DETRs Beat YOLOs on Real-time Object Detection

Abstract

The YOLO series has become the most popular frame work for real-time object detection due to its reasonable trade-off between speed and accuracy. However, we observe that the speed and accuracy of YOLOs are negatively af fected by the NMS. Recently, end-to-end Transformer-based detectors (DETRs) have provided an alternative to eliminat MS COCO Object Detection R101 X 54 R50 R50 L L X X 52 50 L Scaled M M L X M ing NMS. Nevertheless, the high computational cost limits L their practicality and hinders them from fully exploiting the advantage of excluding NMS. In this paper, we propose the Real-Time DEtection TRansformer (RT-DETR), the first real-time end-to-end object detector to our best knowledge that addresses the above dilemma. We build RT-DETR in two steps, drawing on the advanced DETR: first we focus on maintaining accuracy while improving speed, followed by maintaining speed while improving accuracy. Specifi cally, we design an efficient hybrid encoder to expeditiously process multi-scale features by decoupling intra-scale inter action and cross-scale fusion to improve speed. Then, we propose the uncertainty-minimal query selection to provide high-quality initial queries to the decoder, thereby improv ing accuracy. In addition, RT-DETR supports flexible speed tuning by adjusting the number of decoder layers to adapt to various scenarios without retraining. Our RT-DETR-R50 / R101 achieves 53.1% / 54.3% AP on COCO and 108 / 74 FPSonT4GPU,outperforming previously advanced YOLOs in both speed and accuracy. Furthermore, RT-DETR-R50 outperforms DINO-R50 by 2.2% AP in accuracy and about 21 times in FPS. After pre-training with Objects365, RT DETR-R50 / R101 achieves 55.3% / 56.2% AP. The project page: https://zhao-yian.github.io/RTDETR.

Keywords: Real-time Object Detection, YOLO,Transformer-based Detectors,Real-Time DEtection TRansformer.

1 Introduction

Real-time object detection is an important area of research and has a wide range of applications, such as object tracking [1], video surveillance [2], and autonomous driving [3], etc. Existing real-time detectors generally adopt the CNN-based architecture, the most famous of which is the YOLO detectors [4–11] due to their reasonable trade-off between speed and accuracy. However, these detectors typically require Non-Maximum Suppression (NMS) for post-processing, which not only slows down the inference speed but also introduces hyperparameters that cause instability in both the speed and accuracy.Moreover, considering that different scenarios place different emphasis on recall and accuracy, it is necessary to carefully select the appropriate NMS thresholds, which hinders the development of real-time detectors. In this paper, we propose the Real-Time DEtection TRansformer (RT-DETR), the first real-time end-to-end object detector to our best knowledge. To

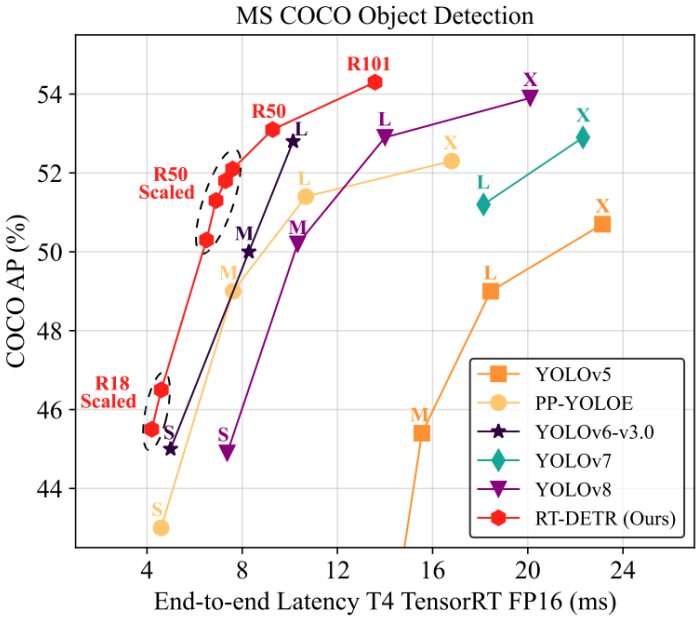

Figure 1. Compared to previously advanced real-time object detectors, our RT-DETR achieves state-of-the-art performance.

