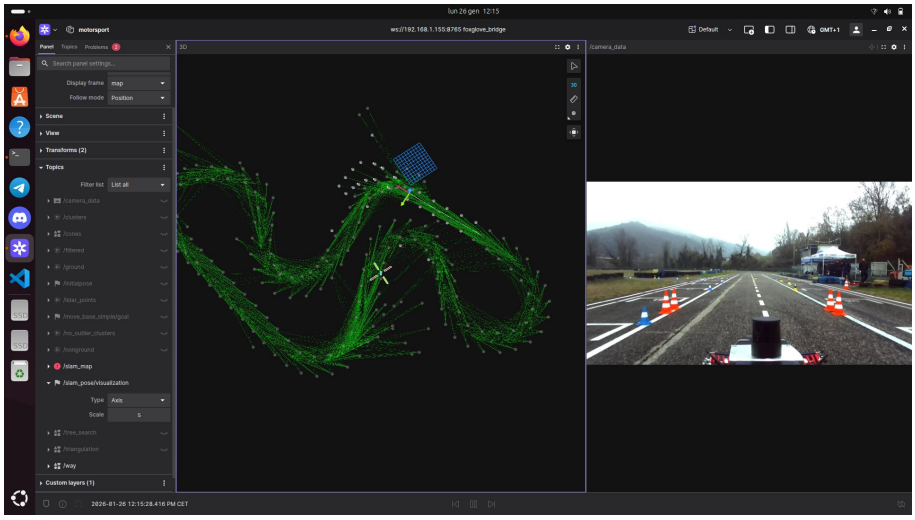

Figure 3.3: 3D visualization of detected cones from the stereo camera and lidar pipelines.

Chapter 4

Results

We evaluate all models on two test sets: the FSOCO-12 benchmark (689 images) and fsoco-ubm (96 images from the car's own camera), reporting standard COCO metrics (Lin et al., 2014).

4.1 FSOCO-12 Benchmark Results

Table 4.1 presents the FSOCO-12 test set results. YOLO26n dominates: the single-stage variant achieves the highest mAP_{50}=0.763 and precision (0.849), while the two-stage variant leads in recall (0.708) and $mAP_{50\text{-}95}$=0.528. Both YOLO26n variants outperform the UBM production model by over 14%, confirming that the architecture upgrade accounts for the bulk of the improvement. YOLO12n and our retrained YOLO11n baseline land nearly identically at mAP_{50}≈0.708, both already 6% above the production model.

Table 4.1: Overall performance on the FSOCO-12 test set (689 images, 12,054 instances).

Model	**Training Data**	**Epochs**	**mAP_{50}**	**$mAP_{50\text{-}95}$**	**Prec.**	**Recall**
YOLO26n (single)	FSOCO-12	300	**0.763**	0.524	**0.849**	0.694
YOLO26n (two-stage)	CD→FSOCO-12	338+300	0.761	**0.528**	0.832	**0.708**
YOLO12n	FSOCO-12	300	0.708	0.485	0.840	0.654
YOLO11n (ours)	FSOCO-12	300	0.707	0.490	0.816	0.662
UBM Production	unknown	300	0.666	0.461	0.803	0.579

CD = cone-detector dataset (22,725 images, pre-training stage).

4.2 Per-Class Analysis

Table 4.2 shows a consistent per-class hierarchy. Large orange cones are easiest (mAP_{50}=0.886), followed by blue (0.863), yellow (0.856), and orange (0.843). Unknown cones remain the

hardest class at mAP_{50}=0.364—these are cones that annotators could not confidently classify, so the model inherits that ambiguity. YOLO26n improves every class over YOLO12n, with the largest gain on unknown cones (+23.4%), likely because it is more willing to commit to a class label on distant cones that occupy as few as 20×20 pixels (fig. 4.1).

