Boosting Perception

Dário Xavier Catarrinho

Layout: typeset by the author using LaTeX.
Cover illustration: Unknown artist

Boosting Perception

Comparative Analysis of Transformer, One-Stage and Two-Stage Based Object Detection Architectures for Real-Time Object Detection on the Booster K1 Platform

Dário Xavier Catarrinho
13678035

Bachelor thesis
Credits: 18 EC

Bachelor *Kunstmatige Intelligentie*

University of Amsterdam
Faculty of Science
Science Park 904
1098 XH Amsterdam

Supervisor
Dr. A. Visser

Informatics Institute
Faculty of Science
University of Amsterdam
Science Park 900
1098 XH Amsterdam

Semester 1, 2025-2026

Abstract

The transition of the RoboCup Standard Platform League and Kid-Size League into the unified Humanoid League presents significant challenges and opportunities for autonomous robotic football. With the introduction of the Booster K1 robot, equipped with the NVIDIA Jetson Orin AGX, the computational constraints that historically limited vision systems to heuristic-based approaches or lightweight CNNs have been lifted. This thesis investigates the feasibility of deploying state-of-the-art Deep Learning object detection architectures on NVIDIA Jetson hardware to address the latency-accuracy trade-off in dynamic robotic environments.

A comparative benchmarking analysis was conducted on six distinct architectures representing three design families: Two-Stage detectors (Faster R-CNN), Single-Stage detectors (SSD, YOLOv8, YOLO11), and Transformer-based detectors (RF-DETR, D-FINE). To ensure competitive viability, a strict real-time inference threshold of 33ms (30 FPS) was established. The models were evaluated using a proxy strategy on the NVIDIA Jetson Orin Nano (4GB), utilizing a dataset of 2,510 images sourced from diverse RoboCup domains to ensure generalization. All models were optimized using the TensorRT framework with FP16 quantization.

The results demonstrate a shift away from single-stage CNN models, towards Transformer based architectures. The Transformer-based **D-FINE-S** architecture emerged as the optimal solution, achieving a state-of-the-art Mean Average Precision (**mAP@.50**) of **0.932**, surpassing the industry-standard YOLOv8-S (0.880) and YOLO11-S (0.881). Furthermore, D-FINE-S demonstrated superior efficiency with an end-to-end latency of **29.9ms**, effectively eliminating the post-processing bottlenecks associated with Non-Maximum Suppression (NMS) in dense game scenarios. These findings confirm that modern Transformer architectures not only meet the rigorous real-time constraints of the RoboCup Humanoid League but also offer a significant accuracy advantage over traditional CNN-based baselines, providing a robust foundation for the next generation of autonomous soccer robots.

Contents

1 Introduction **4**
- 1.1 Object Detection . . . 4
- 1.2 RoboCup & Standard Platform League . . . 4
- 1.3 Humanoid League & Booster K1 . . . 5
- 1.4 Research Question . . . 5
- 1.5 Thesis Outline . . . 6

2 Theoretical Background **7**
- 2.1 Hardware comparison . . . 7
- 2.2 Selected models . . . 8
  - 2.2.1 Two-Stage Detection . . . 8
  - 2.2.2 Single-Stage Detection . . . 9
  - 2.2.3 Transformer-based detection . . . 11
- 2.3 Metrics for Object Detection . . . 13
  - 2.3.1 Intersection over Union (IoU) . . . 14
  - 2.3.2 Mean Average Precision (mAP) . . . 14

3 Method **15**
- 3.1 Selection of Model Architectures . . . 15
  - 3.1.1 Selection of Model Scale . . . 15
  - 3.1.2 Faster R-CNN . . . 15
  - 3.1.3 SSD . . . 16
  - 3.1.4 YOLOv8 & YOLO11 . . . 16
  - 3.1.5 RF-DETR . . . 16
  - 3.1.6 D-FINE . . . 16
  - 3.1.7 Model Overview . . . 17
- 3.2 Data Annotation & Organization . . . 17
  - 3.2.1 Annotation Process . . . 17
  - 3.2.2 CVAT-Pipeline . . . 17
  - 3.2.3 Unified Directory Structure . . . 18

3.3 Transfer Learning & Training Strategy . . . 18
3.3.1 Model Initialization . . . 18
3.3.2 Convergence Criteria . . . 19
3.3.3 Checkpointing Strategy . . . 19
3.4 Deployment & Optimization . . . 19
3.4.1 Proxy Benchmarking . . . 20
3.5 Evaluation . . . 20
3.5.1 Accuracy: Mean Average Precision . . . 20
3.5.2 End-to-End Latency . . . 21

4 Experiments and Results **22**
4.1 Experimental Setup . . . 22
4.1.1 Dataset breakdown . . . 22
4.1.2 Train-Test Splits . . . 25
4.1.3 Training Setup . . . 26
4.1.4 Inference Testing Setup . . . 26
4.2 Results . . . 27
4.3 Qualitative Analysis . . . 29
4.3.1 Legacy Models: Faster R-CNN and SSD . . . 29
4.3.2 YOLO models . . . 30
4.3.3 Transformer Models . . . 31

5 Conclusion & Discussion **32**
5.1 Limitations . . . 33
5.2 Future Work . . . 33

6 Acknowledgements **34**

A Attempted Models **35**
A.1 Cascade R-CNN . . . 35
A.2 DEYO . . . 35

B Experiment Results **37**
B.1 YOLOv8 Inference Study . . . 37
B.2 Training Plots . . . 38
B.3 Model Accuracy Score Tables . . . 41
B.4 Model Outputs & Ground Truths . . . 42

C Figures **44**

Chapter 1

Introduction

1.1 Object Detection

Object detection, the ability to localize and recognize entities within an image, is a crucial part of the perception and sensory input of autonomous systems. While the field of computer vision has achieved near-human performance in static environments, deploying these algorithms on mobile platforms has presented its own set of challenges. Constraints like limited computational resources, power consumption and real time latency have offered a compelling optimization balance to explore, where speed and accuracy get delicately balanced.

1.2 RoboCup & Standard Platform League

The RoboCup federation was established in 1997, with its original goal of fielding a team of robots capable of defeating the human football World Cup champions in 2050. This innovation is driven by the yearly competition, in which teams from all over the world compete in various leagues, each tackling various technical challenges.

Historically, the Standard Platform League (SPL) has served as a benchmark for software development under constrained conditions, requiring teams to use identical hardware: the Aldebaran NAO robot. While this standardization drove annual improvements in optimal hardware utilization on the computationally limited NAO platform, the NAO's aging processor began to limit vision processing. To this day, teams are forced to rely on algorithms purpose-built for aging hardware, using classic techniques such as scanlines and color thresholding (Ruiter et al., 2024) and pruned SSDs (Poppinga & Laue, 2019), rather than leveraging state-of-the-art research.

1.3 Humanoid League & Booster K1

In 2025, the SPL and Kid-Size league merged into a single new league: The RoboCup Humanoid League. With the new league's shift in hardware regulations, the whIRLwind Amsterdam team (formerly known as the Dutch Nao Team) must adapt by transitioning to new robots. For this new line of robots, the team selected the Booster K1 robot from Booster Robotics, equipped with the NVIDIA Jetson Orin AGX, a System-on-Module (SoM) featuring a dedicated GPU and optimizations for autonomous systems.

Figure 1.1: A picture of a K1 match, held at the RoboCup in Brazil, 2025. Image captured with the onboard camera of the K1.

This new hardware now permits the deployment of full-scale, state-of-the-art Deep Learning object detectors. For example, at the World Humanoid Games in Beijing, the team used an unrefined implementation of the YOLOv8 model for ball detection, a capability previously impossible to apply to NAO robots due to hardware limitations. This resulted in a usable, though clearly unoptimized, end product. With these new capabilities, the question arises: what object detection model truly offers the most efficient solution for object detection in RoboCup football conditions?

1.4 Research Question

The objective of this thesis is to provide a comprehensive benchmarking analysis of modern object detection architectures to identify the optimal solution for the Booster K1

platform. Specifically, the research aims to answer the following question:

> *Which of the six object detection models in this analysis strikes the best balance between accuracy and speed on the Jetson Orin platform for K1 usage?*

In this thesis will be elaborated on how this performance is determined.

1.5 Thesis Outline

The remainder of this thesis is structured as follows:

- **Chapter 2 (Theoretical Background)** presents the selected models in this analysis, and goes into detail on the underlying mechanisms that set these models apart.
- **Chapter 3 (Method)** gives an overview of the approach, design choices, and evaluation metrics used to conduct this study.
- **Chapter 4 (Experimental Setup & Results)** presents the dataset, training parameters used, and the training results, after which the models are evaluated and compared in their outputs.
- **Chapter 5 (Conclusion & Discussion)** interpret the findings, answer the research question, evaluate the process of the study, and offer suggestions for future work.

Chapter 2

Theoretical Background

2.1 Hardware comparison

For over a decade, the robot of choice for the RoboCup Standard Platform League (SPL) has been the various iterations of the Aldebaran NAO, going up to the most recent v6 model. This NAO is powered by a 4-core Intel Atom E3845 CPU. This CPU, released in 2013, operates at a base frequency of 1.91 GHz with a Thermal Design Power of 10W.

This design forced a CPU-centred approach, with all processes such as locomotion, perception, and behaviour handled on said CPU simultaneously (de Jong et al., 2023), which in turn necessitated the use of lightweight ball proposal and classification algorithms with heavy filtering (Ruiter et al., 2024) to mitigate this architectural bottleneck. These algorithms were highly sensitive to changing lighting conditions, a problem that, despite efforts to mitigate (Monté et al., 2023), persisted due to the previously stated hardware limitations.

The Booster K1 Professional Edition delivers a significant hardware advancement over the NAO v6. The K1, powered by the NVIDIA Jetson AGX Orin 32GB, uses an NVIDIA Ampere GPU with 1792 CUDA cores and 56 Tensor cores, providing 200 TOPS of AI performance. Paired with an 8-core NVIDIA Arm Cortex CPU running at a maximum frequency of 2.2 GHz, this platform follows the modern System-on-Module (SoM) approach of tightly integrating all components to optimise AI computation. This platform also comes with dedicated Deep Learning Accelerators (DLA 2.0), Vision Accelerators (PVA v2), Video Encoders and Video Decoders, leading to improved performance with hardware acceleration.

The implications of this disparity in hardware capabilities are significant for object detection. With the hardware-acceleration DLA and PVA provide, and the dedicated NVIDIA Ampere GPU, the CPU can be substantially offloaded. This is in stark contrast

to the CPU bottleneck inherent to the NAO architecture. This specialised hardware enables the deployment of complex, deep learning models that rely on massive parallel matrix multiplications, a computational task previously infeasible on a CPU alone.

2.2 Selected models

The six models selected in this thesis can be grouped into 3 families, based on 3 prominent model architectures in the field of object detection over the past few years (Zou et al., 2023). These model architectures are Two-Stage Detection, Single-Stage Detection and Transformers. While it can be argued that transformers fall under the category of single-stage detection, their model architecture differs substantially from traditional CNN-based single-stage detection models that have been the standard in recent years, such as YOLO (Redmon et al., 2016). Due to these transformer models also being the most recent development in the area (Zou et al., 2023), they are the straightforward choice. Below, the models will be described globally by family, with their distinctions separated into their respective sections.

