Real-Time Object Detection for Autonomous Driving: An Empirical Study in a Small-Scale Urban Environment

Krittin Chaowakarn

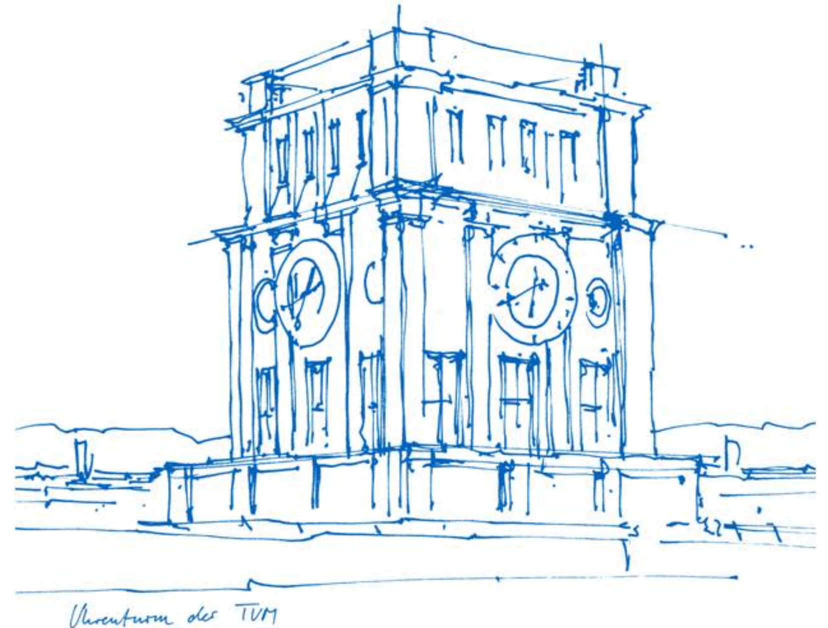
Bachelor’s Thesis

Technical University of Munich

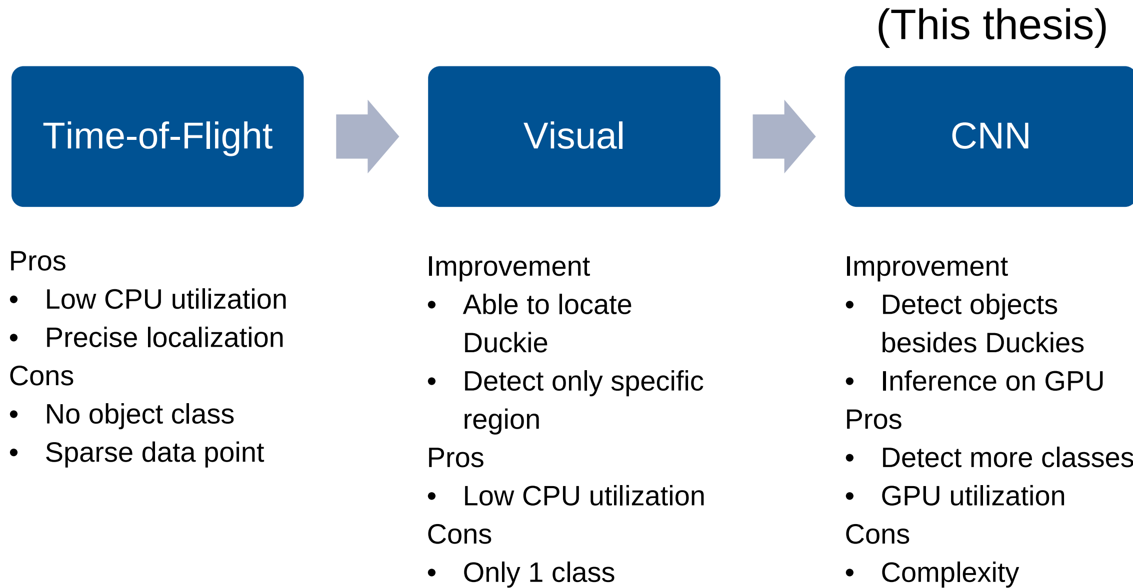
TUM School of Computation, Information, and Technology

Chair of Integrated Systems

Munich, September 10, 2025



Introduction



Picture from [1]

Presentation Outline

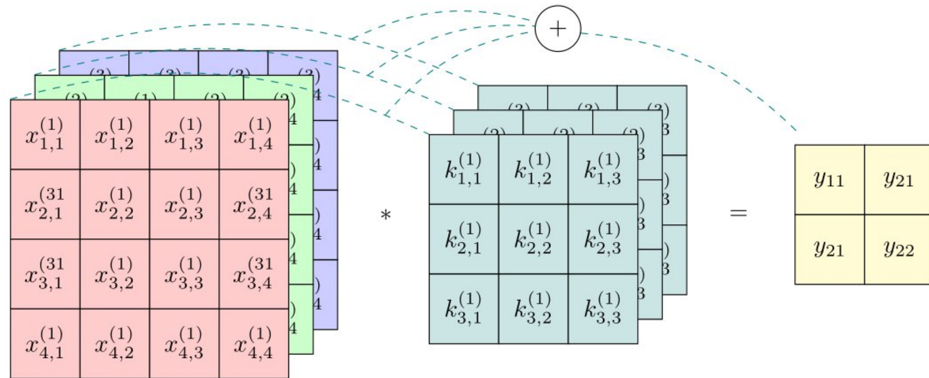
1. Background Knowledge
2. Dataset, Data Collection, and Fine-Tuning YOLO Models
3. Fundamental Decision Making
4. Results
5. Case Study
6. Conclusion and Outlook

Background Knowledge

Convolutional Neural Network (CNN) and Key Advantages

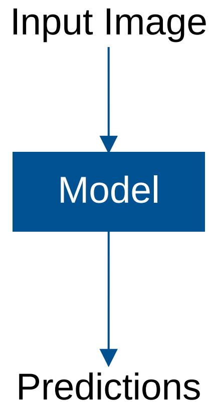
Key Advantages

1. Gathering local features
2. Strong generalization
3. Fewer parameters
4. Hierarchical Features
5. Parallelization on GPU