expeditiously process multi-scale features, we design an efficient hybrid encoder to replace the vanilla Transformer encoder, which significantly improves inference speed by decoupling the intra-scale interaction and cross-scale fusion of features with different scales. To avoid encoder features with low localization confidence being selected as object queries, we propose the uncertainty-minimal query selection, which provides high-quality initial queries to the decoder by explicitly optimizing the uncertainty, thereby increasing the accuracy. Furthermore, RT-DETR supports flexible speed tuning to accommodate various real-time scenarios without retraining, thanks to the multi-layer decoder architecture of DETR.

RT-DETR achieves an ideal trade-off between the speed and accuracy. Specifically, RT-DETR-R50 achieves 53.1% AP on COCO val2017 and 108 FPS on T4 GPU, while RT-DETR-R101 achieves 54.3% AP and 74 FPS, outperforming L and X models of previously advanced YOLO detectors in both speed and accuracy, Figure 1. We also develop scaled RT-DETRs by scaling the encoder and decoder with smaller backbones, which outperform the lighter YOLO detectors (S and M models). Furthermore, RT-DETR-R50 outperforms DINO-Deformable-DETR-R50 by 2.2% AP (53.1% AP vs 50.9% AP) in accuracy and by about 21 times in FPS (108 FPS vs 5 FPS), significantly improves accuracy and speed of DETRs. After pre-training with Objects365 [12], RT-DETR-R50 / R101 achieves 55.3% / 56.2% AP, resulting in surprising performance improvements. More experimental results are provided in the Appendix.

The main contributions are summarized as: (i). We propose the first real-time end-to-end object detector called RT-DETR, which not only outperforms the previously advanced YOLO detectors in both speed and accuracy but also eliminates the negative impact caused by NMS post-processing on real-time object detection; (ii). We quantitatively analyze the impact of NMS on the speed and accuracy of YOLO detectors, and establish an end-to-end speed benchmark to test the end-to-end inference speed of real-time detectors; (iii). The proposed RT-DETR supports flexible speed tuning by adjusting the number of decoder layers to accommodate various scenarios without retraining.

2 Related works

2.1 Real-timeObjectDetectors

YOLOv1 [13] is the first CNN-based one-stage object detector to achieve true real-time object detection. Through years of continuous development, the YOLO detectors have outperformed other one-stage object detectors [14, 15] and become the synonymous with the real-time object detector. YOLO detectors can be classified into two categories: anchor-based [4, 5, 7, 9, 10, 16, 17] and anchor-free [6, 8, 11], which achieve a reasonable trade-off between speed and accuracy and are widely used in various practical scenarios. These advanced real-time detectors produce numerous overlapping boxes and require NMS post-processing, which slows down their speed.

2.2 End-to-endObjectDetectors

DETR, introduced by Carion et al., is an end-to-end object detector based on Transformers, eliminating anchors and NMS components. However, it faces issues like slow convergence, high computational cost, and hard-to-optimize queries. Various variants address these challenges: Deformable-DETR accelerates convergence with multi-scale features, while DAB-DETR and DN-DETR use iterative refinement and denoising. EfficientDETR, SparseDETR, and LiteDETR reduce computational cost by modifying the number of layers or updating features more efficiently.

3 Method

3.1 Overview

RT-DETR consists of a backbone, an efficient hybrid encoder, and a Transformer decoder with auxiliary prediction heads. The overview of RT-DETR is illustrated in Figure 2. Specifically, we feed the features from the last three stages of the backbone $\{S_3, S_4, S_5\}$ into the encoder. The efficient hybrid encoder transforms multi-scale features into a sequence of image features through intra-scale feature interaction and cross-scale feature fusion (cf. Sec. 4.2). Subsequently, the uncertainty-minimal query selection is employed to select a fixed number of encoder features to serve as initial object queries for the decoder (cf. Sec. 4.3). Finally, the decoder with auxiliary prediction heads iteratively optimizes object queries to generate categories and boxes.