2.2.1 Two-Stage Detection

The two-stage detection framework, more widely adopted following the seminal R-CNN (Girshick et al., 2014), established the foundations for the methodology the models in this family follow. Two-stage detectors subdivide the detection problem into two steps: the localisation and the classification step. This approach prioritises accuracy, while sacrificing computational latency when compared to more modern models and methods (Zou et al., 2023).

The objective of the first stage is to filter the possible objects from their image background. This can be achieved through the use of various heuristic algorithms (Girshick et al., 2014). In the original R-CNN model, this was achieved with Selective Search. This heuristic algorithm segments and groups an image into regions of interest using a bottom-up approach, grouping regions based on colour, texture, size, and shape compatibility. This algorithm outputs a set of region proposals to pass on to the second stage.

In the second stage, the proposed regions undergo simultaneous class classification and bounding-box regression to refine and optimise the bounding-box fit. This approach leads to high precision, but takes a significant latency hit as a consequence of the split process.

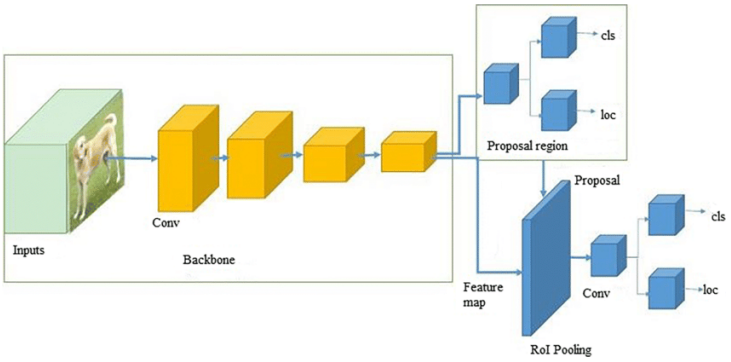

Figure 2.1: Architecture of a two-stage detection model (Pal et al., 2021).

Faster R-CNN

After the R-CNN model underwent significant optimisation with the Fast R-CNN model (Girshick, 2015), which restructured the architecture to take a single image with proposal coordinates as input, rather than processing 2k proposal images sequentially, it drastically sped up training and fine-tuning. With the bottleneck now lying on the region proposal methods, this problem was addressed with the Faster R-CNN model (Ren et al., 2017), which introduced a Region Proposal Network (RPN). This approach utilises a fully convolutional network, where the backbone shares the features with the detection head. The introduction and implementation of anchor boxes, predefined boxes of various scales and aspect ratios, sped up the model significantly by predicting both object bounds and 'objectness' scores per image patch. In the scope of this analysis, Faster R-CNN will offer a high-precision baseline, while sacrificing on inference speed.

2.2.2 Single-Stage Detection

In contrast to the "proposal-refinement" approach of the two-stage detectors, the single-stage detection architecture treats object detection as a single-step problem. With the introduction of the YOLO (You Only Look Once) architecture (Redmon et al., 2016), single-stage models became a significant new category in object detection. Single-stage detection models have since made breakthroughs, for example, in subsequent iterations (which will be discussed in this section). By removing the region proposal step and processing the input image in a single forward pass of a convolutional neural network, it is capable of calculating bounding box coordinates and class probabilities simultaneously. This significantly simplifies the architecture, leading to stronger inference throughput. This, in turn, made these models suitable for real-time object detection applications. This speed comes at the cost of accuracy, with early models struggling to match the accuracy of two-stage detectors. It also posed problems in training, due to the strong class imbalance between foreground and background instances (Lin et al., 2020).

SSD (Single Shot MultiBox Detector)

While the YOLO architecture established the concept and advantages of single-stage detection, it showed limited performance in detection of small objects. The SSD (Single Shot MultiBox Detector) (Liu et al., 2016) addressed this limitation by introducing multi-scale feature maps. By applying anchor-based bounding boxes with predefined aspect ratios at multiple scales, this allowed the model to perform well on both larger and smaller objects. With multi-scale feature maps, this architecture is capable of detecting larger objects in deeper layers, utilising semantic richness, and smaller objects in shallower layers with higher spatial resolution. Adaptations of this model have been implemented in the RoboCup context over the past years (Poppinga & Laue, 2019), (Ruiter et al., 2024) and thus provide a clear benchmark for single-stage detection in this analysis.

YOLOv8

The eighth iteration of the YOLO architecture, YOLOv8 (Jocher et al., 2023), releases the anchor-based mechanics in favour of an anchor-free approach, which predicts the centre of the object and the dimensions of the bounding box. This adaptation minimises the number of box predictions, speeding up NMS. This was combined with replacing the older C3 modules with the C2f (Cross-Stage Partial bottleneck with two convolutions) module in the Feature Pyramid Network (FPN) of the backbone to maximise gradient flow and feature integration (Sohan et al., 2024). These advancements made the YOLOv8 model the industrial standard for real-time object detection. The Dutch Nao Team implemented this model on the Booster T1 robot (with the same NVIDIA Jetson Orin AGX) as a simple object-detection algorithm, making it a familiar baseline for this analysis.

YOLO11

The most recent iteration in the YOLO series, YOLOv11 (Jocher & Qiu, 2024), iterates further on the YOLO formula. Improving on feature extraction in the backbone by replacing the C2f modules introduced in YOLOv8 with C3k2 modules, which are both faster and more efficient, thus enhancing overall performance. The neck of the architecture combines features from different scales, enabling the model to capture multi-scale information more effectively. The implementation of C2PSA modules introduced the concept of attention mechanisms (which will be elaborated on in section 2.2.3) to the YOLO architecture, improving spatial attention in feature maps (Khanam & Hussain, 2024). This model represents the state of the art model within the single-stage detection family for this analysis.

2.2.3 Transformer-based detection

The introduction of the Transformer architecture to object detection marks yet another shift, away from CNN-based architectures. Where CNN models rely on sliding windows to extract local features and patterns, transformers use Self-Attention Mechanisms (Vaswani et al., 2023) to extract the global context of a scene and its features by modelling its long-range dependencies, instead of focusing on local features to form an understanding of the image.

This concept was first applied to images with the DETR (DEtection TRansformer) model (Carion et al., 2020). This model leveraged the transformer architecture to form a global context of the image, after which a fixed set of learned object queries generated a set of object or background proposals, using attention to interact with the encoded image features to form an understanding of what is being proposed. Every query produces a query embedding that either proposes a single object or a "no object" classification, turning object detection into a direct set prediction problem. With a bipartite matching loss (Hungarian algorithm), unique matches between objects and ground truth boxes are forced during training, eliminating the need for handcrafted features like anchor boxes and NMS, since no redundant boxes are generated. However, this original DETR model suffered from slow convergence rates and difficulties with poor performance on small objects, which make application in real time robotics complicated.

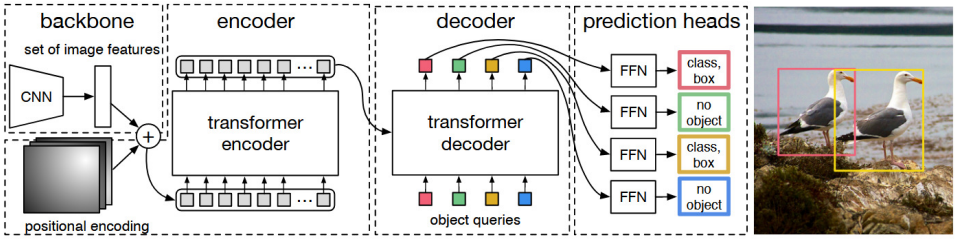

Figure 2.2: The original DETR architecture, as presented by Carion et al. (2020)

Since the introduction of DETR, many improvements have been made to address the issues stated earlier. The introduction of Deformable DETR (Zhu et al., 2021) addressed the slow convergence rates and issues with small objects by replacing the Global Attention mechanism (that considered every single pixel in an image) with Deformable Attention. Instead of attending to every single pixel equally, Deformable Attention uses reference points in the encoder and decoder to limit its attention domain to local space and exponentially scale down the computational cost of applying its attention mechanisms. This, in turn, enabled the use of multi-scale feature maps to address the challenges of

detecting small objects, improving DETR model accuracy while converging in fewer than half the number of epochs.

After further improvements on the DETR formula, efforts were directed towards real-time usage of Vision Transformers (ViT). RT-DETR (Real-Time DETR) (Zhao et al., 2024) and LW-DETR (Light-Weight DETR) (Chen et al., 2024) both adopted approaches to building model architectures suitable for real-time use. RT-DETR sped up computation by replacing the ViT encoder with an efficient hybrid encoder, leveraging both the strengths of an attention layer (applied only at the highest level, on the smallest feature maps to preserve computation) and the efficiency of CNN-based Cross-scale Feature Fusion for low-level feature maps to achieve real-time feasibility. LW-DETR retains a ViT encoder, simplifying and optimising heavily by switching between global attention mechanisms and local windows to speed up computations. The decoder leverages the aforementioned deformable cross-attention to achieve similar levels of speed and accuracy to those of the RT-DETR model. The models used in this comparative analysis have built upon both of these models, respectively, to claim state-of-the-art inference and accuracy scores upon presentation, making them viable for use in a RoboCup setting on the Booster K1.

RF-DETR

The RF-DETR model (Robinson et al., 2025) builds upon the foundations of LW-DETR, by utilising its architecture and exchanging its CAEv2 backbone (Zhang et al., 2022) for DINOv2 (Oquab et al., 2024), a self-supervised foundation model that has a strong understanding of visual features, thus being applicable on a wide variety of different smaller datasets to facilitate finetuning. Though the DINOv2 backbone is heavier than the CAEv2 model, it compensates by enabling higher-quality feature extraction, allowing for a more lightweight encoder and decoder head. Utilising NAS (Neural Architecture Search) (Zoph & Le, 2017) to optimise the architecture of the model for the most efficient accuracy-latency trade-off, these optimisations led to state-of-the-art accuracy, while maintaining inference latency equal to similar models in its respective classes.

D-FINE

The D-FINE model (Peng et al., 2024) utilises the speed of RT-DETR's CNN-based, efficient hybrid encoder and backbone to address its accuracy shortcomings by improving the decoder's ability to accurately localise objects and draw bounding boxes. This is achieved with two components: Fine-grained Distribution Refinement (FDR) and Global Optimal Localisation Self-Distillation (GO-LSD). With FDR, traditional bounding box regression methods are replaced with an iteratively optimising system of bounding box edge distributions, allowing for incremental improvements throughout the decoder layers. This way, uncertainty can be modelled in the earlier layers of the decoder and more

accurately improved upon during training. With GO-LSD, localisation knowledge from the last decoder layer is distilled into shallower layers as a teacher-student system. This way, soft targets are produced for the shallower layers, improving stability and localisation performance in these shallower layers. D-FINE also includes adapted modules and operations from the RT-DETR model, aiming to make it more efficient and lightweight. The aforementioned improvements mitigate any performance losses stemming from these optimisations, enabling D-FINE to achieve state-of-the-art accuracy and latency.