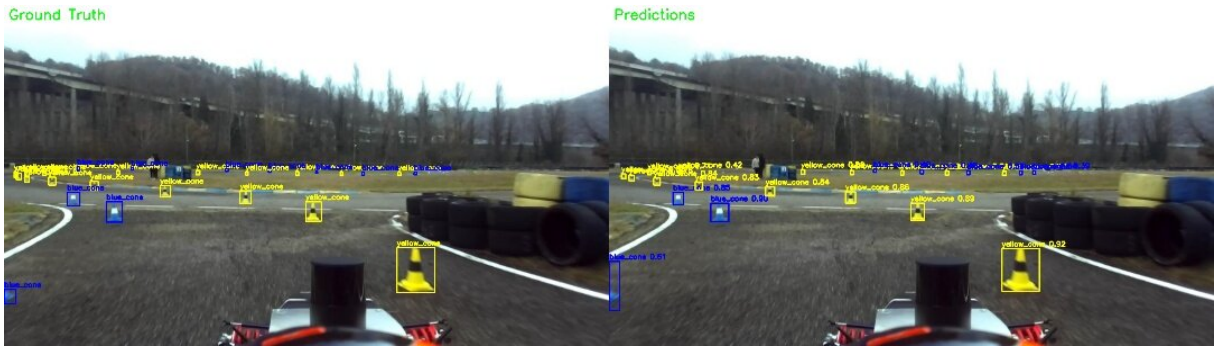

Figure 4.1: YOLO26n (two-stage) detections on a fsoco-ubm image. Distant cones spanning only tens of pixels are detected, though not all.

Table 4.2: Per-class performance of YOLO26n (single-stage) on FSOCO-12 test set.

Class	**Images**	**Instances**	**Prec.**	**Recall**	**mAP_{50}**	**$mAP_{50\text{-}95}$**
Large Orange Cone	154	408	0.873	0.833	**0.886**	0.688
Blue Cone	506	4,437	0.927	0.783	0.863	0.602
Yellow Cone	562	4,844	0.915	0.774	0.856	0.583
Orange Cone	286	1,686	0.892	0.779	0.843	0.571
Unknown Cone	68	679	0.635	0.297	0.364	0.178

4.3 Two-Stage vs. Single-Stage Training

Table 4.3 compares the two training strategies head-to-head. On the FSOCO-12 benchmark the difference is within noise: mAP_{50} differs by just 0.2%, and neither variant is consistently better across all metrics. The two-stage model wins on recall (+1.4 percentage points) and $mAP_{50\text{-}95}$ (+0.4 percentage points), at the cost of lower precision (-1.7 percentage points). On the real-world fsoco-ubm set, the two models tie at mAP_{50}=0.565, but two-stage achieves 3.4 percentage points higher precision, indicating fewer false positives under deployment conditions. We deployed the two-stage model because missing a cone is more dangerous than a false detection in autonomous racing. However, we do not recommend training two stage models due to their quadrupled computational cost; better to train other architectures and improve the datasets.

Table 4.3: Two-stage vs. single-stage YOLO26n training comparison.

Metric	Single-Stage	Two-Stage	Delta
FSOCO-12 mAP_{50}	**0.763**	0.761	−0.2 pp
FSOCO-12 $mAP_{50\text{-}95}$	0.524	**0.528**	+0.4 pp
FSOCO-12 Precision	**0.849**	0.832	−1.7 pp
FSOCO-12 Recall	0.694	**0.708**	+1.4 pp
fsoco-ubm mAP_{50}	0.565	**0.565**	≈ 0 pp
fsoco-ubm Precision	0.615	**0.649**	+3.4 pp
Generalization Gap	−25.9%	**−25.8%**	+0.1 pp

pp = percentage points.

4.4 Real-World Validation (fsoco-ubm)

Table 4.4 shows performance on fsoco-ubm. Every model suffers a substantial drop from the FSOCO-12 benchmark, ranging from −21.5% (YOLO11n) to −27.0% (YOLO12n). This gap reflects the harder real-world conditions described in section 3.2.3. Despite the drop, the YOLO26n variants maintain first place at mAP_{50}=0.565. An interesting finding is that YOLO11n generalizes best, losing only 21.5% and achieving the highest precision on this set (0.874), which suggests that its simpler architecture is more conservative and less prone to false positives on out-of-distribution data. YOLO12n generalizes worst among our trained models (−27.0%) and ends up tied with UBM production at 0.517.

Table 4.4: Real-world performance on fsoco-ubm test set (96 images, 1,426 instances).