3.2 Effcient Hybrid Encoder

The analysis focuses on addressing the computational bottleneck in Deformable-DETR. The introduction of multi-scale features improves performance and accelerates training convergence, but the increased sequence length makes the encoder the primary computational bottleneck. Despite deformable attention reducing computational cost, the encoder still consumes a significant portion of GFLOPs (49%), while contributing only 11% to the Average Precision (AP).Therefore, we design a set of variants with different types of the encoder to prove that the simultaneous intra-scale and cross-scale feature interaction is inefficient, as shown in Figure 3. To overcome this issue, the authors analyze the redundancy in the multi-scale Transformer encoder. They propose that high-level features, which contain rich semantic information, should not undergo feature interaction with

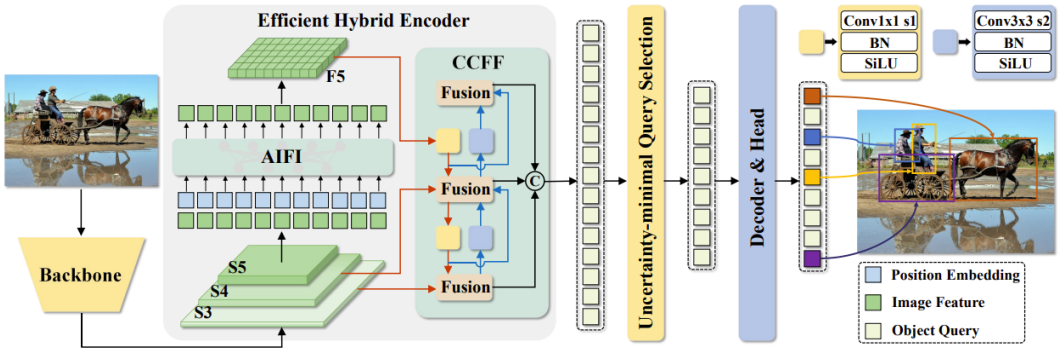

Figure 2. Overview of RT-DETR. We feed the features from the last three stages of the backbone into the encoder. The efficient hybrid encoder transforms multi-scale features into a sequence of image features through the Attention-based Intra-scale Feature Interaction (AIFI) and the CNN-based Cross-scale Feature Fusion (CCFF). Then, the uncertainty-minimal query selection selects a fixed number of encoder features to serve as initial object queries for the decoder. Finally, the decoder with auxiliary prediction heads iteratively optimizes object queries to generate categories and boxes.

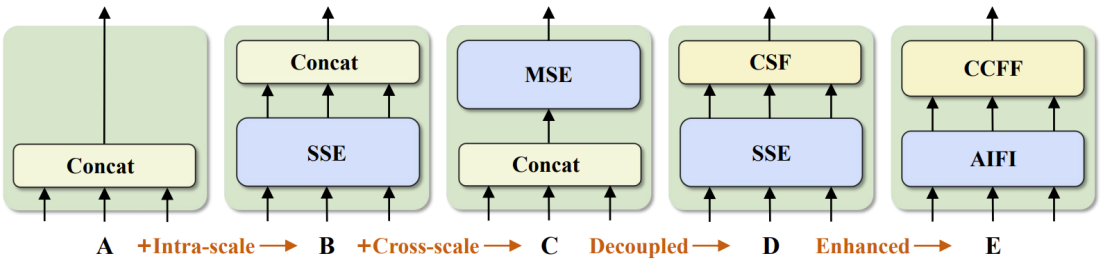

Figure 3. The encoder structure for each variant. SSE represents the single-scale Transformer encoder, MSE represents the multi-scale Transformer encoder, and CSF represents cross-scale fusion. AIFI and CCFF are the two modules designed into our hybrid encoder.

low-level features, as this results in inefficiencies. A set of variants is designed to demonstrate that simultaneous intra-scale and cross-scale feature interaction is inefficient. This is tested using DINO-Deformable-R50 with modifications to the encoder, leading to the creation of a series of variants, including variant A, where the multi-scale Transformer encoder is removed.