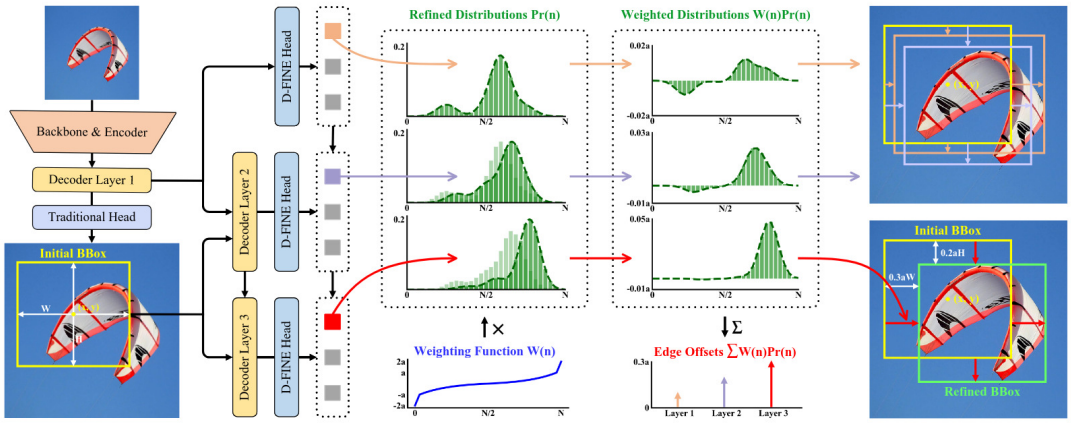

Figure 2.3: A visualisation of the iterative bounding-box refinement with FDR.

2.3 Metrics for Object Detection

To measure and evaluate the performance of trained object detection models, we need to use commonly used performance metrics. Below is an overview of these metrics.

2.3.1 Intersection over Union (IoU)

Intersection over Union (IoU) is the metric used to describe how well an object detection model is able to predict the bounding box placement and dimensions in an image. The IoU is determined by comparing the ground truth bounding box to the prediction bounding box, where the Area of Overlap is divided by the Area of Union of the two bounding boxes (as shown in Figure 2.4). This results in a value between 0 and 1, with 1 indicating perfect overlap and 0 indicating that the prediction does not align with the ground truth. In this thesis, the IoU serves as a threshold for computing mean Average Precision (mAP), as described in Section 2.3.2.

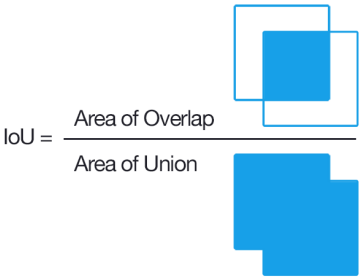

Figure 2.4: The IoU is calculated by dividing the Area of Overlap by the Area of Union between the bounding box prediction and the ground truth.

2.3.2 Mean Average Precision (mAP)

To quantify the model's accuracy across all classes, we use mean Average Precision (mAP). First, the Average Precision (AP) is calculated per class based on the number of correctly predicted boxes compared to the total number of predicted boxes:

$$\text{Precision} = \frac{\text{TP}}{\text{Total number of boxes}} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{2.1}$$

Where the True Positives (TP) stand for the number of boxes that are correctly and accurately predicted, and the False Positives (FP) stand for the number of boxes that do not align with their ground truth box. Whether a prediction is considered a TP or an FP is determined by the IoU threshold used. The mAP averages the AP scores across all classes to determine the overall accuracy of an object detection model at the specified IoU threshold. In object detection literature and model presentation papers, the common IoU thresholds used are mAP@.50 (where a prediction with 50% overlap or higher counts as a true positive), mAP@.75 (for a more strict prediction threshold) and mAP@[.50:.95] (where the average mAP scores are taken from mAP@.50 to mAP@.95 at intervals of 0.05. This provides a more accurate overview of how well a model performs as positioning requirements increase). Usage of the mAP metric will be elaborated on in section 3.5

Chapter 3

Method

3.1 Selection of Model Architectures

Building on the architectural families selected in Section 2.2, this section details the specific model configurations, backbones, and scaling variants used in the comparative analysis.

3.1.1 Selection of Model Scale

For the modern model architectures in this analysis (YOLOv8, YOLO11, RF-DETR and D-FINE), various sizes were available, ranging from Nano to XXL. Due to constraints in the running of inference testing on an NVIDIA Jetson Orin Nano 4GB (see Section 4.1.4 for detailed reasoning), a small-scale study was conducted based on the various versions of YOLOv8, a widely considered industry standard due to its reliability and speed (Ultralytics, n.d.). For this study, the aim was to identify the largest YOLOv8 model size to break below the 33ms threshold without NMS (thus capable of running at 30 FPS on the Jetson Orin Nano). The three smallest YOLOv8 models, Nano (n), Small (s), and Medium (m), were evaluated, and it was concluded (see Section B.1) that YOLOv8-S was the first model capable of running below this 33ms inference threshold. This guaranteed baseline performance on the Orin Nano, which would scale accordingly with the increase in performance towards the NVIDIA Jetson AGX Orin. Consequently, the small variant was selected for model configuration for said modern model architectures to ensure a consistent comparison.

3.1.2 Faster R-CNN

For the two-stage model setup, Faster R-CNN was configured according to the presentation paper (Ren et al., 2017), (“Faster R-CNN — Torchvision Main Documentation”, n.d.),

and initialised with the ResNet-50-FPN provided by the TorchVision library. This will provide a stable baseline for a model that has long delivered state-of-the-art accuracy. The ResNet-50-FPN also automatically handles varying input sizes by resizing input images to a standard size of 800x800, eliminating the need for manual preprocessing in the dataloader.

3.1.3 SSD

The SSD model configuration was also configured according to its presentation paper (Liu et al., 2016) for an input size of 300x300, with a VGG16 backbone ("SSD — Torchvision Main Documentation", n.d.). This image too handles automatic resizing in the backbone, eliminating the need for manual image resizing or batch sorting in the dataloader.

3.1.4 YOLOv8 & YOLO11

The YOLOv8 and YOLO11 models were configured in the Small variant (as stated in section 3.1.1). The backbone remained unchanged from the default CSPDarknet backbone for YOLOv8-S (Yaseen, 2024), and the CSPDarknet with C3k2 for YOLO11-S (Khanam & Hussain, 2024). Both models take an input size of 640x640 (consistent across the YOLO family), with automatic resizing applied within the built-in training function.

3.1.5 RF-DETR

For the RF-DETR model, the variant was kept consistent by selecting the Small variant. This model size utilises the DINOv2-S backbone and takes 512x512 input, with automatic image resizing applied in the built-in training function.

3.1.6 D-FINE

The D-FINE model in the Small variant is initialised with a CNN-based HGNetv2-B0 backbone configuration. Automatic resizing of input images occurs in the default training loop, resizing them to a 640x640 input.

3.1.7 Model Overview

Table 3.1: Model Overview

Model	Param size	Variant/Size	Backbone	Input Size
Faster R-CNN	41.7M	Standard	ResNet-50	800x800
SSD	35.6M	Standard	VGG16	300x300
YOLOv8-S	11.2M	Small	CSPDarknet	640x640
YOLO11-S	9.4M	Small	CSPDarknet (C3k2)	640x640
RF-DETR-S	32.1M	Small	DINOv2-S	512x512
D-FINE-S	10.2M	Small	HGNetv2-B0	640x640

3.2 Data Annotation & Organization

To fine-tune our models for RoboCup competitive usability and generalization, a robust and diverse dataset is crucial. For this thesis, a diverse dataset of images from different leagues, lighting conditions, and angles was compiled (a more in-depth analysis of the dataset is presented in Section 4.1.1). This dataset requires manual, accurate annotation, with precise ground-truth labels to distinguish key game elements. The following sections detail the tooling, personnel, and organizational strategies employed to generate this dataset.

3.2.1 Annotation Process

The raw image data were annotated using the Computer Vision Annotation Tool (CVAT), a web-based, open-source platform for bounding box annotation. This platform enables us to assign annotation tasks to annotators, thereby facilitating the management of open annotation tasks and streamlining the annotation process. To ensure ground-truth quality, particularly under challenging conditions such as motion blur or occluded objects, the annotation process was conducted exclusively by members of the Dutch Nao Team. Using team members enabled the application of domain expertise and experience, mitigating the risks of potential miss-annotations stemming from crowd-sourced annotation work, ensuring consistent quality.

3.2.2 CVAT-Pipeline

Extracting annotations and datasets from CVAT often results in fragmented annotation files, annotation-format mismatches, and file merging. To facilitate and streamline this workflow, a custom command-line tool was developed within the team. This tool handled

downloading and merging dataset and annotation files, formatting them to the COCO JSON format required by most models in this analysis, and creating train-validation splits.

3.2.3 Unified Directory Structure

A practical challenge of this analysis is the conflicting requirements for the file-tree structure, the placement of annotation files, and the varying annotation formats. For instance, the YOLO family models require annotations in individual `.txt` labels for every image in a 0-indexed class-id structure and `.yaml` file, and the `labels/` folder right above the image directory, whereas RF-DETR requires the annotations to be in a file, precisely called `_annotations.coco.json`, in the directory right above the `images/` folder. These COCO JSON annotations required a 0-indexed class-id list, with an explicit background class at index 0. Finally, the remaining models (Faster R-CNN, SSD, and D-FINE) required a 1-indexed annotation format and allowed specifying paths to the annotation file and image folder separately.

To avoid redundant data, the following unified directory structure was designed to accommodate all these requirements. The directory structure in Figure C.1 allowed for seamless switching between models without the necessity for data duplication, and enabled models to train sequentially overnight.

3.3 Transfer Learning & Training Strategy

3.3.1 Model Initialization

Given the limited size of the target dataset (2510 images, analyzed in depth in Section 4.1.1) and the complexity of the architectures used in this analysis, training from random initialization was deemed infeasible due to slow convergence. To mitigate this, Transfer Learning was employed by initializing all architectures on weights pre-trained with the COCO (Common Objects in Context) dataset ("COCO - Common Objects in Context", n.d.). This initialization enables backbone networks to leverage robust learned features from this large-scale dataset, allowing the fine-tuning process to primarily adapt the high-level layers to our dataset classes (like balls, goalposts, and line intersections). Since the target dataset does not contain specialized data, such as medical imaging or area-specific scans for detection, knowledge from the COCO dataset pre-training transferred well to our target dataset.

3.3.2 Convergence Criteria

Defining model convergence is critical to achieving equal comparisons for every model. Since the objective for every model is to maximize accuracy, our primary stopping criterion for the training process was the stabilization of the validation mAP@50 metric, measured after every epoch.

This approach, combined with the utilized checkpointing strategy (described in the next section), ensured optimal model performance on unseen data. This distinction proved particularly crucial for the YOLO model family, where training loss kept decreasing significantly, well after validation accuracy had plateaued (see figures B.1 and B.2 in the appendix).

3.3.3 Checkpointing Strategy

To ensure that the model's accuracy reflected the peak capability reported in the performance metrics, a best-checkpoint system was utilized. After each epoch, the models evaluated their accuracy on the validation set and logged their weights if they achieved an improvement in mAP@[.50:.95] over the best accuracy score. This system offered stability and protection against late-stage fluctuations or overfitting. These best-performing model versions were subsequently used for final model accuracy comparisons on the test set.