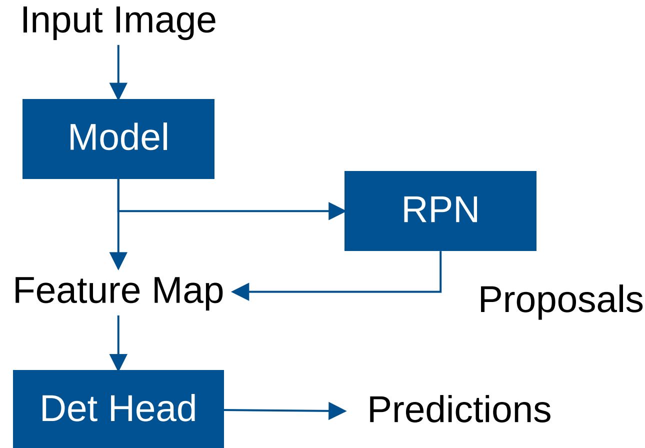


Basic CNN-based Models for Object Detection

One-Stage Model



Two-Stage Model



One-stage model

- Faster
- Less precise

Two-stage model

- Slower
- More precise

RPN: Region Proposal Network
Det Head: final conv block for refinement

You Only Look Once (YOLO) Models

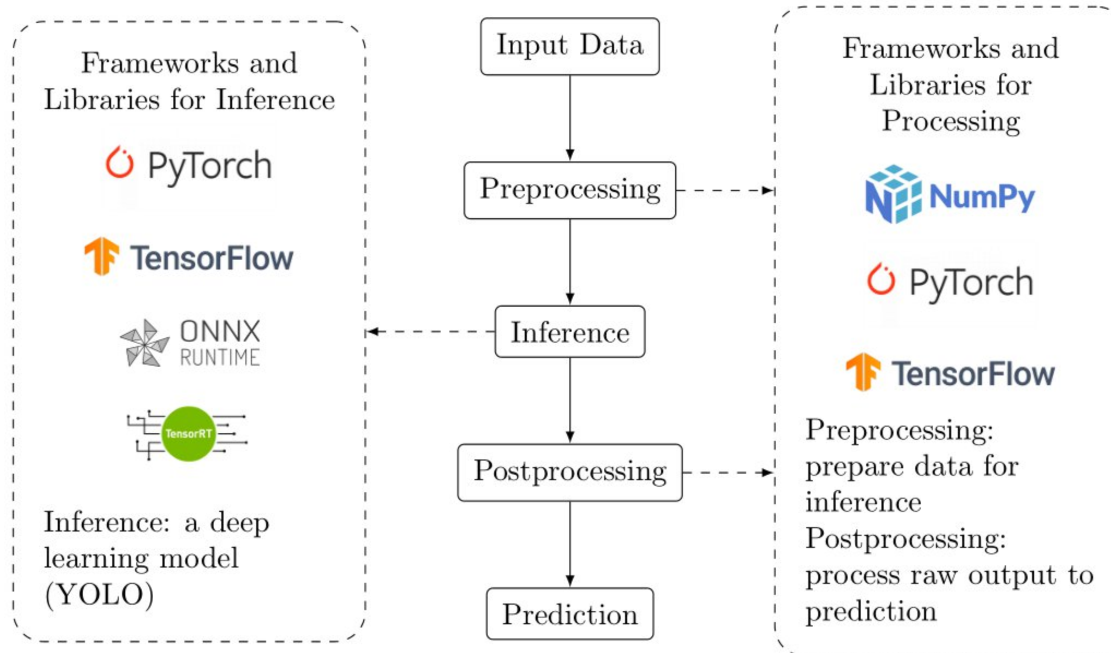
YOLO is a series of real-time object detection systems, maintained by Ultralytics [2].

YOLO has 12 different versions:

1. YOLOv1-v7 (2015-2022)
   - Old
   - Only object detection
2. YOLOv8-v12 (2023-2025)
   - Modern
   - a full range of vision AI tasks

We use only smallest model from each version to ensure a lightweight option.

Pipeline and Frameworks

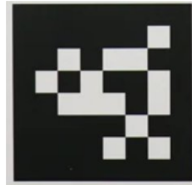


Dataset, Data Collection, and Fine-Tuning YOLOs

Datasets

Open-Source Dataset from Duckietown [3]

7 Classes:



Key difference from our lab:

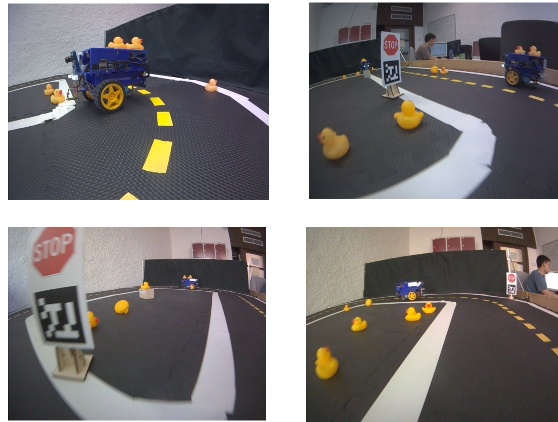
1. New Duckiebot design
2. Just three classes
3. Lighting conditions

All images in this slide can be found on [4]