Building on this, the authors propose an efficient hybrid encoder consisting of two components: Attention-based Intra-scale Feature Interaction (AIFI) and CNN-based Cross-scale Feature Fusion (CCFF). AIFI reduces computational cost by performing intra-scale interaction only on the high-level features (S5), where self-attention is more beneficial due to the richer semantic concepts. In contrast, lower-level features do not need such interactions. Experimental results show that this approach significantly reduces latency (35% faster) and improves accuracy (0.4% AP higher). CCFF optimizes cross-scale fusion by using convolutional fusion blocks that combine adjacent scale features, adjusting the number of channels and merging outputs via element-wise addition.

The final hybrid encoder is computationally more efficient and achieves better performance, as shown by the experimental results.

3.3 Uncertainty-minimal Query Selection

To reduce the difficulty of optimizing object queries in DETR, several subsequent works [42, 44, 45] propose query selection schemes, which have in common that they use the confidence score to select the top K features from the encoder to initialize object queries (or just position queries). The confidence score represents the likelihood that the feature includes foreground objects. Nevertheless, the detector is required to simultaneously model the category and location of objects, both of which determine the quality of the features. Hence, the performance score of the feature is a latent variable that is jointly correlated with both classification and localization. Based on the analysis, the current query selection leads to a considerable level of uncertainty in the selected features, resulting in sub-optimal initialization for the decoder and hindering the performance of the detector.

To address this problem, we propose the uncertainty-minimal query selection scheme, which explicitly constructs and optimizes the epistemic uncertainty to model the joint latent variable of encoder features, thereby providing high-quality queries for the decoder. Specifically, the feature uncertainty U is defined as the discrepancy between the predicted distributions of localization P and classification C in Eq. (1). To minimize the uncertainty of the queries, we integrate the uncertainty into the loss function for the gradient-based optimization in Eq. (2).where $\hat{\mathcal{Y}}$ and $\mathcal{Y}$ denote the prediction and ground truth, $\mathcal{Y} = \{\hat{\mathbf{c}}, \hat{\mathbf{b}}\}$, $\hat{\mathbf{c}}$ and $\hat{\mathbf{b}}$ represent the category and bounding box respectively, $\hat{\mathcal{X}}$ represent the encoder feature.

$$U(\hat{\mathcal{X}}) = \|P(\hat{\mathcal{X}}) - C(\hat{\mathcal{X}})\|, \quad \hat{\mathcal{X}} \in \mathbb{R}^D \tag{1}$$

$$\mathcal{L}(\hat{\mathcal{X}}, \hat{\mathcal{Y}}, \mathcal{Y}) = \mathcal{L}_{box}(\hat{\mathbf{b}}, \mathbf{b}) + \mathcal{L}_{cls}(U(\hat{\mathcal{X}}), \hat{\mathbf{c}}, \mathbf{c}) \tag{2}$$

Effectiveness analysis. To analyze the effectiveness of the uncertainty-minimal query selection, we visualize the classification scores and IoU scores of the selected features on COCO `val2017`, Figure 4. We draw the scatterplot with classification scores greater than 0.5. The purple and green dots represent the selected features from the model trained with uncertainty-minimal query selection and vanilla query selection, respectively. The closer the dot is to the top right of the figure, the higher the quality of the corresponding feature, i.e., the more

