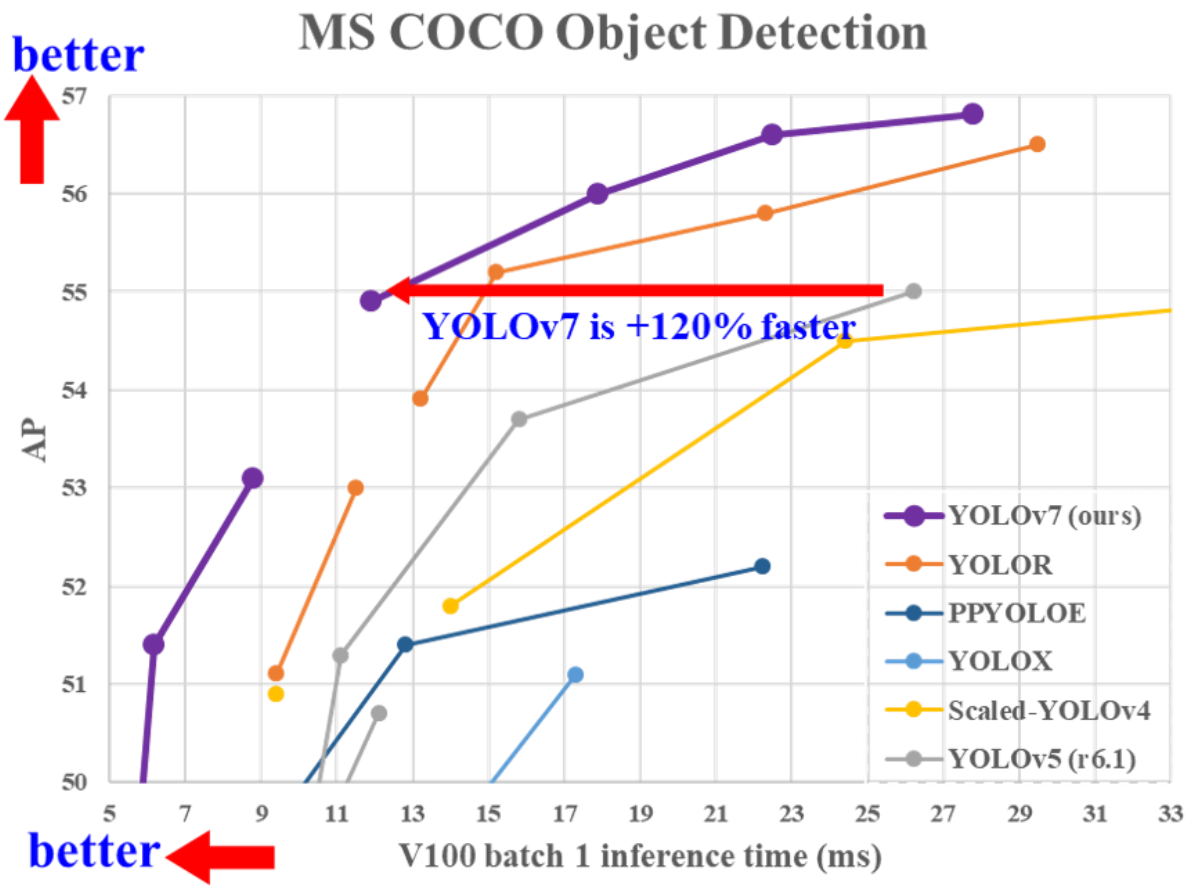
YOLOv7



Objectives

- Increase performance, training, and inference speed.
- Include the popular “Model re-parameterization” and “Dynamic label assignment” topics in the model.
- Focus on fixing SOTA issues related to:
    - More robust loss functions
    - More efficient label assignment method
    - More efficient training method

Contributions

- Designing and utilizing several tricks to enhance efficiency (bag-of-freebies methods):
    - Model re-parameterization: Planned re-parametrized convolution
    - Deep supervision: Coarse for auxiliary and fine for lead loss
    - Batch normalization in conv-bn-activation topology
    - Implicit knowledge from YOLOR
    - EMA model for final inference
- Designing “Extended-ELAN” architecture
- A new model-scaling method for concatenation-based models (such as E-ELAN)

Model re-parameterization

Model-level:

Two types:

- Train multiple identical models with different training data, and then average the weights of multiple trained models
- Perform a weighted average of the weights of models at different iteration number

Module-level (more popular):

Split a module into multiple identical or different module branches during training and integrates multiple branched modules into a completely equivalent module during inference.

Planned re-parameterized convolution (Designed for YOLOv7):

By removing the identity connection in RepConv, the re-parameterization module can be applied to residual and concatenation-based models such as ResNet and DenseNet.