Datasets

The Open-Source Dataset

Our Environment

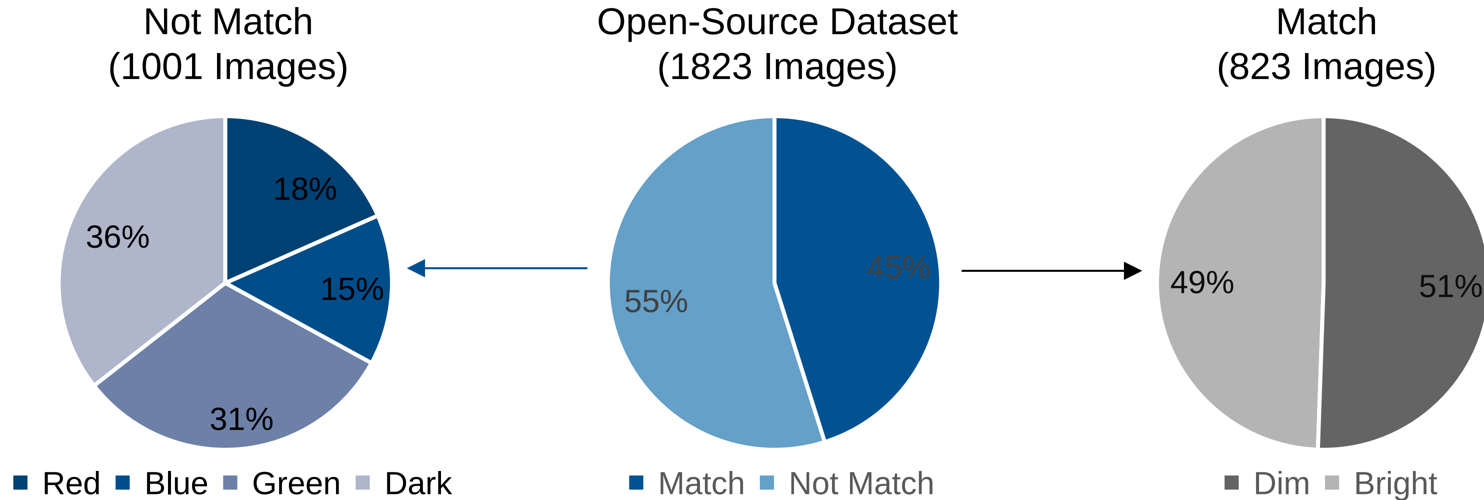


Possible Solutions

1. Collect a new dataset (too much effort)
   i. For training
   ii. For fine-tuning

2. Collect a little data for evaluating

The Open-Source Dataset Distribution



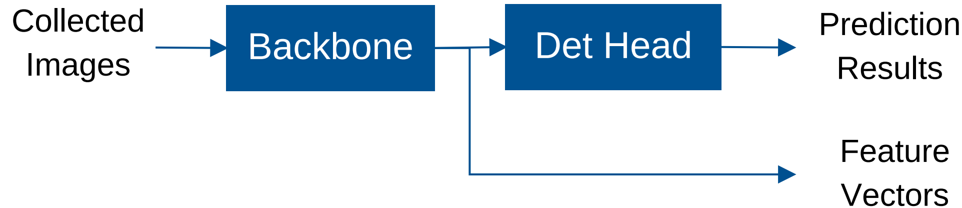
Data Collection and Selection Process

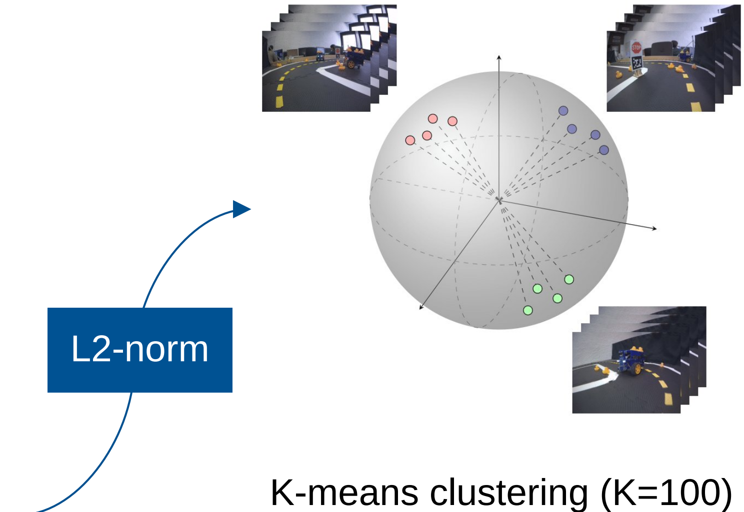
Data Collection

- Automatically collect every 1 seconds

Selection Process

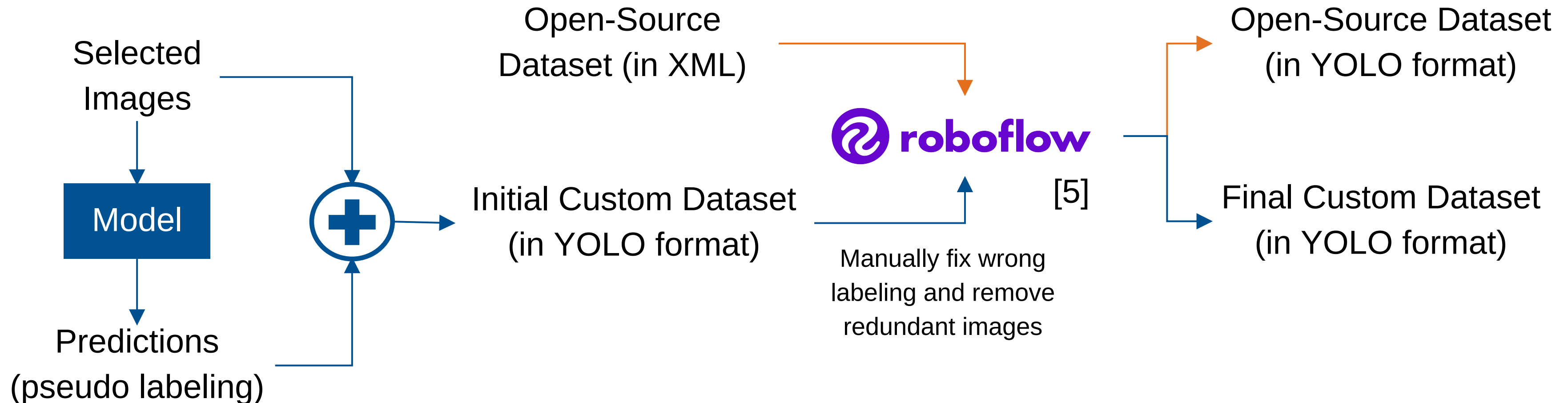
- Use a model fine-tuned with the open-source dataset to extract feature



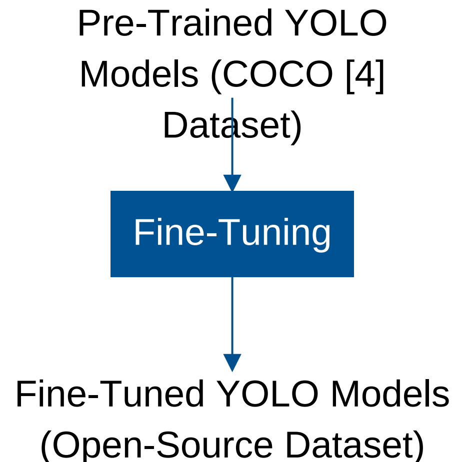


K-means clustering (K=100)

Processing All Datasets into YOLO Format



Fine-Tuning YOLO Models



Fixed (common practice):

1. Learning rate = 1e-4
2. Epochs = 100
3. Batchsize = 16

Varied:

1. Input size = 640, 480, 320, 160 (width/height of a square image)
2. Versions of YOLO nano

Note: COCO is a large-scale object detection, segmentation, and captioning dataset (330K images).

Fine-Tuning Results

640x640px Input Resolution

Model	Open-Source		Custom	
	mAP_{50}	mAP_{50-95}	mAP_{50}	mAP_{50-95}
YOLOv8n	90.98	58.84	**44.57**	**30.41**
YOLOv9t	90.12	58.05	44.34	28.40
YOLOv10n	90.02	58.52	43.20	28.62
YOLOv11n	91.02	59.49	44.36	29.57
YOLOv12n	**92.65**	**60.02**	40.43	24.74

480x480px Input Resolution

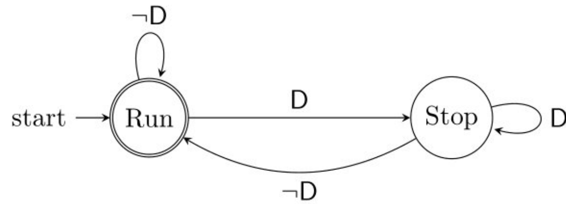
Model	Open-Source		Custom	
	mAP_{50}	mAP_{50-95}	mAP_{50}	mAP_{50-95}
YOLOv8n	**91.17**	58.30	41.08	25.57
YOLOv9t	90.50	57.65	43.20	27.27
YOLOv10n	89.21	57.82	41.57	27.77
YOLOv11n	90.64	58.18	**45.54**	**29.50**
YOLOv12n	91.12	**58.68**	41.30	25.37

From previous stage, we selected only YOLOv11n with input size of 640 and 480 based on their performance on our custom dataset.