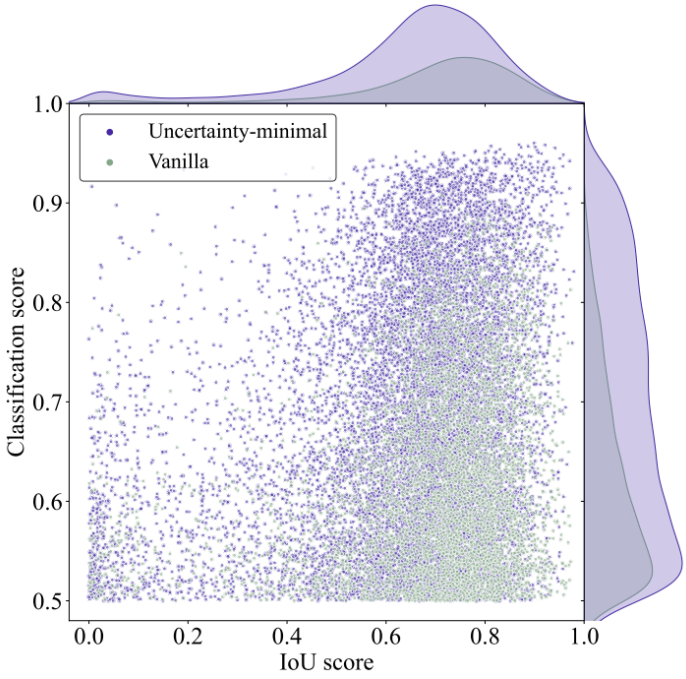

Figure 4. Classification and IoU scores of the selected encoder features. Purple and Green dots represent the selected features from model trained with uncertainty-minimal query selection and vanilla query selection, respectively.

likely the predicted category and box are to describe the true object. The top and right density curves reflect the number of dots for two types.

The most striking feature of the scatterplot is that the purple dots are concentrated in the top right of the figure, while the green dots are concentrated in the bottom right. This shows that uncertainty-minimal query selection produces more high-quality encoder features. Furthermore, we perform quantitative analysis on two query selection schemes. There are 138% more purple dots than green dots, i.e., more green dots with a classification score less than or equal to 0.5, which can be considered low-quality features. And there are 120% more purple dots than green dots with both scores greater than 0.5. The same conclusion can be drawn from the density curves, where the gap between purple and green is most evident in the top right of the figure. Quantitative results further demonstrate that the uncertainty-minimal query selection provides more features with accurate classification and precise location for queries, thereby improving the accuracy of the detector (cf. Sec. 5.3).

3.4 Scaled RT-DETR

Since real-time detectors typically provide models at different scales to accommodate different scenarios, RT-DETR also supports flexible scaling. Specifically, for the hybrid encoder, we control the width by adjusting the embedding dimension and the number of channels, and the depth by adjusting the number of Transformer layers and *RepBlocks*.

The width and depth of the decoder can be controlled by manipulating the number of object queries and decoder layers. Furthermore, the speed of RT-DETR supports flexible adjustment by adjusting the number of decoder layers. We observe that removing a few decoder layers at the end has minimal effect on accuracy,

but greatly enhances inference speed (cf. Sec. 5.4). We compare the RT-DETR equipped with ResNet50 and ResNet101 to the L and X models of YOLO detectors. Lighter RT-DETRs can be designed by applying other smaller (e.g., ResNet18/34) or scalable (e.g., CSPResNet [6]) backbones with scaled encoder and decoder. We compare the scaled RT-DETRs with the lighter (S and M) YOLO detectors in Appendix, which outperform all S and M models in both speed and accuracy.

4 Implementation details

4.1 Comparing with the released source codes

When training the dataset, we employed three different models: YOLOv8, YOLOv11, and RT-DETR. Below is a detailed description of each step:

(1) **YOLO Series Model Training:**

First, we imported the YOLO class and selected the weight files for the YOLOv8 and YOLOv11 models (yolo8n.pt and yolo11n.pt, respectively). We then initiated the training process for the YOLOv11 model, utilizing the dataset configuration file (my_coco.yaml) that you meticulously prepared. In the training configuration, we specified the dataset path (/workspace/ultralytics/datasets_10/my_coco.yaml), set the training epochs to 100, and defined the image size (imgsz) as 640 pixels to ensure all images are resized to this dimension during training. Additionally, we provided the flexibility to choose between GPU and CPU for training.

(2) **RT-DETR Model Training:**

We imported the RTDETR class and loaded a specific configuration file (rtdetr-l-test.yaml). Subsequently, we commenced the training of the RT-DETR model, leveraging pre-trained weights and loading another dataset configuration file (african-wildlife.yaml). The training settings included enabling pre-trained weights (pretrained=True), specifying the dataset path (/workspace/ultralytics/datasetAnimal/african-wildlife.yaml), setting the training epochs to 230, defining the batch size (batch) as 16, and again setting the image size to 640 pixels. To accelerate the data loading process, we employed 8 worker threads (workers=8).