Model	mAP_{50}	Prec.	Recall	Gap vs FSOCO-12
YOLO26n (two-stage)	**0.565**	**0.649**	0.462	−25.8%
YOLO26n (single)	**0.565**	0.615	**0.469**	−25.9%
YOLO11n (ours)	0.555	0.874	0.447	−21.5%
YOLO12n	0.517	0.572	0.454	−27.0%
UBM Production	0.517	0.635	0.393	−22.3%

4.5 Deployment Performance

On the RTX 4060 onboard the race car, the YOLO26n TensorRT FP16 engine averages 2.63 ms per image in isolated benchmarks (`trtexec`: 1.58 ms host-to-device, 1.02 ms GPU compute, 0.03 ms device-to-host). The production YOLO11n engine is comparable at 2.70 ms; both models are transfer-bound rather than compute-bound at this scale. Note that model inference is only one stage of the full perception pipeline—preprocessing, stereo matching, feature matching, and triangulation add substantial overhead. As shown in section 4.6, the complete pipeline runs at 15.61 ms for YOLO26n, just within the 16.7 ms

budget for 60 fps. We did not pursue INT8 quantization as the 1–2% mAP_{50} penalty is not justified when inference itself is not the bottleneck.

4.6 Integration Challenges

Deploying a new YOLO architecture revealed a hidden assumption in the inference pipeline: the output tensor format. When we first loaded the YOLO26n engine, the node produced zero detections with occasional flickering gray bounding boxes (class "unknown cone"). The TensorRT engine check logs revealed the root cause as a tensor shape mismatch between what the code expected and what the model produced.

Table 4.5 shows the critical difference. YOLO11 outputs a tensor of shape $(2, 9, 8400)$, where the 9 channels encode four bounding box coordinates (center x, center y, width, height) plus five class confidences, and 8400 is the total number of anchor-free predictions across three feature map scales ($80^2 + 40^2 + 20^2 = 8{,}400$ for a 640×640 input). These raw predictions require Non-Maximum Suppression (NMS) in postprocessing to filter overlapping detections. YOLO26, following the end-to-end paradigm introduced by YOLOv10 (Wang et al., 2024), performs NMS internally via consistent dual assignments during training and outputs $(2, 300, 6)$: up to 300 pre-filtered detections per image (Sapkota et al., 2026; Ultralytics, 2026). Each detection contains six values: corner coordinates (x_1, y_1, x_2, y_2), confidence, and class ID. The old postprocessing code assumed the YOLO11 layout, reading memory at wrong strides and interpreting random floats as coordinates and class IDs, occasionally producing a garbage detection that passed validation.

Table 4.5: Output tensor formats for legacy and end-to-end YOLO architectures.

Model	**Output Shape**	**Format**	**Box Encoding**	**NMS**
YOLO11	$(2, 9, 8400)$	[batch, $4{+}n_c$, anchors]	x_c, y_c, w, h	Software
YOLO26	$(2, 300, 6)$	[batch, max_det, 6]	x_1, y_1, x_2, y_2	Built-in

n_c = number of classes (5). Anchors: $80^2{+}40^2{+}20^2 = 8{,}400$ for $640{\times}640$ input.

The fix auto-detects the model version from output dimensions at initialization: if `d[2]==6`, the model is end-to-end; if `d[1]<d[2]`, it is legacy. Each format has its own postprocessing path. The change required approximately 40 lines of new code with zero modifications to the existing YOLO11 path, which was simply wrapped in a conditional block. This design maintains backward compatibility: the same node binary can load either model type without recompilation.

Table 4.6 compares per-stage timing between the two architectures on the full inference pipeline. The most striking result is postprocessing: YOLO11 spends 0.35 ms on NMS, while YOLO26 registers 0.00 ms (below measurement resolution) because NMS is eliminated

entirely. The total pipeline time drops from 16.56 ms to 15.61 ms, a 5.7% improvement. Feature matching is also 18% faster with YOLO26, likely because the end-to-end model produces fewer false positives and thus fewer bounding boxes to match across the stereo pair.

Table 4.6: Per-stage timing comparison on RTX 4060 (mean over >10,000 frames).