Fundamental Decision Making

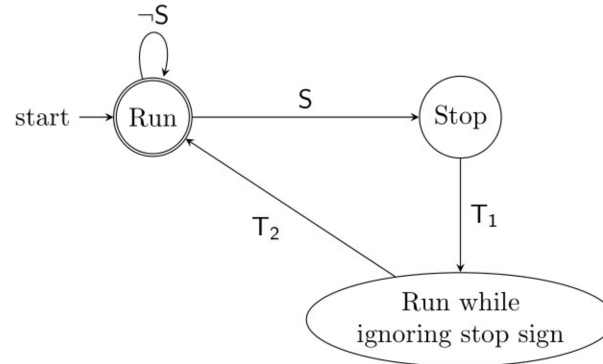
Stop-and-Wait System

Duckie/Duckiebot Detection



D = Duckie/Duckiebot Detected

Stop Signs Detection



S = Stop Sign Detected

T_1 = 3-Seconds Timer

T_2 = 5-Seconds Timer

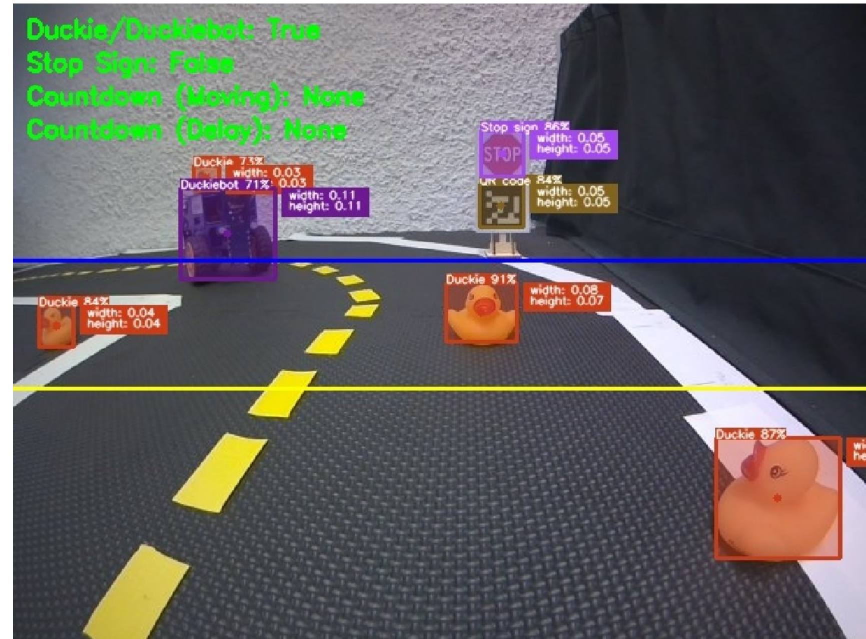
Simple Rule-Based Decision Making

Region-Based Method

1. Below blue line: Duckiebot
2. Below yellow line: Duckie

Normalized Object Dimension-Based Method

1. Duckie: Threshold = 0.08
2. Stop sign: Threshold = 0.1



Results

Processing and Inference Times During Deployment

Times in milliseconds, results averaged over 30-step moving average.
Pre- and postprocessing always on CPU

Framework	Model	Preprocessing		Inference		Postprocessing		Total	
		CPU	GPU	CPU	GPU	CPU	GPU	CPU	GPU
NumPy	YOLOv11n@480	**40.81**	53.19	1443.25	**131.73**	2.27	1.85	1486.33	**186.77**
	YOLOv11n@640	69.00	75.55	2506.90	162.58	**1.02**	1.41	2576.92	239.54
PyTorch	YOLOv11n@480	66.05	56.16	1477.23	140.62	14.76	22.42	1558.04	219.20
	YOLOv11n@640	64.82	91.02	2261.47	189.28	12.77	7.38	2339.06	287.68

Runtime on GPU is always faster than runtime on CPU

ROS Publishing Frequencies During Deployment

Frequencies of important topics in Hz

Framework	Model	Image In		Detection Array		Car CMD	
		CPU	GPU	CPU	GPU	CPU	GPU
No Detection		**14.0**		0		**21.3**	
NumPy	YOLOv11n@480	13.5	10.3	0.9	**4.9**	10.5	11.2
	YOLOv11n@640	11.5	9.6	0.7	3.8	11.5	12.8
PyTorch	YOLOv11n@480	11.5	7.3	1.1	4.5	9.5	12.5
	YOLOv11n@640	10.0	8.5	0.5	3.6	12.0	12.5

Publishing Rate of Detection Array on GPU is always faster than on CPU

Hardware Utilization During Deployment

RAM, CPU, and GPU Utilization when Inference is on CPU and GPU

Framework	Model	RAM (%)		CPU (%)		GPU (%)	
		CPU	GPU	CPU	GPU	CPU	GPU
No Detection		74.85		**93.84**		0	
Numpy	YOLOv11n@480	88.04	94.01	98.97	97.96	0	31.22
	YOLOv11n@640	**42.27**	80.36	99.08	97.40	0	40.83
PyTorch	YOLOv11n@480	79.66	92.40	99.02	97.66	0	**25.97**
	YOLOv11n@640	60.49	55.87	98.89	97.50	0	34.92

GPU utilization remains below approximately 40%

YOLO Model Selection

“YOLOv11n@480 + NumPy” on a GPU is used for our case study.

Case Study

Case Study

Nominal Cases

Duckie
- Different facing directions
- Different distances
- Different angles

Duckiebot
- Different facing directions
- Different distances

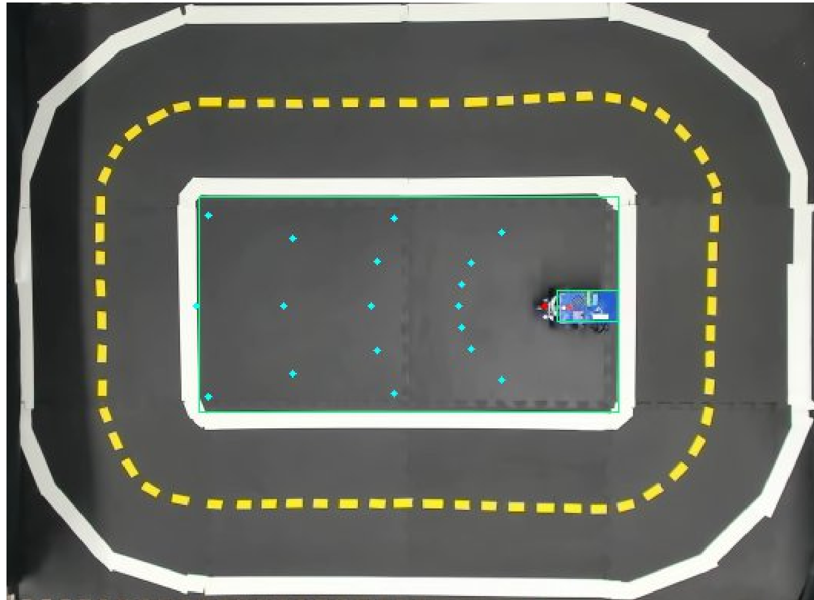
Stop sign
- Different distances
- Always facing the camera