These three models were trained using different architectures. YOLOv8 and YOLOv11 belong to the YOLO series, specifically designed for object detection tasks; whereas RT-DETR is a Transformer-based detection model, adept at handling more complex visual tasks. We selected different models and configurations to compare their performance on the dataset and assess their accuracy, efficiency, and generalization capabilities. Through this comparison, we can better understand the strengths and weaknesses of each model and choose the most suitable model architecture for future research and applications.

4.2 Experimental Environment Setup

Our experiments were conducted on Tencent Cloud servers, which were equipped with the following hardware specifications suitable for deep learning training:

- **GPU Memory:** 16GB or more, accommodating the demands of deep learning model training.
- **Computational Power:** Over 8 TFlops SP, supporting the training of large-scale models.
- **CPU:** 8 cores, facilitating data processing tasks.
- **RAM:** 32GB, enabling multi-threading and handling of large datasets.

The Python packages required for our experiments include:

- **ultralytics:** For YOLO and RT-DETR model implementations.
- **torch:** The PyTorch library for building and training neural networks.
- **torchvision:** (Optional) For image preprocessing tasks.
- **numpy:** For numerical computations.
- **opencv-python:** For image processing capabilities.
- **matplotlib, seaborn:** For data visualization.

This setup ensures that our experiments are conducted in an environment that is capable of handling the computational intensity of deep learning model training and provides the necessary tools for data preprocessing, model implementation, and result visualization.

4.3 Experimental Process

We selected the "African Poultry" and "African Wildlife" datasets for training. These datasets include images of various African poultry or wildlife in different environments, aiming to simulate the complexity and diversity of real-world application scenarios.

(1) **Model Selection**

In this experiment, we chose three advanced object detection models for comparison and evaluation, namely:

(a) **RT-detr:** A real-time object detection model based on Transformer, known for its efficient feature extraction and accurate target localization capabilities.

(b) **YoloV8:** One of the latest versions of the YOLO series, widely recognized for its fast detection speed and high accuracy.

(c) **YoloV11:** Another version of the YOLO series, further optimizing the model structure and detection performance.

(2) **Evaluation Criteria**

To comprehensively evaluate the performance of the models, we adopted the following evaluation metrics:

(a) **map50:** The mean Average Precision (mAP) at an Intersection over Union (IoU) threshold of 0.5, an important indicator of the model's detection accuracy.

(b) **recall:** The proportion of all targets correctly identified by the model.

(c) **FLOP:** The number of floating-point operations, used to assess the computational complexity and efficiency of the model.

(3) **Experimental Procedure**

(a) **Input Original Images:** At the beginning of the experiment, we first input the original images of African poultry into the network model. These images contain the targets we need to detect and identify.

(b) **Network Model Prediction:** After the input images are processed by the network model, the model predicts the targets in the images, including their categories and locations.

(c) **Output Inference Images:** After the model prediction is completed, we obtain the inference images. In these images, the targets identified by the model are marked with bounding boxes and labeled with corresponding category tags (e.g., "chicken").

(4) **Experimental Results**

By comparing the performance of different models on the same dataset, we can evaluate their strengths and weaknesses and select the most suitable model for our research needs. This process not only helps improve the accuracy of object detection but also optimizes the computational efficiency of the model, providing strong support for practical applications.

The research flowchart of this experiment provides us with a clear perspective, demonstrating the entire process from data preparation to model evaluation, laying a solid foundation for our research work.

5 Conclusion and future work

Experimental Analysis

In this experiment, we compared three different object detection models: YOLOv8, YOLOv11, and RT-DETR, to evaluate their performance on two distinct datasets. These datasets are the "Custom Animal Dataset" and the "African Wildlife Dataset". Our evaluation metrics included model parameters (Param), computational complexity (GFLOPs), model size (ModelSize), mean Average Precision (mAP@50), mean Average Precision (mAP@0.5-0.95), and the number of training epochs (epoch).

Custom Animal Dataset,as shown in table 1.

On the Custom Animal Dataset, we observed the following:

- The YOLOv8 model had 3.01M parameters, 8.1 GFLOPs, and a model size of 6.3MB. It achieved an mAP@50 of 0.917 and an mAP@0.5-0.95 of 0.69, with a training period of 100 epochs.
- The YOLOv11 model had slightly fewer parameters than YOLOv8, with 2.58M parameters, 6.3 GFLOPs, and a model size of 5.5MB. It achieved an mAP@50 of 0.925 and an mAP@0.5-0.95 of 0.692, also trained for 100 epochs.