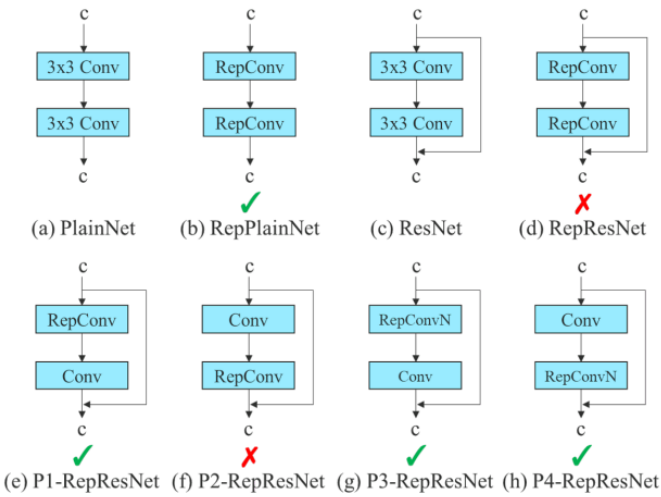


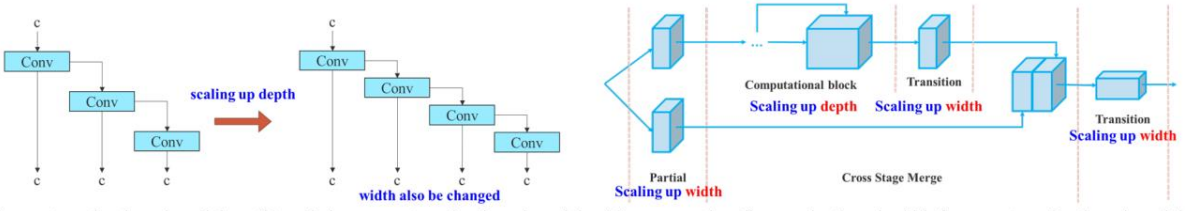
Figure 4: Planned re-parameterized model. In the proposed planned re-parameterized model, we found that a layer with residual or concatenation connections, its RepConv should not have identity connection. Under these circumstances, it can be replaced by RepConvN that contains no identity connections.

Model scaling

Model scaling is a way to scale up or down an already designed model and make it fit in different computing devices.

Factors such as depth and width of the model must be considered for the scaling. But in concatenation-based models (e.g., DenseNet, ELAN), these factors are not independent (e.g., increasing depth affects the width).

The below figure suggests a model scaling for concatenation-based models by scaling the depth only in computational blocks.



(a) concatenation-based model (b) scaled-up concatenation-based model (c) compound scaling up depth and width for concatenation-based model

Figure 3: Model scaling for concatenation-based models. From (a) to (b), we observe that when depth scaling is performed on concatenation-based models, the output width of a computational block also increases. This phenomenon will cause the input width of the subsequent transmission layer to increase. Therefore, we propose (c), that is, when performing model scaling on concatenation-based models, only the depth in a computational block needs to be scaled, and the remaining of transmission layer is performed with corresponding width scaling.

Extended-ELAN

ELAN controls the shortest longest gradient path to learn and converge more effectively using a deeper network. Extended-ELAN uses operations such as expand, shuffle, and merge cardinality to enhance the learning ability of the network without destroying the original gradient path.