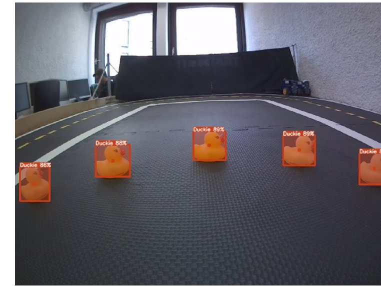
Safety-Critical Cases

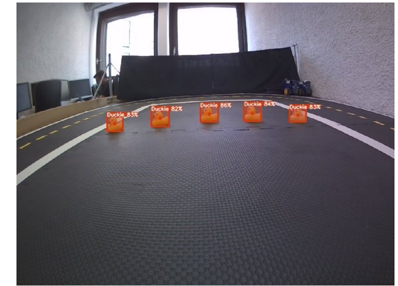
Duckie
- Occlusion at different levels
- Very close

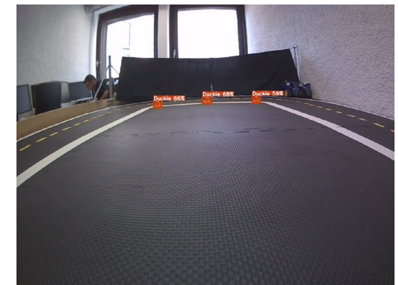
Duckiebot
- Occlusion at different levels