- The RT-DETR model had significantly more parameters than the previous two, with 31.99M parameters, 103.4 GFLOPs, and a model size of 66.2MB. Despite its higher computational complexity and model size, it only achieved an mAP@50 of 0.791 and an mAP@0.5-0.95 of 0.564, with a training period of 230 epochs. This indicates that RT-DETR did not perform as well as the YOLO series models on the Custom Animal Dataset.

African Wildlife Dataset,as shown in table 2.

On the African Wildlife Dataset, we obtained the following results:

- The YOLOv8 model achieved an mAP@50 of 0.95 and an mAP@0.5-0.95 of 0.813, both better than its performance on the Custom Animal Dataset, with a training period of 100 epochs.
- The YOLOv11 model achieved an mAP@50 of 0.958 and an mAP@0.5-0.95 of 0.822, further demonstrating its superiority in object detection tasks, also trained for 100 epochs.
- The RT-DETR model continued to underperform on the African Wildlife Dataset, with an mAP@50 of only 0.16 and an mAP@0.5-0.95 of 0.0876, with a training period of 100 epochs. This suggests that RT-DETR has a significant performance bottleneck when dealing with the African Wildlife Dataset.

Conclusion

Combining the experimental results from both datasets, we can draw the following conclusions:

- The YOLO series models (YOLOv8 and YOLOv11) performed excellently in object detection tasks, with high average precision and recall rates, and relatively low computational complexity and model sizes, making them suitable for deployment in resource-constrained environments.
- Although the RT-DETR model theoretically has the potential to handle complex visual tasks, its performance in this experiment did not meet expectations, especially on the African Wildlife Dataset. This may be due to mismatches in model structure, training strategies, or dataset characteristics.
- Model selection should be based on specific application scenarios and dataset characteristics. For specific datasets like the Custom Animal and African Wildlife, the YOLO series models may be more appropriate.

This experiment provides valuable insights that guide us in choosing the right object detection models for future work and optimizing model structures and training strategies to improve detection performance.Also attach our results diagram, as shown in Figure 5.

Table 1. Performance comparison on the custom dataset

Model	Param (M)	GFLOPs	ModelSize (MB)	map@50	map@0.5-0.95	epoch
YOLOv8	3.01	8.1	6.3	0.917	0.69	100
YOLOv11	2.58	6.3	5.5	0.925	0.692	100
RT-DETR	31.99	103.4	66.2	0.791	0.564	230

Table 2. Performance comparison on the animal dataset

(animal)	Param(M)	GFLOPs	ModelSize(MB)	map@50	map@0.5-0.95	epoch
YOLOv8	3.01	8.1	6.2	0.95	0.813	100
YOLOv11	2.58	6.3	5.5	0.958	0.822	100
RT-DETR	31.99	103.4	66.2	0.16	0.0876	100