Stage	**YOLO11 (ms)**	**YOLO26 (ms)**	**Delta**
Preprocessing	0.32	0.31	−3%
Inference	5.42	5.29	−2%
Postprocessing	**0.35**	**0.00**	**−100%**
BBox matching	2.35	2.40	+2%
Feature matching	2.88	2.37	−18%
Triangulation	0.12	0.11	−8%
Sending	0.02	0.02	0%
Total	**16.56**	**15.61**	**−5.7%**

Finally, we note that the auto-exposure problem identified in Fusa's thesis (Fusa, 2025) (Section 5.2.2) has been addressed by a fix already present in the codebase. The ZED 2i camera's Auto-Exposure/Auto-Gain Control (AEC/AGC) region is restricted to the bottom half of the frame (y=360 to y=710 in the 720p image), causing the camera to expose for cones on the ground rather than the bright sky. This is a naive but effective solution: since the horizon in track conditions is approximately at the vertical center, the top half (sky) may be over- or underexposed, but cone visibility is preserved. Figure 4.2 shows YOLO26n correctly detecting cones after all integration issues were resolved.

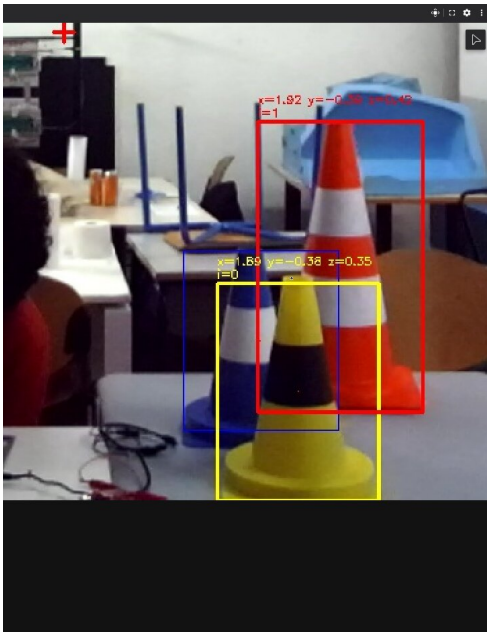

Figure 4.2: YOLO26n detecting cones in the workshop after integration fixes. The large orange cone and partially occluded yellow cone are both detected with high confidence.

Chapter 5

Conclusion

5.1 Summary of Contributions

Of the three architectures evaluated, YOLO26n achieved the best results: mAP_{50}=0.763 on FSOCO-12 (+14.6% over the previous production model) and 2.63 ms model inference on the onboard RTX 4060. The full perception pipeline—including stereo matching, feature matching, and triangulation—runs at 15.61 ms, just within the 16.7 ms budget for 60 fps. A two-stage training variant traded marginal benchmark precision for better recall and real-world generalization. Hyperparameter tuning proved ineffective. Our real-world test set, fsoco-ubm, revealed a 22–27% accuracy drop from the internet benchmark across all models. This gap would have gone unnoticed without car-specific validation data.

5.2 Lessons Learned

Contributing to the existing codebase was straightforward thanks to the modular software architecture established by Fusa (2025): detection, stereo matching, and triangulation are separate stages, so swapping the YOLO model required only replacing the TensorRT engine and updating the confidence threshold. This report also fills a documentation gap: the previous training configuration was lost when team members graduated, so we have documented every run, dataset version, and export step for reproducibility. The key technical lessons are:

- **Architecture selection matters more than hyperparameter tuning.** The sweep explored 13 parameters across 21 runs and found no improvement over defaults, while upgrading from YOLO11n to YOLO26n gained 7.9% mAP_{50}.
- **Always validate on real data.** FSOCO-12 performance does not reliably predict deployment performance: all models lost 22–27% on our car's camera data. Without fsoco-ubm, we would have overestimated the deployed model's accuracy.

- **Two-stage training provides marginal gains at this data scale.** Pre-training on 3× more data improved recall and real-world precision, but the benchmark mAP_{50} remained within 0.2% of single-stage for 4× the compute, suggesting that either the nano capacity has been saturated or the two dataset distributions overlap too much for pre-training to add value.
- **Verify framework internals.** The Ultralytics `optimizer='auto'` setting silently overwrites user-specified learning rates, which caused catastrophic forgetting during fine-tuning and cost a full day of debugging (section A.3).

5.3 Limitations

Several limitations should be noted. We could not reproduce the mAP_{50}=0.824 reported in Fusa (2025): the training configuration and dataset version used to produce that figure were lost, and the best we achieved with YOLO11n on FSOCO-12 under default hyperparameters was 0.707, which we adopted as our reproducible baseline. The fsoco-ubm test set is small (96 images) and captures a single track on a single day, so it does not cover the full range of conditions the car will face at competition—different track layouts, rain, and dusk lighting are absent. The unknown cone class remains poorly detected (mAP_{50}=0.364), a dataset-level problem rooted in annotator ambiguity. We only evaluated nano-sized models (∼2.5M parameters); larger variants may improve accuracy within the available latency budget. Finally, we did not ablate individual augmentation parameters.