Case Study – Duckie Nominal Cases

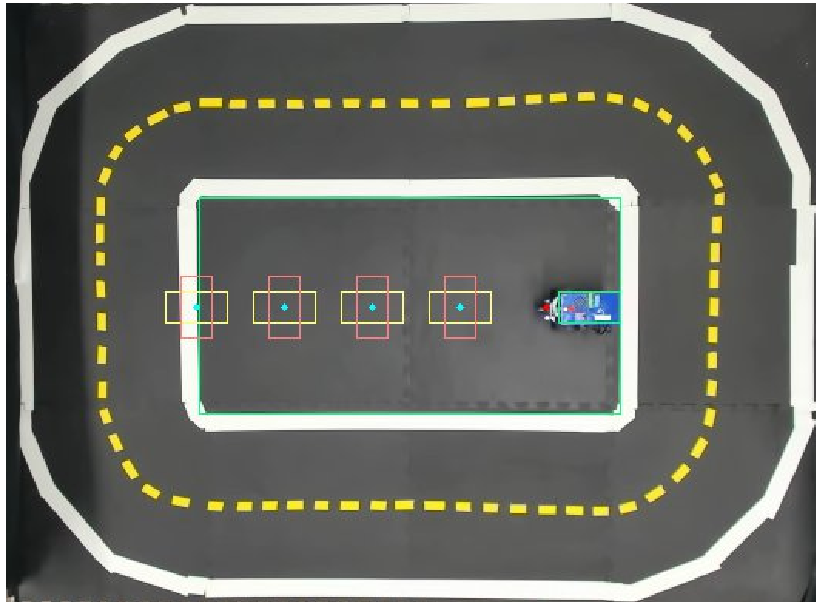


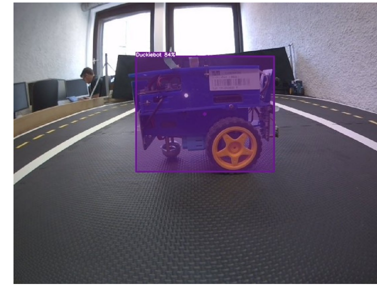


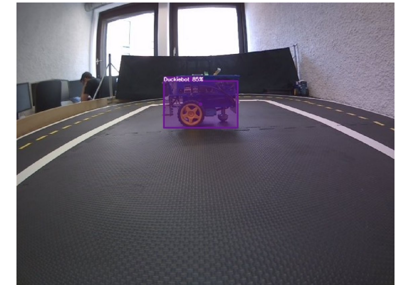




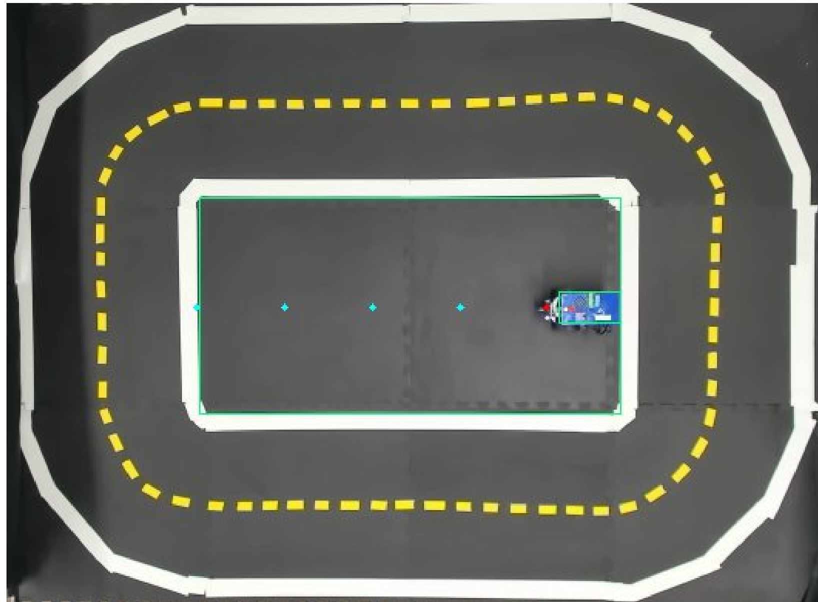
Case Study – Duckiebot Nominal Cases

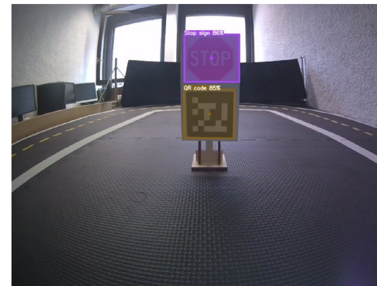


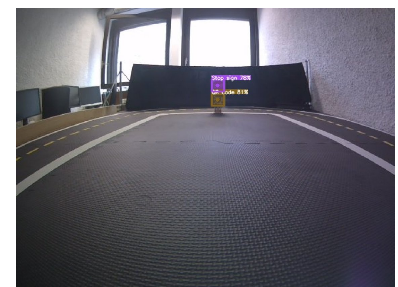




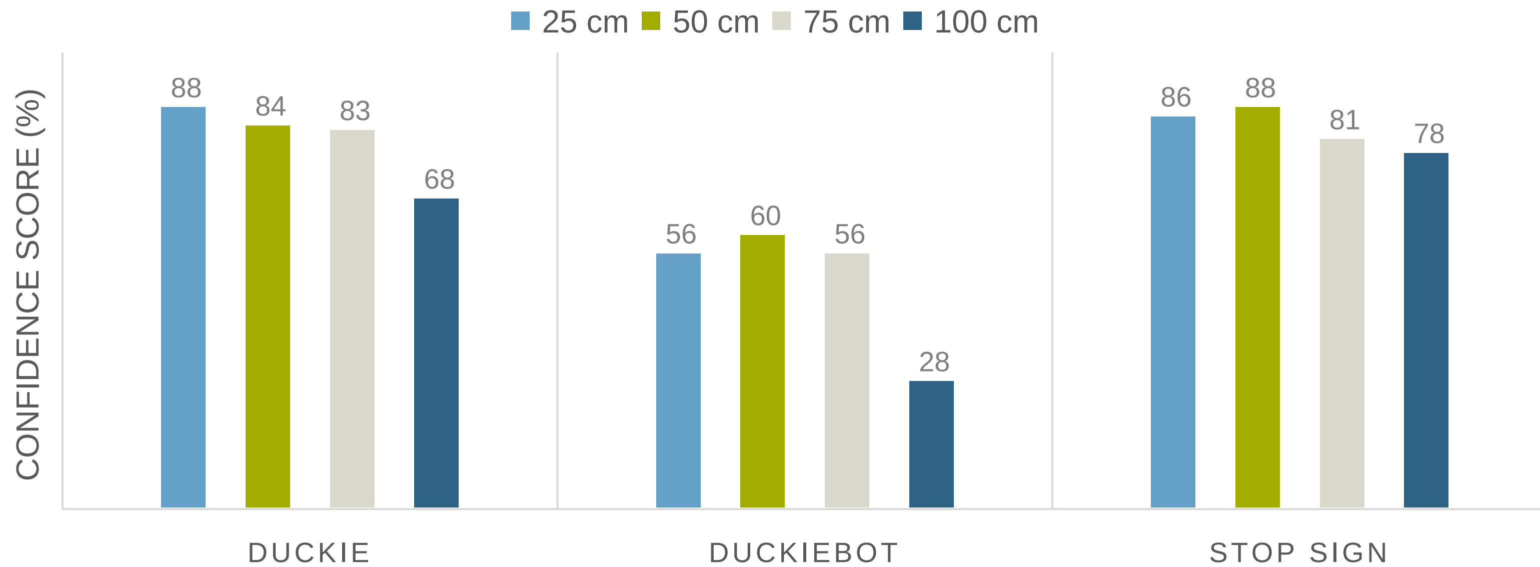
Case Study – Stop Sign Nominal Cases







Case Study – Nominal Case Summary



Case Study – Selected Safety-Critical Cases (Occlusion)

Duckie with 75% occlusion

Duckiebot with 50% occlusion

59% Confidence Score

Undetected

77% Confidence Score

Undetected

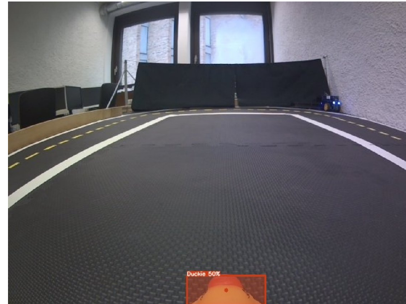
Duckie’s beak contains high feature.

Duckiebot’s wheels contain high feature.

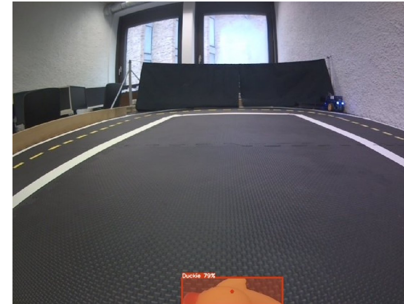
Case Study – Selected Safety-Critical Cases (Very Close)

Duckie positioned at 5 cm from the Duckiebot's camera

Undetected
(Front)



50% Confidence Score
(Back)



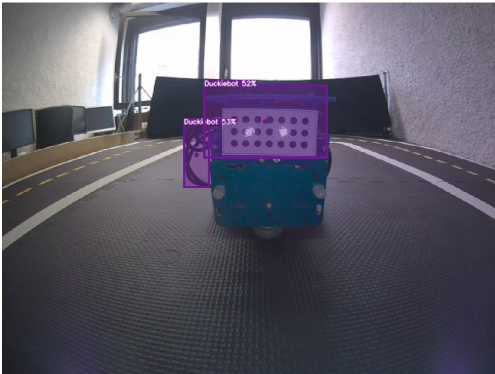
79% Confidence Score
(Left)

67% Confidence Score
(Right)

Duckie's beak contains high feature.

Case Study – More Analysis

The back of the Duckiebot



Picture from [1]

Shared features of both designs:

- Cables
- Dot matrix

Conclusion and Outlook

Conclusion

1. CNN-based Object Detection is viable for real-time application.
2. Models on GPU highly outperform models on CPU.
3. Using the GPU saves CPU utilization.
4. A custom dataset was collected to evaluate performance.
5. Prediction results can be used for simple decision-making.

Research Opportunities

- High CPU and RAM utilization model quantization, converting weights and biases from FP32 to INT8
- Extending our custom dataset
- Adding reasoning capabilities (VLM/VLA)

Reference

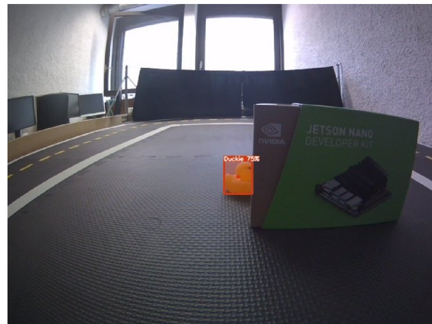
1. https://duckietown.com/
2. https://www.ultralytics.com/
3. https://github.com/duckietown/duckietown-objdet
4. https://cocodataset.org/
5. https://roboflow.com/
6. https://arxiv.org/abs/2405.05885
7. https://arxiv.org/abs/2506.13757

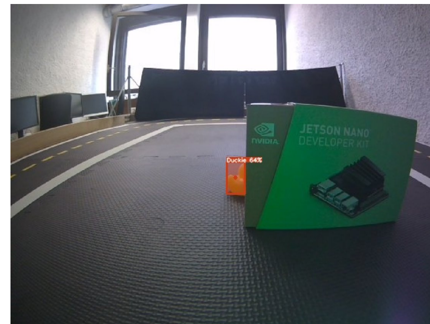
Thank You For Your Attention!