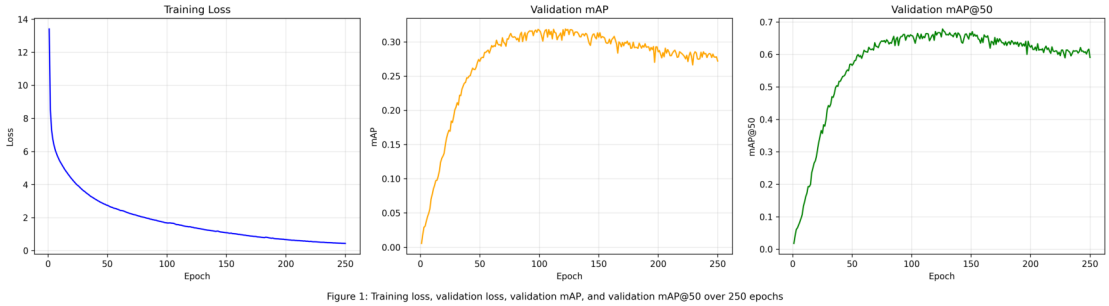

Figure 3.1: The training metrics for the SSD model, with drops in validation accuracy scores to indicate overfitting. The checkpoint strategy guaranteed that these effects were mitigated by benchmarking on the model weights of the best-performing epoch.

3.4 Deployment & Optimization

To evaluate our models under realistic RoboCup conditions, the models were separated from their training environments and optimized for inference on the NVIDIA Orin Architecture. By converting the trained PyTorch models to the Open Neural Network Exchange (ONNX) format ("ONNX | Home", n.d.), we decouple the model from its

PyTorch environment, allowing compilation of the models into TensorRT engines for optimization on the target hardware.

This compilation performs various optimizations to tailor the model specifically for the NVIDIA Orin GPU. Crucially, during this compilation, all models were quantized to Half-Precision (FP16). The Orin's Ampere architecture features dedicated Tensor cores optimized for FP16 operations, reducing memory bandwidth requirements by 50% compared to standard FP32 precision. This allows for significantly higher throughput while maintaining sufficient numerical stability for object detection tasks.

3.4.1 Proxy Benchmarking

A key challenge in this thesis was benchmarking for the Booster K1 robot (equipped with a Jetson Orin AGX) while using a Jetson Orin Nano as a hardware proxy, due to the absence of the K1 robot at the time of this thesis.

Since the Orin AGX and Orin Nano both share the same Ampere GPU architecture with the same TensorRT optimization backend, this allowed for the relative latency scores between the models to stay consistent. By constraining the model selection to fit the limited 4GB of unified memory on the Orin Nano, a performance floor was established that would scale well for guaranteed performance on the target robot platform, the Orin AGX.

3.5 Evaluation

To quantify the trade-off between detection performance and computational efficiency, two primary metrics were established: Mean Average Precision (mAP) for accuracy and End-to-End Latency for speed.

3.5.1 Accuracy: Mean Average Precision

Standard object detection protocols utilize the mAP metric, which calculates the mean of the recorded Average Precision (AP) scores over all classes.

For this study, two variations are reported:

- **mAP@[.50:.95]**: This rigorous metric takes the mean over the mAP scores from 0.5 to 0.95, at intervals of 0.05. This offers a balanced overview of both detection consistency and quality, to serve as the primary accuracy metric for this study.

- **mAP@.50**: This measure considers a detection a True Positive for an Intersection over Union with the ground truth of at least 0.5. In the context of RoboCup, consistent detection and rough localization are often more important than perfect bounding boxes, thereby providing a strong secondary metric for model performance.

3.5.2 End-to-End Latency

Raw inference time (a model's forward pass) often offers an insufficient measurement of real-world speed, particularly when comparing two-stage and single-stage models with transformer architectures. Therefore, we define end-to-end latency as our speed metric, as the sum of 3 components:

$$T_{total} = T_{pre} + T_{inf} + T_{post} \tag{3.1}$$

- **Pre-processing**: The time required to resize the input image and normalize pixel values.
- **Inference**: The execution time of a forward pass of the TensorRT engine on the GPU.
- **Post-processing**: Time required to decode outputs. Crucially, for the single-stage models, this includes NMS. For end-to-end transformer models, this component is theoretically negligible.

Chapter 4

Experiments and Results

4.1 Experimental Setup

4.1.1 Dataset breakdown

The model evaluation was conducted on a curated dataset of 2510 images, compiled from 9 datasets in total, that can be divided into 3 categories:

- **T1 data**, sourced by the whIRLwind team using the onboard camera from the K1, during and in between the Humanoid Games matches in Beijing, 2025.
- **NAO match data**, sourced from the HULKs team from Hamburg (Göttsch, 2025)
- **Kid-Size League data**, sourced from the Bit-Bots team during the RoboCup competitions in Nagoya and Sydney in 2017 and 2019, respectively (Bestmann et al., 2021)

Table 4.1: Detailed Dataset Composition and T1 (Robot) Class Availability

Dataset Name	**Source / Origin**	**Images**	**T1 Class Present**
*whIRLwind T1 Data (Booster T1) – **57.3% of Total***			
ice-ribbon-1	Humanoid Games Beijing 2025	264	✓
ice-ribbon-2	Humanoid Games Beijing 2025	341	✓
ice-ribbon-3	Humanoid Games Beijing 2025	497	✓
panda-eye-1	Humanoid Games Beijing 2025	337	✓
*HULKs Data (NAO - RoboCup SPL) – **36.8% of Total***			
hulks-top-1	RoboCup Brazil 2024	342	–
hulks-top-2	RoboCup Brazil 2024	266	–
hulks-top-3	RoboCup Brazil 2024	315	–
*BitBots Data (Kid-Size League) – **5.9% of Total***			
nagoya-1	RoboCup Nagoya	28	–
sydney-1	RoboCup Sydney	120	–
Total		**2,510**	**4 / 9**

As shown in the table above, T1 data is significantly more prevalent. This aligns with the dataset's objective: to create a dataset for K1 use with strong generalization across different lighting conditions and fields. With the additions of varying field marks and line visibility (shown in Figures 4.1 and 4.2), our models get prepared for any type of circumstance.

It is important to note that only T1 robots have been annotated; NAO robots and Kid-Size league robots are not labeled as classes in the dataset. This also implies that any detections of NAO robots or Kid-Size league robots will be classified as false positives.

In the figures shown below, a representative image has been sampled for each of the dataset source categories. While the simple images display clear line markings, stable lighting conditions and sharp images, the difficult conditions display the "worst-case scenarios" the models are faced with in training and validation.

Figure 4.1: 3 Examples of simple images, little to no occlusion of objects to be detected, stable light conditions with decent exposure.

Figure 4.2: 3 Examples of difficult lighting conditions, "distracting" objects on the sidelines, and long-range detections.

4.1.2 Train-Test Splits

The data was divided into three stratified splits in the 70-15-15 ratio: train, validation, and test. The data splitting was executed by the cvat-pipeline tool, ensuring representative model class distributions over the three data splits:

Table 4.2: Overview of dataset splits

Split	**# Images**	**# Objects**	**Images without Objects (%)**
Train	1,757	10,615	13.66%
Validation	376	2,325	10.64%
Test	377	2,292	11.94%
Total	2,510	15,232	–

Table 4.3: Object class distribution per dataset split

Class	**Train**	**Validation**	**Test**
T1	3,328	728	724
L-Intersections	2,244	475	474
T-Intersections	1,812	401	391
X-Intersections	993	233	226
Goalpost	1,088	220	252
Ball	589	143	114
Penalty Marks	561	125	111
Total	10,615	2,325	2,292

Table 4.4: Relative class distribution (%) per dataset split

Class	**Train (%)**	**Validation (%)**	**Test (%)**
T1	31.35	31.31	31.59
L-Intersections	21.14	20.43	20.68
T-Intersections	17.07	17.25	17.06
X-Intersections	9.35	10.02	9.86
Goalpost	10.25	9.46	10.99
Ball	5.55	6.15	4.97
Penalty Marks	5.28	5.38	4.84

4.1.3 Training Setup

To ensure a rigorous and fair comparison across all models, training was kept as standardized as possible. All models were trained to convergence (training plots for all models can be found in Appendix Section B.2). All models were trained on the same workstation, equipped with the following specifications:

Table 4.5: Training Workstation Hardware Specifications

Component	Specification
CPU	Intel i9-9900 (8 cores, 16 threads) @ 5GHz
GPU	NVIDIA GeForce RTX 2080Ti (12GB VRAM)
RAM	64GB
OS	Ubuntu 20.04 LTS
CUDA Version	12.7

The models were trained as close to default hyperparameters as possible, according to the following conditions:

Table 4.6: Training Hyperparameters and Convergence Results

Model Architecture	Optimizer	Batch Size	Total Epochs	Best Epoch
Faster R-CNN (ResNet50)	SGD	8	59	8
SSD (VGG16)	SGD	32	250	122
YOLOv8-S	Auto	16	350	222
YOLO11-S	Auto	8	350	315
RF-DETR-S	AdamW	4 (grad accum: 4)	200	175
D-FINE-S	AdamW	8	237	207

For the YOLO models, the default optimizer is set to Auto, which automatically selects the optimal optimizer given the specified hyperparameters and model configuration. Retrieving the selected optimizer post-hoc proved impossible.

4.1.4 Inference Testing Setup

For inference benchmarking, we used the Jetson Orin Nano as a proxy for the Jetson Orin AGX. Possessing the same architecture (NVIDIA Ampere), this guaranteed equal relative performance scaling, ensuring that the efficiency rankings established on the Nano would directly translate to the target AGX platform.

To prepare models for inference, the trained PyTorch checkpoints were first exported to the Open Neural Network Exchange (ONNX) format, decoupling the model graph from the training framework. Subsequently, these intermediate graphs were compiled into TensorRT engines using the NVIDIA TensorRT SDK (v10.1.1) on the Jetson. This compilation process applied graph-level optimizations, targeted at the Jetson's Ampere GPU architecture. Crucially, during this compilation process, Half-Precision (FP16) quantization was applied, reducing memory bandwidth by 50% while maintaining numerical stability.

End-to-end inference was measured, with 50 warm-up runs to stabilize GPU clock frequencies, and averaged over 1000 iterations.

4.2 Results

Table 4.7: Model Overview

Model	Param size	Mean Inference Jetson (FP16)	VRAM usage inference	mAP@.50:.95	mAP@.50
Faster R-CNN	41.7M	135.3ms	266.3 MiB	0.449	0.818
SSD	35.6M	19.4ms	23.5 MiB	0.281	0.607
YOLOv8-S	11.2M	30.3ms	17 MiB	0.548	0.880
YOLO11-S	9.4M	31.7ms	18.2 MiB	0.552	0.881
RF-DETR-S	32.1M	31.2ms	15.3 MiB	0.582	0.918
D-FINE-S	10M	29.9ms	31.3 MiB	0.595	0.932

This overview indicates both speed and accuracy advantages for Transformer models. Curiously, the YOLO models perform extremely similarly, with the legacy R-CNN and SSD models showing their limitations in optimizing the speed-accuracy trade-off required for robotics. While Faster R-CNN maintains respectable detection performance (mAP@.50: 0.818), its heavy two-stage architecture does not transfer well to real-time detection, with an inference speed of 135ms, exceeding our 33ms threshold for 30 FPS throughput. Conversely, SSD achieves the lowest latency of all models (19.4ms) but delivers suboptimal results, with an mAP@.50 of 0.607.