5.4 Future Work

We recommend the following directions for continued development:

- **Evaluate larger YOLO26 variants.** Model inference accounts for only 5.29 ms of the 15.61 ms pipeline, so a moderately larger model such as YOLO26s could improve detection of distant and ambiguous cones without exceeding the real-time budget, provided the downstream stages are also optimized.
- **Expand fsoco-ubm.** Add images from different tracks, weather conditions, and times of day. The current 96 images from a single session are a starting point, not a representative benchmark. With 1,000+ annotated images, fsoco-ubm could also serve as a fine-tuning set, closing the benchmark-to-deployment gap directly rather than just measuring it.
- **Target the unknown cone class.** Data augmentation strategies such as color jittering and occlusion simulation may help the model learn to flag uncertain detections. Alternatively, merging unknown cones into the nearest color class and

combining the two orange cone classes during training could improve overall recall at the cost of classification granularity.

- **Optimize downstream pipeline stages.** Feature matching (2.37 ms) and bounding box matching (2.40 ms) together consume nearly as much time as model inference (5.29 ms), and the full pipeline sits at 15.61 ms with only 1 ms of headroom. Optimizing these stages from CPU to CUDA by computing a dense disparity map then for each left-image detection sampling the median disparity within the bounding box region, eliminating the need to run YOLO or match bounding boxes on the right image, would both improve real-time margin and create room for a larger detection model.

- **Race-condition testing.** The full autonomous stack ran successfully in a workshop setting (section 4.6), but has not yet been tested under race conditions with the new model. The racecar will be tested on track the week of 23-28 of February 2026.

UBM will compete at Formula Student Germany from 11-16 August 2026. The goal is to complete all four dynamic events (Trackdrive, Autocross, Skid Pad, and Acceleration) for the first time since last year a broken axle ended the campaign before we could attempt any of them. The priority is reliability over outright speed: finishing every event and collecting data is more valuable than optimizing for a fast lap, as the data enables iteration for the 2027 competition. The improvements in this report contribute to that goal by providing a more accurate, faster, and better documented perception pipeline.

Appendix A

Technical Details

A.1 Hyperparameter Sweep Configuration

Listing A.1 shows the full W&B Bayesian sweep configuration. The 13 tuned parameters span learning rate scheduling, regularization, and data augmentation. Fixed parameters (epochs=100, batch=48, image size=640) were held constant across all runs to isolate the effect of the tuned parameters. Batch size was reduced from 64 to 48 due to a memory leak between consecutive W&B runs.

```
program: train_sweep.py
method: bayes
metric:
  name: metrics/mAP50(B)
  goal: maximize
early_terminate:
  type: hyperband
  min_iter: 30
  eta: 2
  s: 3
parameters:
  lr0:           {distribution: uniform, min: 0.005, max: 0.02}
  lrf:           {distribution: uniform, min: 0.01,  max: 0.1}
  warmup_epochs: {values: [1, 3, 5]}
  hsv_h:         {distribution: uniform, min: 0.0,   max: 0.03}
  hsv_s:         {distribution: uniform, min: 0.5,   max: 0.9}
  hsv_v:         {distribution: uniform, min: 0.3,   max: 0.6}
  mosaic:        {distribution: uniform, min: 0.5,   max: 1.0}
  close_mosaic:  {values: [0, 5, 10, 15, 20]}
  mixup:         {distribution: uniform, min: 0.0,   max: 0.3}
  copy_paste:    {distribution: uniform, min: 0.0,   max: 0.3}
  weight_decay:  {distribution: uniform, min: 0.0001,max: 0.001}
```

```
  dropout:        {distribution: uniform, min: 0.0,    max: 0.2}
  degrees:        {distribution: uniform, min: 0.0,    max: 10.0}
  # Fixed
  epochs: {value: 100}
  batch:  {value: 48}
  imgsz:  {value: 640}
```

Listing A.1: W&B Bayesian sweep configuration.