Extra Results

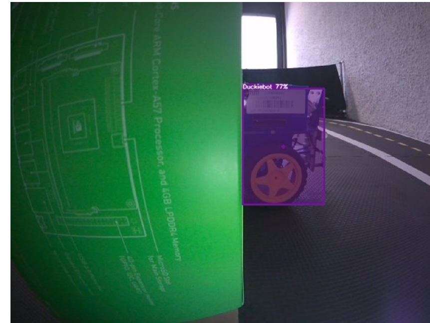
Safety-Critical Cases

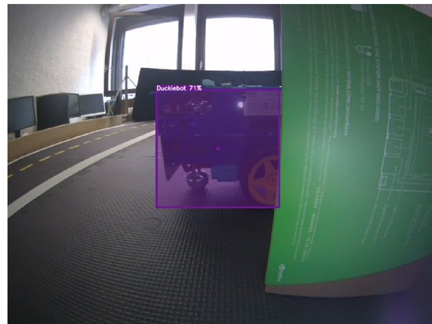
Safety-Critical Cases

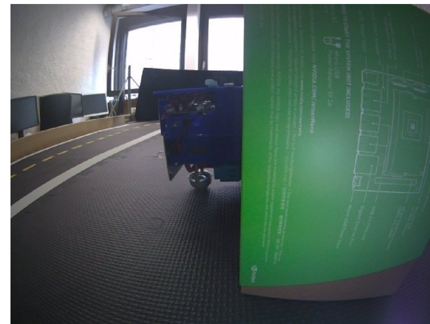




Safety-Critical Cases



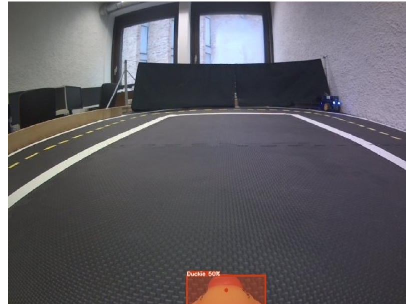




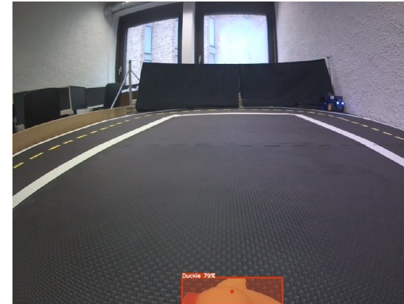
Safety-Critical Cases

Duckie positioned at 5 cm from the Duckiebot’s camera

Front



Back



Left

Right

Model Performance

Model Size and FLOPs

Model Parameters and FLOPs [79].

Model	Params (M)	FLOPs (B)	mAP_{50-95} on COCO [47]
YOLOv8n	3.2	8.7	37.3
YOLOv9t	2.0	7.7	38.3
YOLOv10n	2.3	6.7	38.5
YOLOv11n	2.6	6.5	39.5
YOLOv12n	2.6	6.5	40.6

Performance Comparison of YOLO Models

Performance evaluation of YOLO-nano models with input size 640.

Model	Open-Source		Custom		ΔmAP$_{50}$	ΔmAP$_{50-95}$
	mAP$_{50}$	mAP$_{50-95}$	mAP$_{50}$	mAP$_{50-95}$		
YOLOv8n	90.98	58.84	**44.57**	**30.41**	-46.41	-28.43
YOLOv9t	90.12	58.05	44.34	28.40	-45.78	-29.65
YOLOv10n	90.02	58.52	43.20	28.62	-46.82	-29.90
YOLOv11n	91.02	59.49	44.36	29.57	-46.66	-29.92
YOLOv12n	**92.65**	**60.02**	40.43	24.74	-52.22	-35.28

Performance Comparison of YOLO Models

Performance evaluation of YOLO-nano models with input size 480.

Model	Open-Source		Custom		ΔmAP_{50}	ΔmAP_{50-95}
	mAP_{50}	mAP_{50-95}	mAP_{50}	mAP_{50-95}		
YOLOv8n	**91.17**	58.30	41.08	25.57	-50.09	-32.73
YOLOv9t	90.50	57.65	43.20	27.27	-47.30	-30.38
YOLOv10n	89.21	57.82	41.57	27.77	-47.64	-30.05
YOLOv11n	90.64	58.18	**45.54**	**29.50**	-45.10	-28.68
YOLOv12n	91.12	**58.68**	41.30	25.37	-49.82	-33.31

Performance Comparison of YOLO Models

Performance evaluation of YOLO-nano models with input size 320.

Model	Open-Source		Custom		ΔmAP_{50}	ΔmAP_{50-95}
	mAP_{50}	mAP_{50-95}	mAP_{50}	mAP_{50-95}		
YOLOv8n	**87.32**	**54.19**	37.89	**23.08**	-49.43	-31.11
YOLOv9t	85.99	52.67	**37.93**	21.78	-48.06	-30.89
YOLOv10n	85.58	53.55	37.06	22.73	-48.52	-30.82
YOLOv11n	85.59	53.26	37.78	22.42	-47.81	-30.84
YOLOv12n	86.14	52.53	35.19	19.94	-50.95	-32.59

Performance Comparison of YOLO Models

Performance evaluation of YOLO-nano models with input size 160.

Model	Open-Source		Custom		ΔmAP$_{50}$	ΔmAP$_{50-95}$
	mAP$_{50}$	mAP$_{50-95}$	mAP$_{50}$	mAP$_{50-95}$		
YOLOv8n	**59.77**	32.22	**24.75**	**13.62**	-35.02	-18.60
YOLOv9t	57.85	30.79	21.19	10.65	-36.66	-20.14
YOLOv10n	57.50	31.57	20.62	10.65	-36.88	-20.92
YOLOv11n	59.60	**32.95**	22.87	11.31	-36.73	-21.64
YOLOv12n	57.50	31.55	22.62	12.02	-34.88	-19.53

Model Performance while Duckiebot is Stationary

Performance while a Duckiebot is stationary

Pre-, post-processing, inference, and total times are reported in milliseconds and averaged using a 30-step simple moving average while stationary. Inference may run on either CPU or GPU, as indicated, while pre- and post-processing always run on CPU regardless of the inference device.

Framework	Model	Preprocessing		Inference		Postprocessing		Total	
		CPU	GPU	CPU	GPU	CPU	GPU	CPU	GPU
NumPy	YOLOv11n@480	50.00	53.70	1615.93	**142.67**	1.02	1.94	1666.95	**198.31**
	YOLOv11n@640	68.12	127.14	2627.56	179.10	2.34	**0.54**	2698.02	306.78
PyTorch	YOLOv11n@480	**46.56**	64.51	1761.40	150.35	25.39	13.98	1833.35	228.84
	YOLOv11n@640	62.18	107.52	2535.40	198.31	19.52	13.76	2617.10	319.59

Performance while a Duckiebot is stationary

Publishing frequencies (Hz) of the primary stages in the object detection pipeline while stationary.

Framework	Model	Image In		Detection Array		Car CMD	
		CPU	GPU	CPU	GPU	CPU	GPU
No Detection		9.6		0		**14.2**	
NumPy	YOLOv11n@480	**9.8**	7.7	1.0	3.5	12.0	7.8
	YOLOv11n@640	9.0	7.7	0.5	**4.2**	10.5	11.5
PyTorch	YOLOv11n@480	7.5	7.5	0.9	4.7	8.5	11.5
	YOLOv11n@640	8.2	7.0	0.6	2.7	12.0	9.5

Performance while a Duckiebot is stationary

Comparison of RAM, CPU, and GPU utilization while stationary.