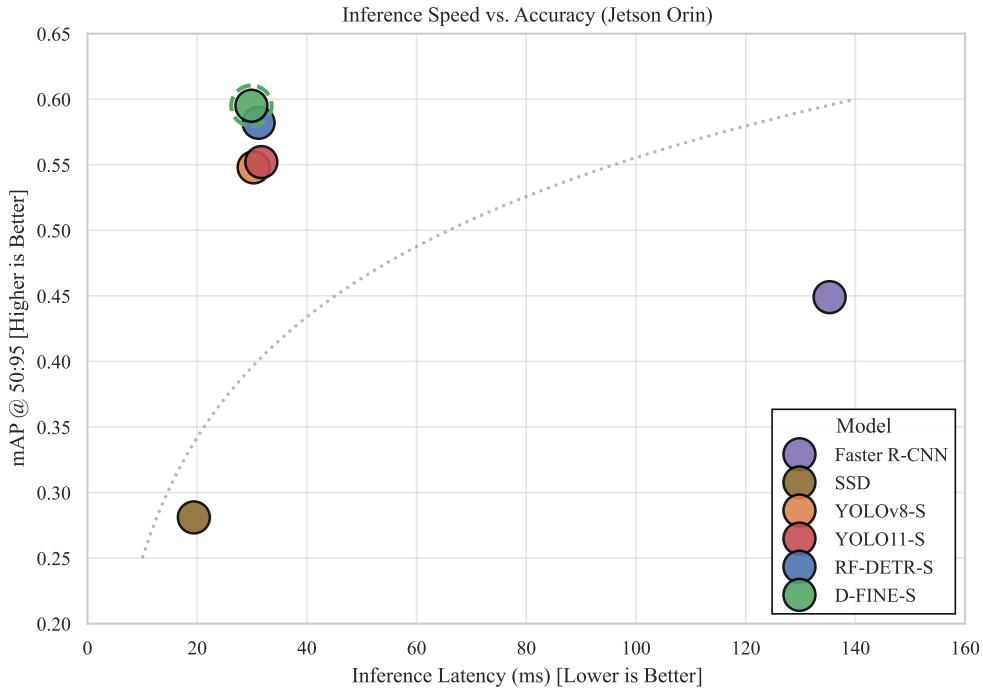

Figure 4.3: The plotted speed-accuracy trade-off.

In the plot above, the results are plotted against each other, with the dotted line indicating the speed-accuracy trade-off direction. As latency increases, accuracy is expected to improve. From this plot, it becomes visible how the SSD and Faster R-CNN model seemingly underperform against its modern counterparts, and how both transformer models break the tradeoff trend by improving on accuracy, but maintaining latency on par with the YOLO family models.

For the YOLO models, a stagnation in improvement from version 8 to 11 is visible. While performing similarly on speed, the cross-attention mechanisms in the YOLO11 backbone (which should improve detections with partially occluded or small objects) do not seem to get fully leveraged, only delivering 0.4%+ improvements on mAP@[.50:.95].

For the transformer models, D-FINE-S achieved both the highest precision in mAP@.50 (0.932) and mAP@[.50:.95] (0.595), while maintaining the lowest latency of all the modern models in this analysis. This contrasts with the benchmarks on the COCO datasets presented by RF-DETR, which indicated superior accuracy for the RF-DETR model (mAP@[.50:.95] of 0.506 for D-FINE-S, versus 0.529 for RF-DETR)(Robinson et al., 2025). It seems that the D-FINE-S model, with its FDR for improving fine-grained edge placement in blurry images, was better suited to our target dataset.

4.3 Qualitative Analysis

In this section, we will analyse the performance of the models per model family on the following image, which is a representative image with conditions typical of an average in-match image (skewed angles, occluded objects, long-range detections, and complex lighting at distance):

Figure 4.4: Representative image for analysis.

4.3.1 Legacy Models: Faster R-CNN and SSD

For the Faster R-CNN and SSD model, the images in Appendix Section B.4 show overlapping duplicate bounding boxes for the R-CNN model, indicating issues with the NMS not removing duplicate boxes. For SSD, a clear difficulty with line intersections is evident: it fails to detect all but one. These images reflect the results shown in the table, which struggle to keep up with modern architectures in this analysis. Due to these results, we will not analyse these models in depth on the image above in this Chapter.

4.3.2 YOLO models

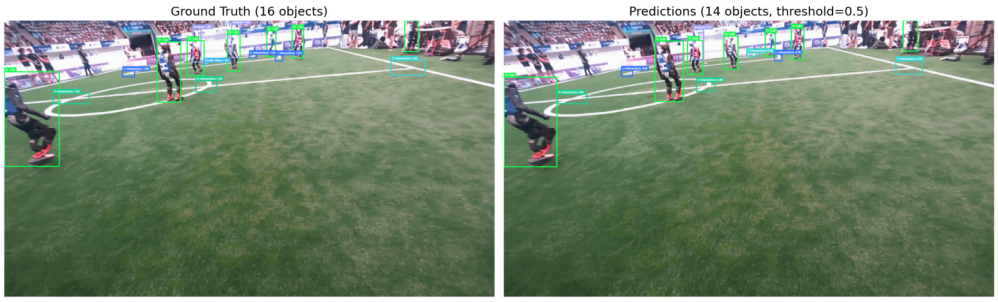

Figure 4.5: Annotated images of the ground truth (left) and YOLOv8-S predictions (right).

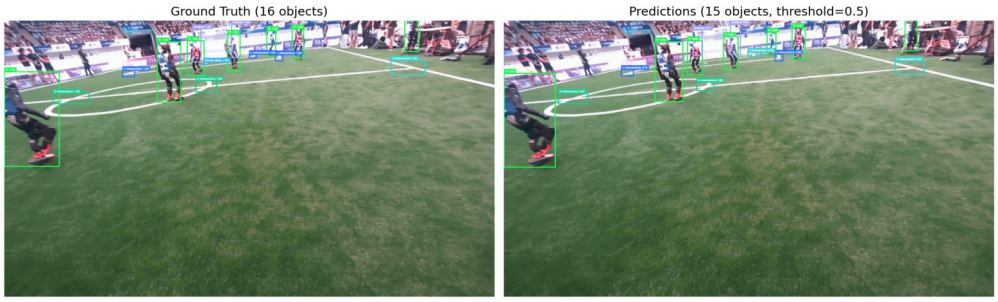

Figure 4.6: Annotated images of the ground truth (left) and YOLO11-S predictions (right).

The images tell a story consistent with the advancements from YOLOv8 to YOLO11. While both models failed to detect the penalty mark, which is a small, long-range target, the YOLO11 model outperforms YOLOv8 in detecting the second, more occluded right goalpost. This image demonstrates YOLO's long-standing reputation as the industry standard for reliable, fast object detection.

4.3.3 Transformer Models

For the transformer models, the differences, strengths and weaknesses of both the D-FINE-S and RF-DETR-S model become clearly visible when looking at the images:

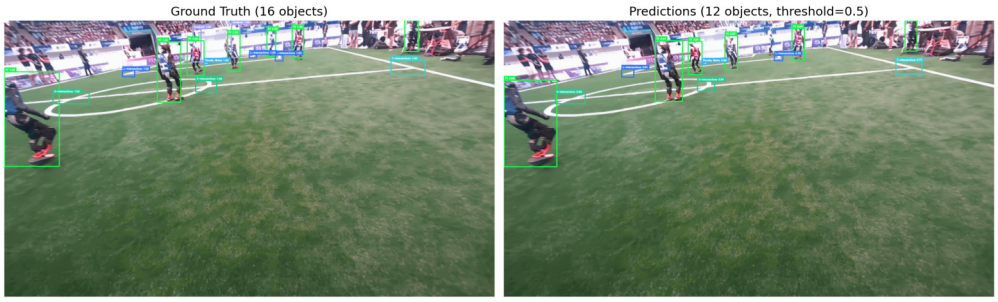

Figure 4.7: Annotated images of the ground truth (left) and RF-DETR-S predictions (right).

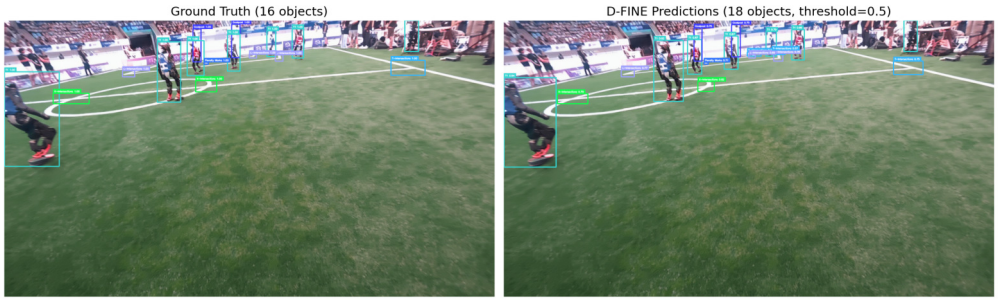

Figure 4.8: Annotated images of the ground truth (left) and D-FINE-S predictions (right).

Due to limitations in the annotation tool, these images can get quite cluttered. In the images, it is evident that RF-DETR detects a respectable number of objects, only struggling on both goalposts, the long-range detection of a T1 robot, and the smaller L-intersection of the inner penalty box. In contrast, at first sight, D-FINE seems to have produced some false positives, with 20 predictions on a ground truth of 16 objects. When we analyse the image, it is evident that D-FINE successfully detected all objects, including those missed by RF-DETR. Additionally, D-FINE detects the two T-intersections behind the L-intersections, which were left unannotated by the annotation team due to the steep angle, making detection near-impossible. The small object detection on the D-FINE model is robust, exceeding even the annotation team's implicit expectations and successfully detecting objects under difficult circumstances.

Chapter 5

Conclusion & Discussion

The research question in this thesis is: *Which of the six object detection models in this analysis strikes the best balance between accuracy and speed on the Jetson Orin platform for Booster K1 usage?* To answer this question, the six models were trained to convergence on our target dataset and classes and benchmarked on their best-performing epochs, based on accuracy and inference speed. The results showed that on the Jetson Orin platform, transformer-based models (RF-DETR and D-FINE) offered the best speed-accuracy trade-off, matching or outperforming the industry standard for inference set by YOLOv8 and YOLO11, while achieving up to 4.5%+ improvements on mAP@[50:95] accuracy, and 5.1%+ improvements on mAP@50.

The Faster R-CNN and SSD models were deemed too dated, with insufficient performance in inference speed and accuracy compared with modern standards. With an inference latency of 135.3ms, the Faster R-CNN model failed to achieve the speed targets required for real-time detection, despite reaching a respectable mAP@.50 of 0.818. For SSD, the lowest inference time was achieved in this analysis at 19.4ms, but it showed limitations in accuracy, achieving only 0.607 mAP@.50.
The YOLO model results showed only marginal improvements from YOLOv8 to YOLO11, indicating that YOLO11 has difficulty leveraging the backbone improvements over YOLOv8.