A.2 Two-Stage Training Hyperparameters

Table A.1 documents the exact configuration for each training stage. Stage 2 was split into two phases after the initial attempt caused catastrophic forgetting (section A.3). Phase 2A freezes the backbone and trains only the detection head; Phase 2B unfreezes all layers with an ultra-low learning rate.

Table A.1: Two-stage training configuration.

Parameter	**Stage 1**	**Phase 2A**	**Phase 2B**
Dataset	cone-detector (22,725)	FSOCO-12 (7,120)	FSOCO-12 (7,120)
Epochs	338 (early stop)	50	250
Batch size	64	64	64
Optimizer	auto (SGD)	AdamW	AdamW
Learning rate	0.01	0.001	0.00005
LR final (lrf)	0.01	0.0001	0.000005
Warmup epochs	3	10 (20%)	50 (20%)
Frozen layers	None	0–10 (backbone)	None
LR schedule	Linear	Cosine	Cosine
Pretrained weights	COCO (`yolo26n.pt`)	Stage 1 `best.pt`	Phase 2A `best.pt`
Image size	640	640	640

A.3 The `optimizer='auto'` Bug

Our first attempt at Stage 2 fine-tuning failed immediately. We configured `lr0=0.001` (10× lower than training from scratch), but the model's mAP_{50} dropped from 0.754 to 0.628 in two epochs, an instance of catastrophic forgetting. The training logs revealed the cause:

```
optimizer: 'optimizer=auto' found, ignoring 'lr0=0.001'
           and 'momentum=0.937' and determining best
           'optimizer', 'lr0' and 'momentum' automatically...
optimizer: MuSGD(lr=0.01, momentum=0.9) with parameter groups
```

Listing A.2: Ultralytics log output showing the overridden learning rate.

The `optimizer='auto'` default silently replaced our learning rate with a hardcoded `lr=0.01`, the same rate used for training from scratch, and 10× higher than what we specified. The internal heuristic selects SGD with `lr=0.01` whenever the total iteration count exceeds 10,000 (ours was 33,375), regardless of user-specified parameters. This behavior is documented in Ultralytics GitHub issues #17444 and #9182 but is not prominently noted in the official documentation.

The fix was to explicitly set `optimizer='AdamW'`, which respects the user-specified `lr0`. Combined with the two-phase freeze/unfreeze strategy described in section 3.6, this eliminated the forgetting entirely. When fine-tuning with Ultralytics, never rely on `optimizer='auto'` and always set the optimizer explicitly.

A.4 fsoco-ubm Dataset Creation Pipeline

The fsoco-ubm test set was created in five steps from raw ROS bag recordings captured on November 20, 2025 at the Rioveggio test track.

Step 1: Video extraction. The `.mcap` ROS bag files were converted to AVI videos using ROS2 tooling on the ASU. The two resulting files (LidarTest1 and LidarTest2) contain stereo-stitched frames at 2560×720 resolution. Although exported at 60 FPS, the original recording was 30 FPS, so the video runs at double speed.

Step 2: Frame sampling. We extracted one frame every 60 frames (2 seconds of real-world time) to avoid near-duplicate images while preserving temporal diversity. Each stereo frame was split into left and right images of 1280×720. This yielded 46 stereo pairs (92 images) from the two videos, as shown in listing A.3.

```
import cv2

cap = cv2.VideoCapture("media/lidar1.avi")
frame_count = 0
saved = 0
while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        break
    if frame_count % 60 == 0:  # every 2 seconds
        h, w = frame.shape[:2]
        mid = w // 2
        left  = frame[:, :mid]
        right = frame[:, mid:]
```

```
        cv2.imwrite(f"images/lidar1_left_{saved:04d}.jpg", left)
        cv2.imwrite(f"images/lidar1_right_{saved:04d}.jpg", right
            )
        saved += 1
    frame_count += 1
cap.release()
```

Listing A.3: Frame extraction and stereo splitting.

Step 3: Upload and annotation. The 96 images were uploaded to Roboflow and annotated using Label Assist (automated pre-labeling) followed by manual review for the five cone classes. Annotation took approximately 8 hours.

Step 4: Export. The annotated dataset was exported from Roboflow in YOLO format for evaluation.