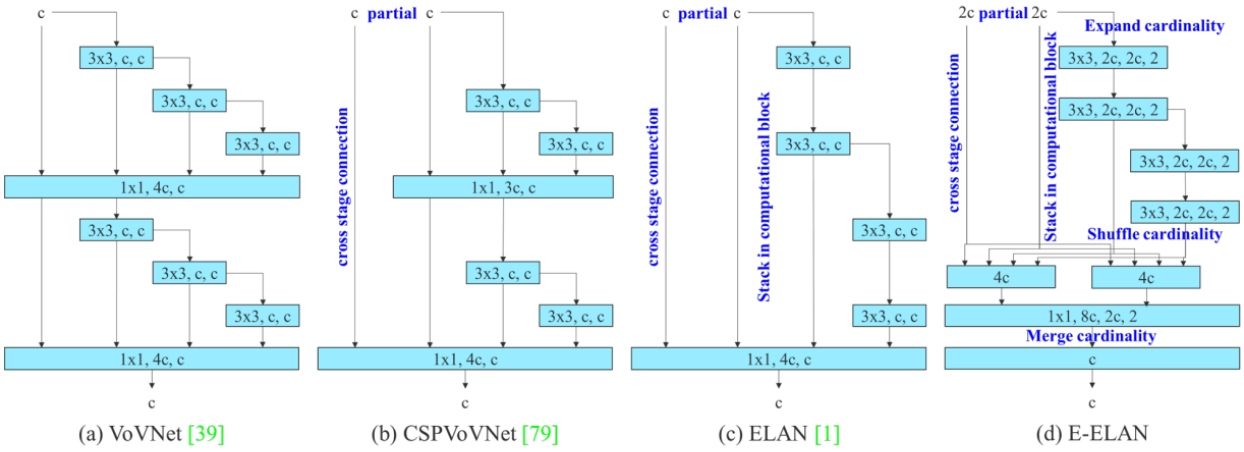


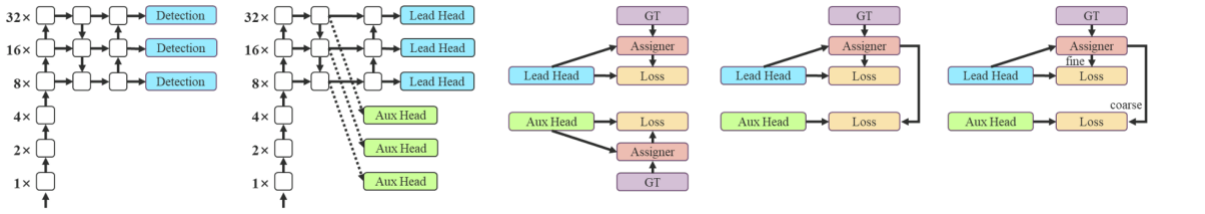
Figure 2: Extended efficient layer aggregation networks. The proposed extended ELAN (E-ELAN) does not change the gradient transmission path of the original architecture at all, but use group convolution to increase the cardinality of the added features, and combine the features of different groups in a shuffle and merge cardinality manner. This way of operation can enhance the features learned by different feature maps and improve the use of parameters and calculations.

Deep supervision & Label assignment

Deep supervision is a technique that is often used in training deep networks. Its main concept is to add extra auxiliary head in the middle layers of the network, and the shallow network weights with assistant loss as the guide.

In object detection task, instead of generating labels directly from ground truth labels, researchers often use quality and distribution of prediction output by the network, and then consider together with the ground truth to use some calculation and optimization methods to generate a **reliable soft label**.

Two new methods are designed in the paper to handle label assignment for auxiliary heads:



(a) Normal model (b) Model with auxiliary head (c) Independent assigner (d) Lead guided assigner (e) Coarse-to-fine lead guided assigner

Figure 5: Coarse for auxiliary and fine for lead head label assigner. Compare with normal model (a), the schema in (b) has auxiliary head. Different from the usual independent label assigner (c), we propose (d) lead head guided label assigner and (e) coarse-to-fine lead head guided label assigner. The proposed label assigner is optimized by lead head prediction and the ground truth to get the labels of training lead head and auxiliary head at the same time. The detailed coarse-to-fine implementation method and constraint design details will be elaborated in Apendix.

- **Lead head guided label assigner:** Auxiliary head's label is calculated using GT and lead head's label, instead of calculating the labels independently for the heads. The reason to do this is because lead head has a relatively strong learning capability, so the soft label generated from it should be more representative of the distribution and correlation between the source data and the target.
- **Coarse-to-fine lead head guided label assigner:** Fine label is the same as the soft label generated by lead head guided label assigner, and coarse label is generated by allowing more grids to be treated as positive target by relaxing the constraints of the positive sample assignment process. The reason for this is that the learning ability of an auxiliary head is not as strong as that of a lead head, and in order to avoid losing the information that needs to be learned, we will focus on optimizing the recall of auxiliary head in the object detection task.

Experiments

Baseline Comparison

Table 1: Comparison of baseline object detectors.