For the transformer models, the D-FINE model proved more robust than the RF-DETR model for our target dataset with blurry conditions and long-range detections.
To conclude, based on our test results, **D-FINE-S offers the optimal speed-accuracy trade-off for the Jetson Orin platform for K1 usage**.

5.1 Limitations

In this thesis, there were certain limitations that could be improved upon to form even more robust conclusions for this domain of object detection for RoboCup Humanoid League football:

- **Inference on Jetson Orin AGX**: This study had its main constraint in the form of the absence of the actual K1 hardware. Once these become available, tests should be conducted on the final hardware to assess the models' actual performance and real-life performance by running live inference on the robot.
- **Structuring annotation process:** Since the annotation was conducted by a team, this meant a bigger dataset could be used for this analysis. The downside of this approach lies in the subjectivity of when to annotate an object or line (due to blur or distance) and in the size of bounding boxes, which affect the accuracy benchmarks (influencing the IoU between a ground truth and a prediction). A model could have predicted a box tightly according to the rest of the dataset, but it will get punished in benchmarking if the annotator drew a too large bounding box).

5.2 Future Work

In future research, the following steps are recommended:

- **Model Architecture Optimization**: The model architectures of both D-FINE and RF-DETR, the superior models in this analysis, are well documented and highly customizable. Research should be conducted on opportunities to integrate performance-boosting aspects of models into one another's architectures, or to remove layers, to study the impact on performance and inference speed.
- **INT8 Quantization:** For performance gains, it could be interesting to look into the impact on speed and accuracy with INT8 quantization. If the trade-off is worth it, this would also significantly free up memory bandwidth on the robot for other processes.

Chapter 6

Acknowledgements

I would like to express my sincere gratitude to my supervisor, Dr. Arnoud Visser, for his wisdom and guidance throughout this thesis. I would also like to thank the members of the whIRLwind Amsterdam team (formerly known as the Dutch Nao Team), who I consider my dear friends, for their round-the-clock efforts in annotating the dataset and for their consistent support throughout this period. This thesis wouldn't have been possible without these people.

I would like to thank Gijs de Jong for his technical support, guidance, and willingness to support this thesis from beginning to end. His help has been instrumental and is deeply appreciated.

Por fim, gostaria de agradecer aos meus pais, Dulce e Gilmar, ao meu irmão, Pedro, e à minha namorada, Teresa, pelo vosso apoio constante ao longo deste período da tese. O vosso carinho e amor foram palpáveis e fundamentais para a conclusão desta tese. Muito obrigado.

Appendix A

Attempted Models

A.1 Cascade R-CNN

Where Faster R-CNN unified the proposal and detection stages, it performance is still defined by the quality of the Intersection over Union (IoU) threshold used during training. This fixed threshold can lead to many false-positive samples when set too low, while a high threshold can in turn lead to false negatives, causing the model to overfit. The Cascade R-CNN architecture (Cai & Vasconcelos, 2018) addresses this issue by introducing a multi-stage classification head, that employs detectors trained with increasing IoU thresholds. The outputs of the consequent stages serve as the inputs for the following, which leads to progressively refined bounding boxes. This architecture offers yet another improvement in the two-stage detection landscape, further deepening the trade-off between accuracy and speed. Due to the absence of any published code, this model would need to be built on an adaptation of Faster R-CNN. Time constraints put the feasibility of this model outside the scope of this thesis.

A.2 DEYO

The final architecture selected for the two-stage family is DEYO (DETR with YOLO) (Ouyang, 2023). While Faster R-CNN has latency as its primary constraint, this approach proposes a two-stage hybrid architecture of two modern architectures to address this issue. The DEYO model combines the YOLO model with the DETR model as follows:

The first stage utilizes a backbone based on the YOLO architecture (specifically YOLOv8), which serves effectively as a high efficiency RPN that achieves high quality object proposals to serve as an input for the DETR model. This significantly speeds up the training process for the DETR model.

The second stage takes the object proposals from the YOLO model and refines these with the DETR Transformer model. This approach also eliminates the need for Non-Maximum Suppression (NMS), the post-processing step required by two-stage and single-stage models to remove duplicate bounding boxes. DEYO offers a modern, alternative approach to the two-stage formula within this analysis. Time constraints also put this model outside of the scope of this thesis, offering an interesting option for future work.

Appendix B

Experiment Results

B.1 YOLOv8 Inference Study

Table B.1: Detailed Latency Analysis of YOLOv8 Variants over 1000 runs

Model	**Min** (ms)	**Max** (ms)	**Mean** (ms)	**Median** (ms)	**P95** (ms)	**P99** (ms)
YOLOv8n	6.50	10.37	**9.21**	9.66	10.02	10.08
YOLOv8s	11.14	18.33	**17.24**	18.19	18.27	18.31
YOLOv8m	27.44	44.85	**39.38**	43.27	44.21	44.85

B.2 Training Plots

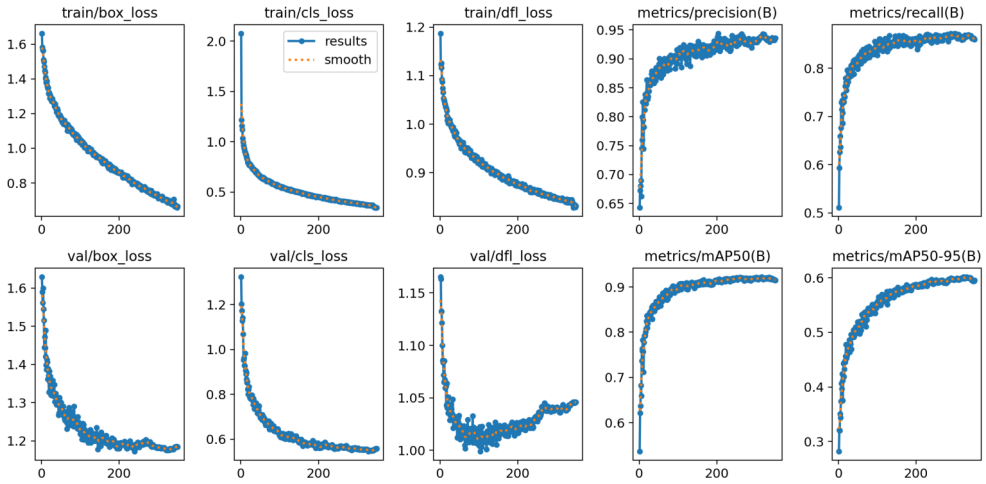

Figure B.1: YOLO11 training metrics.

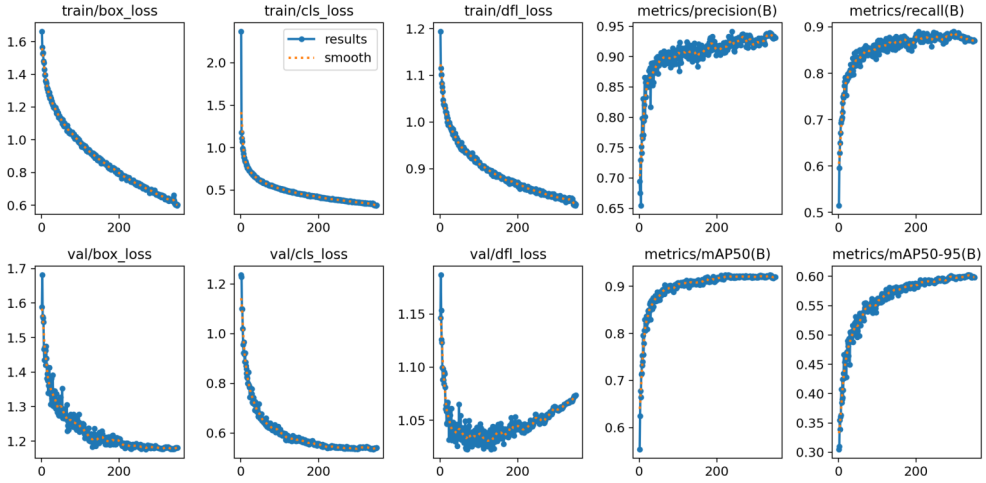

Figure B.2: YOLOv8 training metrics.

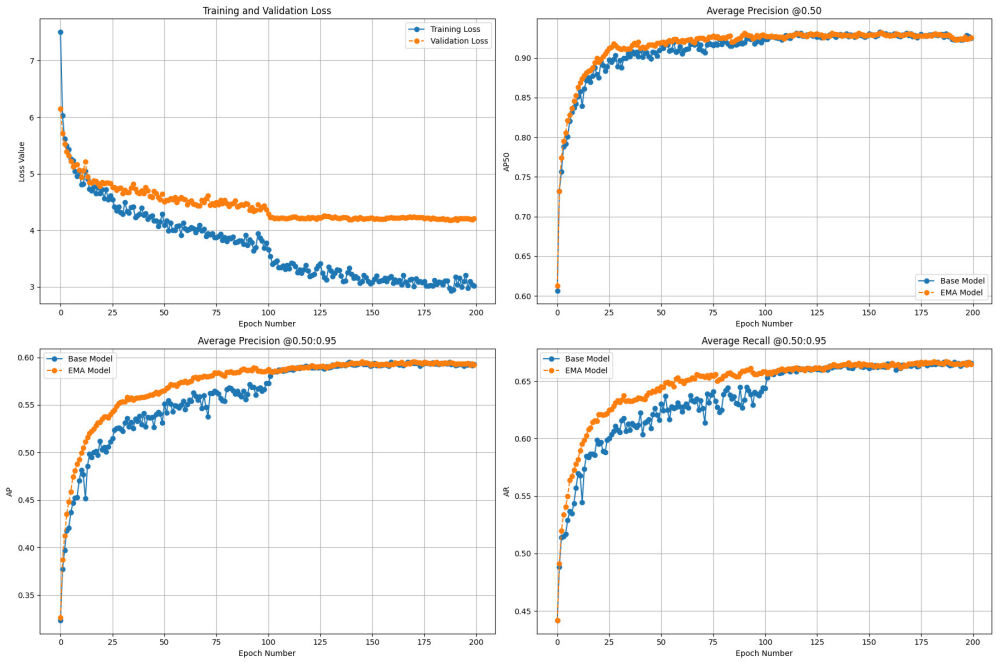

Figure B.3: RF-DETR-S training metrics.