Framework	Model	RAM (%)		CPU (%)		GPU (%)	
		CPU	GPU	CPU	GPU	CPU	GPU
No Detection		**40.89**		**96.48**		0	
Numpy	YOLOv11n@480	62.88	60.60	99.30	98.72	0	26.35
	YOLOv11n@640	80.80	79.78	99.38	98.05	0	29.86
PyTorch	YOLOv11n@480	43.32	79.44	99.38	98.04	0	**19.75**
	YOLOv11n@640	75.88	94.14	99.45	98.08	0	33.89

Performance while a Duckiebot is stationary

Comparison of CPU and GPU temperatures while stationary.

Framework	Model	CPU Temp (°C)		GPU Temp (°C)	
		CPU	GPU	CPU	GPU
No Detection		33.64		34.43	
Numpy	YOLOv11n@480	**33.13**	33.73	33.84	**32.83**
	YOLOv11n@640	33.96	34.48	34.46	33.45
PyTorch	YOLOv11n@480	34.40	34.60	35.04	33.11
	YOLOv11n@640	33.98	37.17	34.86	35.89

Model Performance while Duckiebot is Running

Performance while a Duckiebot is running

Pre-, post-processing, inference, and total times are reported in milliseconds and averaged using a 30-step simple moving average while running. Inference may run on either CPU or GPU, as indicated, while pre- and post-processing always run on CPU regardless of the inference device.

Framework	Model	Preprocessing		Inference		Postprocessing		Total	
		CPU	GPU	CPU	GPU	CPU	GPU	CPU	GPU
NumPy	YOLOv11n@480	**40.81**	53.19	1443.25	**131.73**	2.27	1.85	1486.33	**186.77**
	YOLOv11n@640	69.00	75.55	2506.90	162.58	**1.02**	1.41	2576.92	239.54
PyTorch	YOLOv11n@480	66.05	56.16	1477.23	140.62	14.76	22.42	1558.04	219.20
	YOLOv11n@640	64.82	91.02	2261.47	189.28	12.77	7.38	2339.06	287.68

Performance while a Duckiebot is running

Publishing frequencies (Hz) of the primary stages in the object detection pipeline while running.

Framework	Model	Image In		Detection Array		Car CMD	
		CPU	GPU	CPU	GPU	CPU	GPU
No Detection		**14.0**		0		**21.3**	
NumPy	YOLOv11n@480	13.5	10.3	0.9	**4.9**	10.5	11.2
	YOLOv11n@640	11.5	9.6	0.7	3.8	11.5	12.8
PyTorch	YOLOv11n@480	11.5	7.3	1.1	4.5	9.5	12.5
	YOLOv11n@640	10.0	8.5	0.5	3.6	12.0	12.5

Performance while a Duckiebot is running

Comparison of RAM, CPU, and GPU utilization while running.

Framework	Model	RAM (%)		CPU (%)		GPU (%)	
		CPU	GPU	CPU	GPU	CPU	GPU
No Detection		74.85		**93.84**		0	
Numpy	YOLOv11n@480	88.04	94.01	98.97	97.96	0	31.22
	YOLOv11n@640	**42.27**	80.36	99.08	97.40	0	40.83
PyTorch	YOLOv11n@480	79.66	92.40	99.02	97.66	0	**25.97**
	YOLOv11n@640	60.49	55.87	98.89	97.50	0	34.92

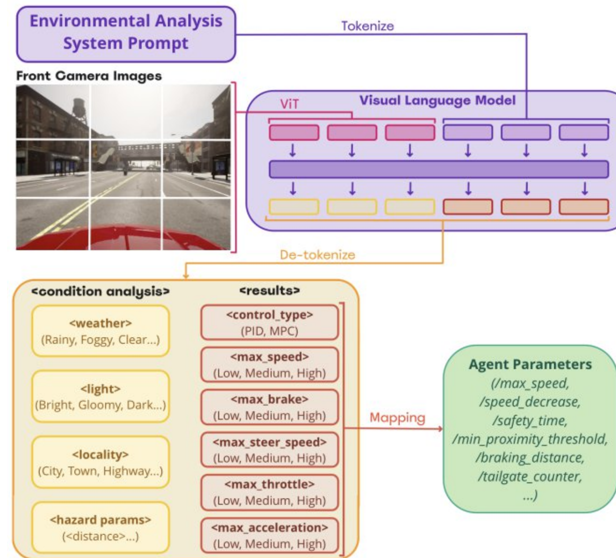
Performance while a Duckiebot is running

Comparison of CPU and GPU temperatures while running.

Framework	Model	CPU Temp (°C)		GPU Temp (°C)	
		CPU	GPU	CPU	GPU
No Detection		**33.87**		34.97	
Numpy	YOLOv11n@480	34.06	35.18	34.59	**34.37**
	YOLOv11n@640	34.79	36.15	35.37	35.19
PyTorch	YOLOv11n@480	34.85	35.69	35.59	34.57
	YOLOv11n@640	34.96	37.66	35.33	36.48

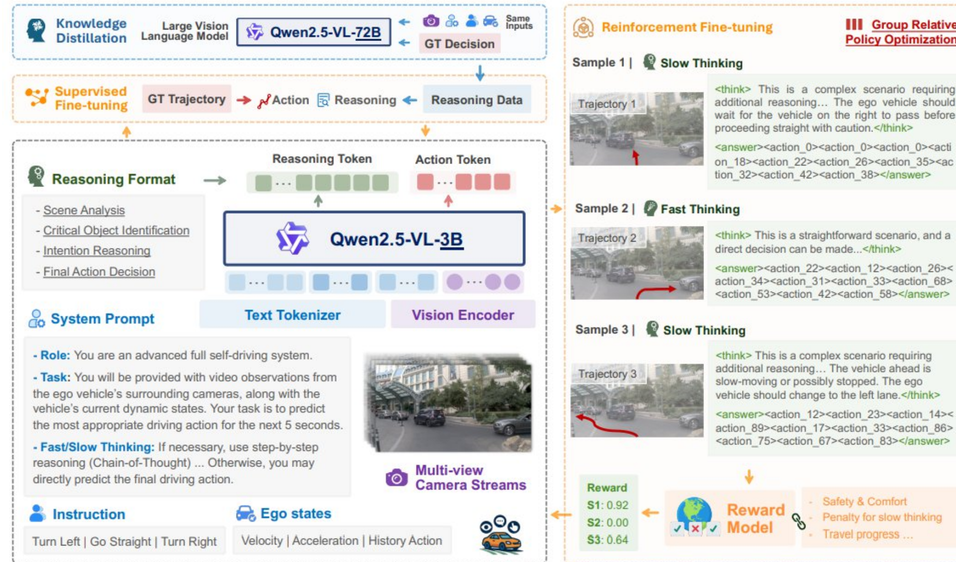
VLM/VLA Examples

VLM Example



VLM-Auto [6] (Oct 2, 2024)

VLA Example



AutoVLA [7] (Jun 16, 2025)