Step 5: Format mismatch bug. We initially exported the dataset in `yolo26` format, which uses a different annotation structure than the `yolov11` format our models were trained on. YOLO26 places normalized `xywh` coordinates in a different column order, causing the evaluation script to silently misparse bounding boxes. All initial fsoco-ubm results were invalid with models appeared to perform far worse than they actually did. The fix was re-exporting in `yolov11` format to match the training data. This failure mode produces no error message; the only symptom is anomalously low metrics. When evaluating a model on a new dataset, always verify that the annotation format matches the format used during training.

Bibliography

Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. YOLOv4: Optimal speed and accuracy of object detection. *arXiv preprint arXiv:2004.10934*, 2020.

Nicolas Cridlig. Code and training scripts for cone detection improvement, 2026. URL https://github.com/ncridlig/ml4cv_project. Accessed February 2026.

Mark Everingham, Luc Van Gool, Christopher K. I. Williams, John Winn, and Andrew Zisserman. The PASCAL visual object classes (VOC) challenge. *International Journal of Computer Vision*, 88(2):303–338, 2010. doi: 10.1007/s11263-009-0275-4.

FMDV. FSOCO cone detection dataset (version 12), 2024. URL https://universe.roboflow.com/fmdv/fsoco-kxq3s/dataset/12. Roboflow Universe. Accessed January 2026.

Formula Student Germany. Formula Student Germany — driverless, 2024. URL https://www.formulastudent.de/fsg/. Accessed January 2026.

Robert M. French. Catastrophic forgetting in connectionist networks. *Trends in Cognitive Sciences*, 3(4):128–135, 1999. doi: 10.1016/S1364-6613(99)01294-2.

FSB Driverless. Cone detector dataset (version 1), 2024. URL https://universe.roboflow.com/fsbdriverless/cone-detector-zruok/dataset/1. 22,725 images. Roboflow Universe. Accessed January 2026.

FSOCO Contributors. FSOCO: Formula student objects in context, 2020. URL https://www.fsoco-dataset.com/. Community dataset for Formula Student cone detection.

Edoardo Fusa. Pushing cars' limits: Exploring autonomous technologies in the formula SAE driverless competition. Master's thesis, University of Bologna, 2025. Stereocamera pipeline for autonomous racing.

Glenn Jocher, Ayush Chaurasia, and Jing Qiu. Ultralytics YOLOv8, 2023. URL https://github.com/ultralytics/ultralytics. Accessed January 2026.

Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: Common objects in context. *arXiv preprint arXiv:1405.0312*, 2014.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations (ICLR)*, 2019.

NVIDIA Corporation. NVIDIA TensorRT: Programmable inference accelerator, 2024. URL https://developer.nvidia.com/tensorrt. Accessed January 2026.

ONNX Community. ONNX: Open neural network exchange, 2019. URL https://onnx.ai/. Accessed January 2026.

Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788, 2016.

Roboflow, Inc. Roboflow: Computer vision tools, 2024. URL https://roboflow.com/. Dataset management and annotation. Accessed January 2026.

SAE International. FSAE driverless supplement v11, 2025. URL https://www.fsaeonline.com/CompResources/2025/8f030a58-d9e4-49b8-bc83-6ca16c7ce715/FSAE-Driverless-Supplement-V11.pdf. Section: Dynamic Event Markings.

Ranjan Sapkota et al. YOLO26: Key architectural enhancements and performance benchmarking for real-time object detection. *arXiv preprint arXiv:2509.25164*, 2026.

Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. Practical Bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 25, 2012.

Stereolabs. ZED 2i stereo camera, 2021. URL https://www.stereolabs.com/products/zed-2i. Accessed January 2026.

Yunjie Tian, Qiang Ye, and David Doermann. YOLO12: Attention-centric real-time object detectors. *arXiv preprint arXiv:2502.12524*, 2025.

Ultralytics. YOLO11: Real-time object detection, 2024. URL https://docs.ultralytics.com/models/yolo11/. Accessed January 2026.

Ultralytics. Ultralytics YOLO26 documentation, 2026. URL https://docs.ultralytics.com/models/yolo26/. Accessed January 2026.

Ao Wang, Hui Chen, Lihao Liu, Kai Chen, Zijia Lin, Jungong Han, and Guiguang Ding. YOLOv10: Real-time end-to-end object detection. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 37, 2024.

Weights & Biases, Inc. Weights & Biases, 2024. URL https://wandb.ai/. Experiment tracking platform. Accessed January 2026.

Hongyi Zhang, Moustapha Cisse, Yann N. Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412*, 2018.