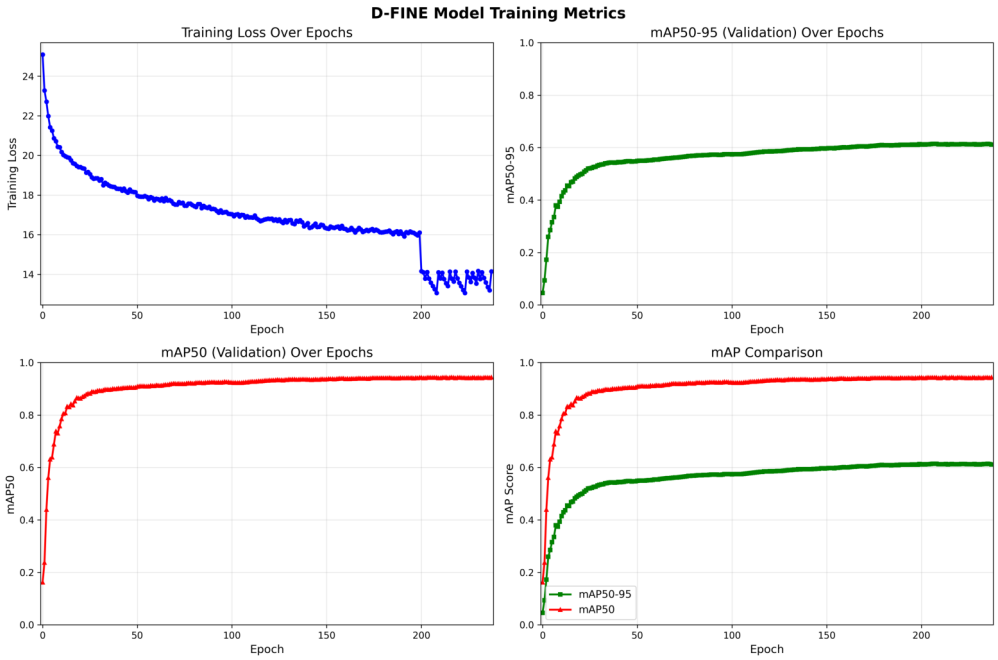

Figure B.4: D-FINE-S training metrics.

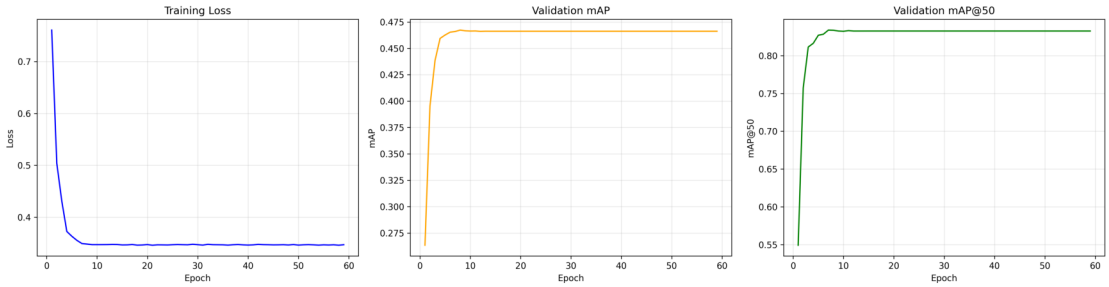

Figure 1: Training loss, validation loss, validation mAP, and validation mAP@50 over 59 epochs

Figure B.5: Faster R-CNN training metrics.

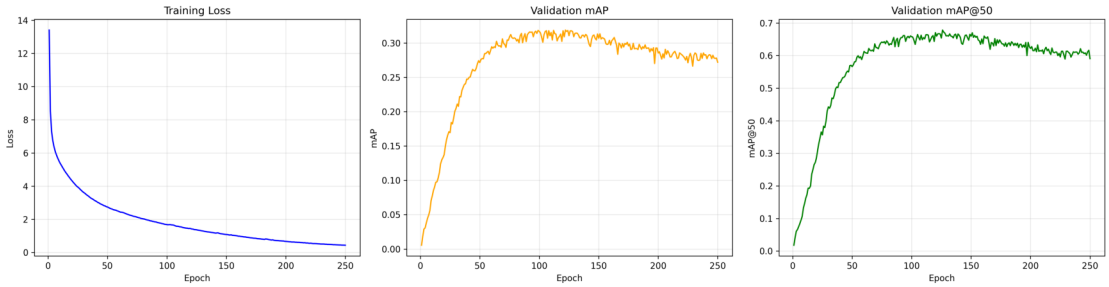

Figure 1: Training loss, validation loss, validation mAP, and validation mAP@50 over 250 epochs

Figure B.6: SSD Multibox Detector training metrics.

B.3 Model Accuracy Score Tables

Table B.2: Average Precision (AP) metrics on the test set.

Model	mAP@[.50:.95]	mAP@.50	mAP@.75	mAP_{Small}	AP_{Medium}	AP_{Large}
D-FINE-S	0.595	0.932	0.617	0.540	0.626	0.708
RF-DETR-S	0.582	0.918	0.602	0.502	0.626	0.704
YOLO11-S	0.552	0.881	0.547	0.461	0.610	0.700
YOLOv8-S	0.548	0.880	0.552	0.460	0.600	0.681
Faster R-CNN	0.449	0.818	0.410	0.361	0.496	0.609
SSD300-VGG16	0.281	0.607	0.224	0.155	0.377	0.501

Table B.3: Average Recall (AR) metrics on the test set.

Model	AR@1	AR@10	AR@100	AR_S	AR_M	AR_L
D-FINE-S	0.452	0.663	0.697	0.646	0.725	0.808
RF-DETR-S	0.453	0.649	0.654	0.587	0.695	0.758
YOLO11-S	0.430	0.625	0.626	0.555	0.674	0.756
YOLOv8-S	0.431	0.622	0.623	0.556	0.664	0.744
Faster R-CNN	0.364	0.545	0.546	0.479	0.589	0.675
SSD300-VGG16	0.261	0.379	0.384	0.260	0.482	0.583

Table B.4: Native YOLO validation metrics (Ultralytics evaluation).

Model	mAP@.50:.95	mAP@.50	mAP@.75
YOLO11-S	0.588	0.905	0.600
YOLOv8-S	0.587	0.905	0.603