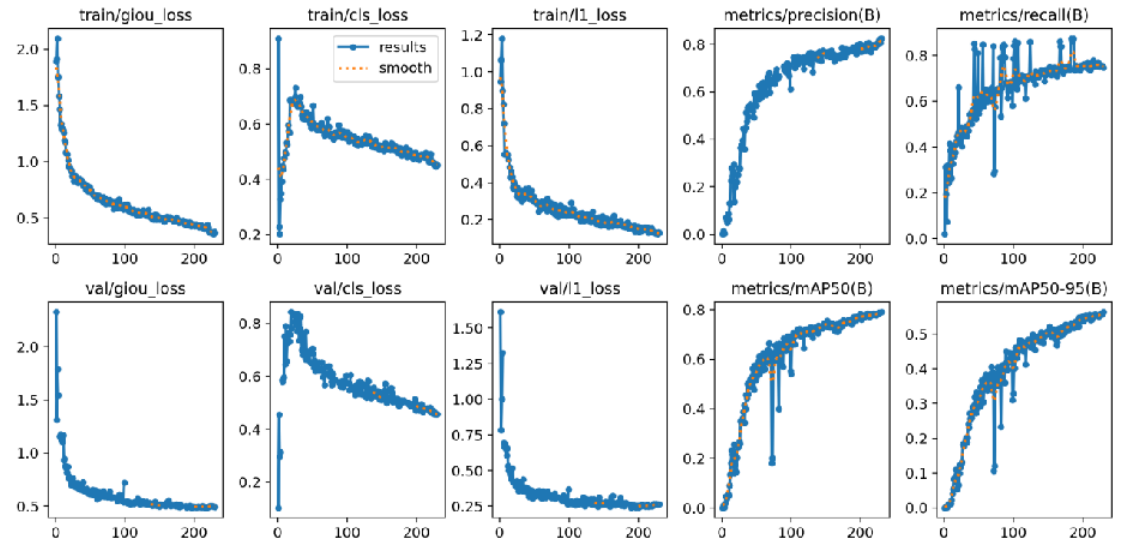

Figure 5. Visualized results

References

[1] Fangao Zeng, Bin Dong, Yuang Zhang, Tiancai Wang, Xiangyu Zhang, and Yichen Wei. Motr: End-to-end multiple-object tracking with transformer. In *European Conference on Computer Vision*, pages 659–675. Springer, 2022.

[2] Rashmika Nawaratne, Damminda Alahakoon, Daswin De Silva, and Xinghuo Yu. Spatiotemporal anomaly detection using deep learning for real-time video surveillance. *IEEE Transactions on Industrial Informatics*, 16(1):393–402, 2019.

[3] Daniel Bogdoll, Maximilian Nitsche, and J Marius Zöllner. Anomaly detection in autonomous driving: A survey. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4488–4499, 2022.

[4] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4: Optimal speed and accuracy of object detection. *arXiv preprint arXiv:2004.10934*, 2020.

[5] Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan Mark Liao. Yolov7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 7464–7475, 2023.

[6] Ming Ma and Mengkun Guo. Pp-yolo-cs: A novel approach for real-time fire and smoke detection in industrial park environments. In *2024 International Joint Conference on Neural Networks (IJCNN)*, pages 1–7. IEEE, 2024.

[7] Xin Huang, Xinxin Wang, Wenyu Lv, Xiaying Bai, Xiang Long, Kaipeng Deng, Qingqing Dang, Shumin Han, Qiwen Liu, Xiaoguang Hu, et al. Pp-yolov2: A practical object detector. *arXiv preprint arXiv:2104.10419*, 2021.

[8] Chuyi Li, Lulu Li, Yifei Geng, Hongliang Jiang, Meng Cheng, Bo Zhang, Zaidan Ke, Xiaoming Xu, and Xiangxiang Chu. Yolov6 v3. 0: A full-scale reloading. *arXiv preprint arXiv:2301.05586*, 2023.

[9] Xiang Long, Kaipeng Deng, Guanzhong Wang, Yang Zhang, Qingqing Dang, Yuan Gao, Hui Shen, Jianguo Ren, Shumin Han, Errui Ding, et al. Pp-yolo: An effective and efficient implementation of object detector. *arXiv preprint arXiv:2007.12099*, 2020.

[10] Joseph Redmon. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[11] Z Ge. Yolox: Exceeding yolo series in 2021. *arXiv preprint arXiv:2107.08430*, 2021.

[12] Shuai Shao, Zeming Li, Tianyuan Zhang, Chao Peng, Gang Yu, Xiangyu Zhang, Jing Li, and Jian Sun. Objects365: A large-scale, high-quality dataset for object detection. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 8430–8439, 2019.

[13] J Redmon. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016.

[14] T-YLPG Ross and GKHP Dollár. Focal loss for dense object detection. In *proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2980–2988, 2017.

[15] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part I 14*, pages 21–37. Springer, 2016.

[16] Joseph Redmon and Ali Farhadi. Yolo9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7263–7271, 2017.

[17] Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan Mark Liao. Scaled-yolov4: Scaling cross stage partial network. In *Proceedings of the IEEE/cvf conference on computer vision and pattern recognition*, pages 13029–13038, 2021.