Model	#Param.	FLOPs	Size	AP^{val}	AP^{val}_{50}	AP^{val}_{75}	AP^{val}_{S}	AP^{val}_{M}	AP^{val}_{L}
YOLOv4 [3]	64.4M	142.8G	640	49.7%	68.2%	54.3%	32.9%	54.8%	63.7%
YOLOR-u5 (r6.1) [81]	46.5M	109.1G	640	50.2%	68.7%	54.6%	33.2%	55.5%	63.7%
YOLOv4-CSP [79]	52.9M	120.4G	640	50.3%	68.6%	54.9%	34.2%	55.6%	65.1%
YOLOR-CSP [81]	52.9M	120.4G	640	50.8%	69.5%	55.3%	33.7%	56.0%	65.4%
YOLOv7	36.9M	104.7G	640	**51.2%**	**69.7%**	**55.5%**	**35.2%**	**56.0%**	**66.7%**
improvement	-43%	-15%	-	+0.4	+0.2	+0.2	+1.5	=	+1.3
YOLOR-CSP-X [81]	96.9M	226.8G	640	52.7%	**71.3%**	57.4%	36.3%	57.5%	68.3%
YOLOv7-X	71.3M	189.9G	640	**52.9%**	71.1%	**57.5%**	**36.9%**	**57.7%**	**68.6%**
improvement	-36%	-19%	-	+0.2	-0.2	+0.1	+0.6	+0.2	+0.3
YOLOv4-tiny [79]	6.1	6.9	416	24.9%	42.1%	25.7%	8.7%	28.4%	39.2%
YOLOv7-tiny	6.2	5.8	416	**35.2%**	**52.8%**	**37.3%**	**15.7%**	**38.0%**	**53.4%**
improvement	+2%	-19%	-	+10.3	+10.7	+11.6	+7.0	+9.6	+14.2
YOLOv4-tiny-3l [79]	8.7	5.2	320	30.8%	47.3%	32.2%	**10.9%**	31.9%	51.5%
YOLOv7-tiny	6.2	3.5	320	**30.8%**	**47.3%**	**32.2%**	10.0%	**31.9%**	**52.2%**
improvement	-39%	-49%	-	=	=	=	-0.9	=	+0.7
YOLOR-E6 [81]	115.8M	683.2G	1280	55.7%	73.2%	60.7%	40.1%	**60.4%**	69.2%
YOLOv7-E6	97.2M	515.2G	1280	**55.9%**	**73.5%**	**61.1%**	**40.6%**	60.3%	**70.0%**
improvement	-19%	-33%	-	+0.2	+0.3	+0.4	+0.5	-0.1	+0.8
YOLOR-D6 [81]	151.7M	935.6G	1280	56.1%	73.9%	61.2%	**42.4%**	60.5%	69.9%
YOLOv7-D6	154.7M	806.8G	1280	56.3%	73.8%	61.4%	41.3%	60.6%	70.1%
YOLOv7-E6E	151.7M	843.2G	1280	**56.8%**	**74.4%**	**62.1%**	40.8%	**62.1%**	**70.6%**
improvement	=	-11%	-	+0.7	+0.5	+0.9	-1.6	+1.6	+0.7

Real-time

Table 2: Comparison of state-of-the-art real-time object detectors.