B.4 Model Outputs & Ground Truths

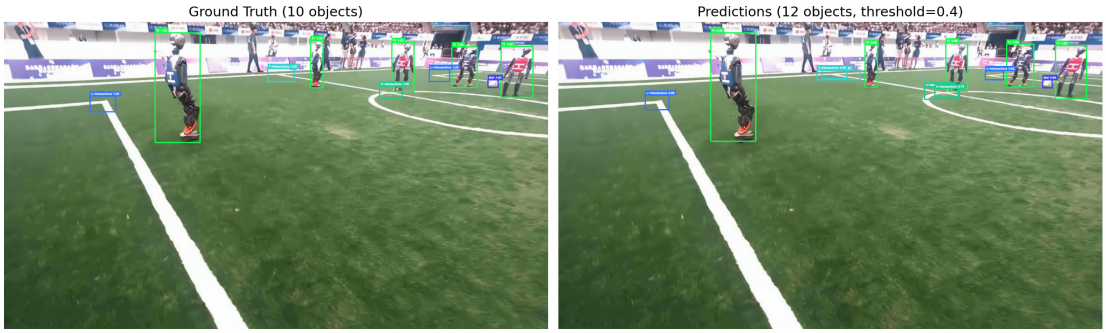

Figure B.7: Model outputs Faster-RCNN

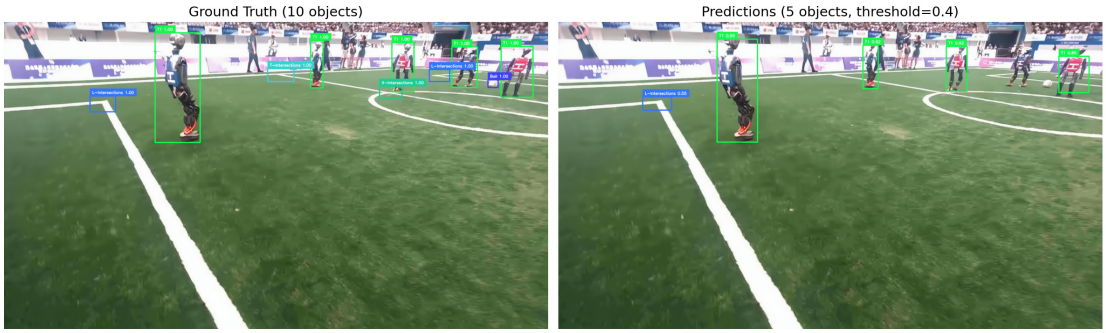

Figure B.8: Model outputs SSD Multibox Detector

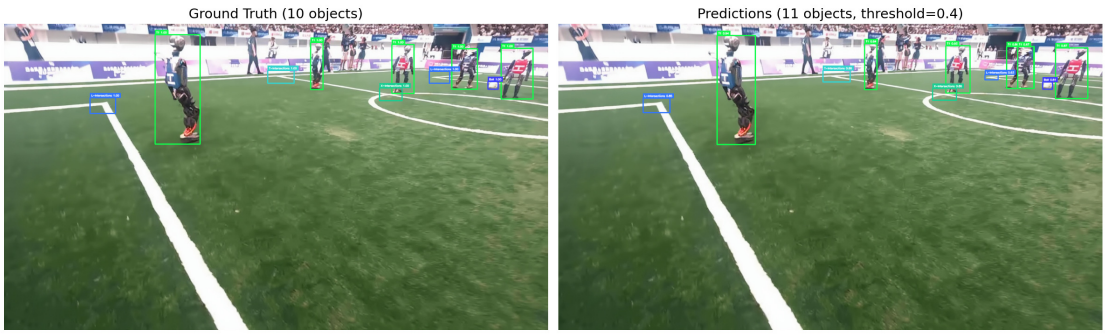

Figure B.9: Model outputs YOLOv8-S

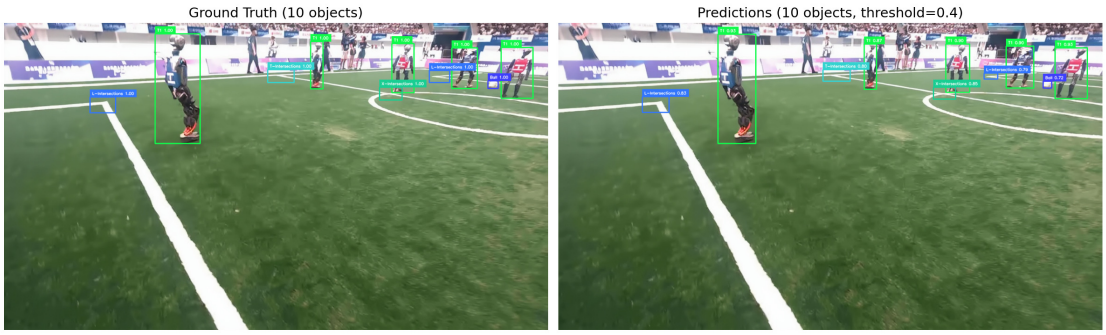

Figure B.10: Model outputs YOLO11-S

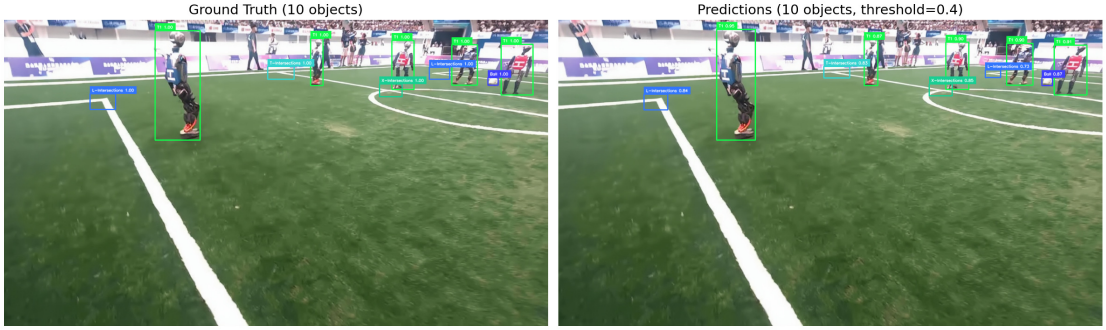

Figure B.11: Model outputs RF-DETR-S

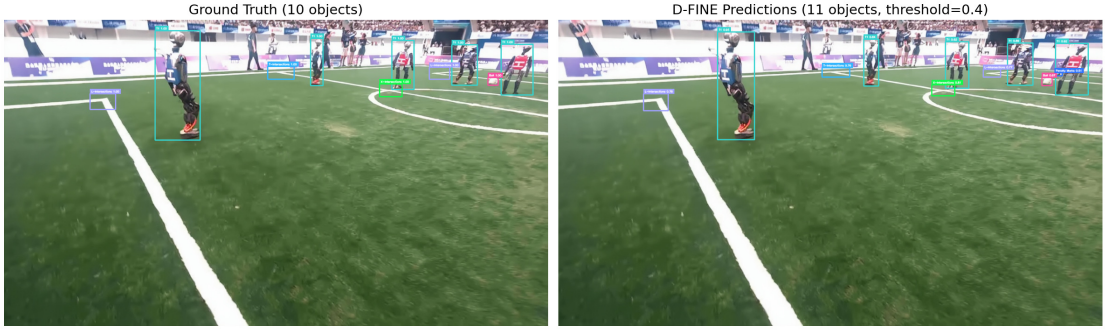

Figure B.12: Model outputs D-FINE-S

Appendix C

Figures

```
.
 dataset/
     annotations/ (COCO JSONs)
        test.json
        train.json
        valid.json
     test/
        images/
           img1.jpg
           ...
        labels/ (YOLO TXT)
           img1.txt
           ...
        _annotations.coco.json (RF-DETR's annotation JSONs)
     train/
        images/
        labels/
        _annotations.coco.json
     valid/
        images/
        labels/
        _annotations.coco.json
     data.yaml (YOLO yaml)
```

Figure C.1: Unified Dataset Directory Structure.

Bibliography

Bestmann, M., Engelke, T., Fiedler, N., Güldenstein, J., Gutsche, J., Hagge, J., & Vahl, F. (2021). TORSO-21 Dataset: Typical Objects in RoboCup Soccer 2021. *RoboCup 2021: Robot World Cup XXIV*.

Cai, Z., & Vasconcelos, N. (2018). Cascade R-CNN: Delving Into High Quality Object Detection. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 6154–6162. https://doi.org/10.1109/CVPR.2018.00644

Carion, N., Massa, F., Synnaeve, G., Usunier, N., Kirillov, A., & Zagoruyko, S. (2020). End-to-End Object Detection with Transformers. In A. Vedaldi, H. Bischof, T. Brox, & J.-M. Frahm (Eds.), *Computer Vision – ECCV 2020* (pp. 213–229, Vol. 12346). Springer International Publishing. https://doi.org/10.1007/978-3-030-58452-8_13

Chen, Q., Su, X., Zhang, X., Wang, J., Chen, J., Shen, Y., Han, C., Chen, Z., Xu, W., Li, F., Zhang, S., Yao, K., Ding, E., Zhang, G., & Wang, J. (2024, June 5). *LW-DETR: A Transformer Replacement to YOLO for Real-Time Detection*. arXiv: 2406.03459 [cs]. https://doi.org/10.48550/arXiv.2406.03459

COCO - Common Objects in Context. (n.d.). Retrieved January 29, 2026, from https://cocodataset.org/#home

de Jong, G., Ruiter, H., Prinzhorn, D. W., Kaiser, J., Honkoop, M., Weerheim, J., Gunnewiek, F. K., Geurts, R., Catarrinho, D. X., Visser, S., Werkhoven, D., van der Veen, R., & Bernardy, M. (2023, December 31). *Dutch nao team - technical report* (tech. rep.).

Faster R-CNN — Torchvision main documentation. (n.d.). Retrieved January 18, 2026, from https://docs.pytorch.org/vision/master/models/faster_rcnn.html

Girshick, R. (2015). Fast R-CNN. *2015 IEEE International Conference on Computer Vision (ICCV)*, 1440–1448. https://doi.org/10.1109/ICCV.2015.169

Girshick, R., Donahue, J., Darrell, T., & Malik, J. (2014). Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. *2014 IEEE Conference on Computer Vision and Pattern Recognition*, 580–587. https://doi.org/10.1109/CVPR.2014.81

Göttsch, F.-S. (2025). Exploring Lightweight Data-Driven Methods for Image Segmentation in the RoboCup SPL.

Jocher, G., Chaurasia, A., & Qiu, J. (2023). *Ultralytics YOLOv8* (Version 8.0.0). https://github.com/ultralytics/ultralytics

Jocher, G., & Qiu, J. (2024). *Ultralytics YOLO11* (Version 11.0.0). https://github.com/ultralytics/ultralytics

Khanam, R., & Hussain, M. (2024, October 23). *YOLOv11: An Overview of the Key Architectural Enhancements* (1). arXiv: 2410.17725 [cs]. https://doi.org/10.48550/arXiv.2410.17725

Lin, T.-Y., Goyal, P., Girshick, R., He, K., & Dollar, P. (2020). Focal Loss for Dense Object Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *42*(2), 318–327. https://doi.org/10.1109/TPAMI.2018.2858826

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., & Berg, A. C. (2016). SSD: Single Shot MultiBox Detector. https://doi.org/10.1007/978-3-319-46448-0_2

Monté, X., van der Kaaij, J., van der Weerd, R., & Visser, A. (2023). Using neural factorization of shape and reflectance for ball detection.

ONNX | Home. (n.d.). Retrieved January 30, 2026, from https://onnx.ai/

Oquab, M., Darcet, T., Moutakanni, T., Vo, H. V., Szafraniec, M., Khalidov, V., Fernandez, P., Haziza, D., Massa, F., El-Nouby, A., Assran, M., Ballas, N., Galuba, W., Howes, R., Huang, P.-Y., Li, S.-W., Misra, I., Rabbat, M., Sharma, V., … Bojanowski, P. (2024). DINOv2: Learning Robust Visual Features without Supervision. *Trans. Mach. Learn. Res.*, *2024*. Retrieved December 3, 2025, from https://openreview.net/forum?id=a68SUt6zFt

Ouyang, H. (2023, June 16). *DEYO: DETR with YOLO for Step-by-Step Object Detection*. arXiv: 2211.06588 [cs]. https://doi.org/10.48550/arXiv.2211.06588

Pal, S., Pramanik, A., Maiti, J., & Mitra, P. (2021). Deep learning in multi-object detection and tracking: State of the art. *Applied Intelligence*, *51*. https://doi.org/10.1007/s10489-021-02293-7

Peng, Y., Li, H., Wu, P., Zhang, Y., Sun, X., & Wu, F. (2024, October 17). *D-FINE: Redefine Regression Task in DETRs as Fine-grained Distribution Refinement*. arXiv: 2410.13842 [cs]. https://doi.org/10.48550/arXiv.2410.13842

Poppinga, B., & Laue, T. (2019). JET-Net: Real-Time Object Detection for Mobile Robots. In S. Chalup, T. Niemueller, J. Suthakorn, & M.-A. Williams (Eds.), *RoboCup 2019: Robot World Cup XXIII* (pp. 227–240, Vol. 11531). Springer International Publishing. https://doi.org/10.1007/978-3-030-35699-6_18

Redmon, J., Divvala, S. K., Girshick, R. B., & Farhadi, A. (2016). You Only Look Once: Unified, Real-Time Object Detection. *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, 779–788. https://doi.org/10.1109/CVPR.2016.91

Ren, S., He, K., Girshick, R., & Sun, J. (2017). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *39*(6), 1137–1149. https://doi.org/10.1109/TPAMI.2016.2577031

Robinson, I., Robicheaux, P., Popov, M., Ramanan, D., & Peri, N. (2025, November 12). *RF-DETR: Neural Architecture Search for Real-Time Detection Transformers*. arXiv: 2511.09554 [cs]. https://doi.org/10.48550/arXiv.2511.09554

Ruiter, H., de Jong, G., Meijer, M., González, M. O., Honkoop, M., Blaauboer, J., van der Veen, R., Gunnewiek, F. K., de Haan, M., Nagelhout, F., Weerheim, J., Visser, S., Sprott, J., & Yao, J. (2024, December 30). *Dutch nao team - technical report* (tech. rep.).

Sohan, M., Sai Ram, T., & Rami Reddy, C. V. (2024). A Review on YOLOv8 and Its Advancements. In I. J. Jacob, S. Piramuthu, & P. Falkowski-Gilski (Eds.), *Data Intelligence and Cognitive Informatics* (pp. 529–545). Springer Nature Singapore. https://doi.org/10.1007/978-981-99-7962-2_39

SSD — Torchvision main documentation. (n.d.). Retrieved January 27, 2026, from https://docs.pytorch.org/vision/main/models/ssd.html

Ultralytics. (n.d.). *Explore Ultralytics YOLOv8*. Retrieved January 29, 2026, from https://docs.ultralytics.com/models/yolov8/

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2023, August 2). *Attention Is All You Need*. arXiv: 1706.03762 [cs]. https://doi.org/10.48550/arXiv.1706.03762

Yaseen, M. (2024, August 28). *What is YOLOv8: An In-Depth Exploration of the Internal Features of the Next-Generation Object Detector* (1). arXiv: 2408.15857 [cs]. https://doi.org/10.48550/arXiv.2408.15857

Zhang, X., Chen, J., Yuan, J., Chen, Q., Wang, J., Wang, X., Han, S., Chen, X., Pi, J., Yao, K., Han, J., Ding, E., & Wang, J. (2022, November 17). *CAE v2: Context Autoencoder with CLIP Target*. arXiv: 2211.09799 [cs]. https://doi.org/10.48550/arXiv.2211.09799

Zhao, Y., Lv, W., Xu, S., Wei, J., Wang, G., Dang, Q., Liu, Y., & Chen, J. (2024). DETRs Beat YOLOs on Real-time Object Detection. *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 16965–16974. https://doi.org/10.1109/CVPR52733.2024.01605

Zhu, X., Su, W., Lu, L., Li, B., Wang, X., & Dai, J. (2021, March 18). *Deformable DETR: Deformable Transformers for End-to-End Object Detection*. arXiv: 2010.04159 [cs]. https://doi.org/10.48550/arXiv.2010.04159

Zoph, B., & Le, Q. V. (2017, February 15). *Neural Architecture Search with Reinforcement Learning*. arXiv: 1611.01578 [cs]. https://doi.org/10.48550/arXiv.1611.01578

Zou, Z., Chen, K., Shi, Z., Guo, Y., & Ye, J. (2023, January 18). *Object Detection in 20 Years: A Survey*. arXiv: 1905.05055 [cs]. https://doi.org/10.48550/arXiv.1905.05055