Model	#Param.	FLOPs	Size	FPS	AP^{test} / AP^{val}	AP^{test}_{50}	AP^{test}_{75}	AP^{test}_{S}	AP^{test}_{M}	AP^{test}_{L}
YOLOX-S [21]	9.0M	26.8G	640	102	40.5% / 40.5%	-	-	-	-	-
YOLOX-M [21]	25.3M	73.8G	640	81	47.2% / 46.9%	-	-	-	-	-
YOLOX-L [21]	54.2M	155.6G	640	69	50.1% / 49.7%	-	-	-	-	-
YOLOX-X [21]	99.1M	281.9G	640	58	51.5% / 51.1%	-	-	-	-	-
PPYOLOE-S [85]	7.9M	17.4G	640	208	43.1% / 42.7%	60.5%	46.6%	23.2%	46.4%	56.9%
PPYOLOE-M [85]	23.4M	49.9G	640	123	48.9% / 48.6%	66.5%	53.0%	28.6%	52.9%	63.8%
PPYOLOE-L [85]	52.2M	110.1G	640	78	51.4% / 50.9%	68.9%	55.6%	31.4%	55.3%	66.1%
PPYOLOE-X [85]	98.4M	206.6G	640	45	52.2% / 51.9%	69.9%	56.5%	33.3%	56.3%	66.4%
YOLOv5-N (r6.1) [23]	1.9M	4.5G	640	159	- / 28.0%	-	-	-	-	-
YOLOv5-S (r6.1) [23]	7.2M	16.5G	640	156	- / 37.4%	-	-	-	-	-
YOLOv5-M (r6.1) [23]	21.2M	49.0G	640	122	- / 45.4%	-	-	-	-	-
YOLOv5-L (r6.1) [23]	46.5M	109.1G	640	99	- / 49.0%	-	-	-	-	-
YOLOv5-X (r6.1) [23]	86.7M	205.7G	640	83	- / 50.7%	-	-	-	-	-
YOLOR-CSP [81]	52.9M	120.4G	640	106	51.1% / 50.8%	69.6%	55.7%	31.7%	55.3%	64.7%
YOLOR-CSP-X [81]	96.9M	226.8G	640	87	53.0% / 52.7%	71.4%	57.9%	33.7%	57.1%	66.8%
YOLOv7-tiny-SiLU	6.2M	13.8G	640	286	38.7% / 38.7%	56.7%	41.7%	18.8%	42.4%	51.9%
YOLOv7	36.9M	104.7G	640	161	51.4% / 51.2%	69.7%	55.9%	31.8%	55.5%	65.0%
YOLOv7-X	71.3M	189.9G	640	114	53.1% / 52.9%	71.2%	57.8%	33.8%	57.1%	67.4%
YOLOv5-N6 (r6.1) [23]	3.2M	18.4G	1280	123	- / 36.0%	-	-	-	-	-
YOLOv5-S6 (r6.1) [23]	12.6M	67.2G	1280	122	- / 44.8%	-	-	-	-	-
YOLOv5-M6 (r6.1) [23]	35.7M	200.0G	1280	90	- / 51.3%	-	-	-	-	-
YOLOv5-L6 (r6.1) [23]	76.8M	445.6G	1280	63	- / 53.7%	-	-	-	-	-
YOLOv5-X6 (r6.1) [23]	140.7M	839.2G	1280	38	- / 55.0%	-	-	-	-	-
YOLOR-P6 [81]	37.2M	325.6G	1280	76	53.9% / 53.5%	71.4%	58.9%	36.1%	57.7%	65.6%
YOLOR-W6 [81]	79.8G	453.2G	1280	66	55.2% / 54.8%	72.7%	60.5%	37.7%	59.1%	67.1%
YOLOR-E6 [81]	115.8M	683.2G	1280	45	55.8% / 55.7%	73.4%	61.1%	38.4%	59.7%	67.7%
YOLOR-D6 [81]	151.7M	935.6G	1280	34	56.5% / 56.1%	74.1%	61.9%	38.9%	60.4%	68.7%
YOLOv7-W6	70.4M	360.0G	1280	84	54.9% / 54.6%	72.6%	60.1%	37.3%	58.7%	67.1%
YOLOv7-E6	97.2M	515.2G	1280	56	56.0% / 55.9%	73.5%	61.2%	38.0%	59.9%	68.4%
YOLOv7-D6	154.7M	806.8G	1280	44	56.6% / 56.3%	74.0%	61.8%	38.8%	60.1%	69.5%
YOLOv7-E6E	151.7M	843.2G	1280	36	56.8% / 56.8%	74.4%	62.1%	39.3%	60.5%	69.0%

[1] Our FLOPs is calaculated by rectangle input resolution like 640 × 640 or 1280 × 1280.
[2] Our inference time is estimated by using letterbox resize input image to make its long side equals to 640 or 1280.

No extra training data

Table 9: More comparison (batch=1, no-TRT, without extra object detection training data)

Model	#Param.	FLOPs	Size	FPS^{V100}	AP^{test} / AP^{val}	AP^{test}_{50}	AP^{test}_{75}
YOLOv7-tiny-SiLU	6.2M	13.8G	640	286	**38.7% / 38.7%**	**56.7%**	**41.7%**
PPYOLOE-S [85]	7.9M	17.4G	640	208	**43.1% / 42.7%**	**60.5%**	**46.6%**
YOLOv7	36.9M	104.7G	640	161	**51.4% / 51.2%**	**69.7%**	**55.9%**
YOLOv5-N (r6.1) [23]	1.9M	4.5G	640	159	- / 28.0%	-	-
YOLOv5-S (r6.1) [23]	7.2M	16.5G	640	156	- / 37.4%	-	-
PPYOLOE-M [85]	23.4M	49.9G	640	123	48.9% / 48.6%	66.5%	53.0%
YOLOv5-N6 (r6.1) [23]	3.2M	18.4G	1280	123	- / 36.0%	-	-
YOLOv5-S6 (r6.1) [23]	12.6M	67.2G	1280	122	- / 44.8%	-	-
YOLOv5-M (r6.1) [23]	21.2M	49.0G	640	122	- / 45.4%	-	-
YOLOv7-X	71.3M	189.9G	640	114	**53.1% / 52.9%**	**71.2%**	**57.8%**
YOLOR-CSP [81]	52.9M	120.4G	640	106	51.1% / 50.8%	69.6%	55.7%
YOLOX-S [21]	9.0M	26.8G	640	102	40.5% / 40.5%	-	-
YOLOv5-L (r6.1) [23]	46.5M	109.1G	640	99	- / 49.0%	-	-
YOLOv5-M6 (r6.1) [23]	35.7M	200.0G	1280	90	- / 51.3%	-	-
YOLOR-CSP-X [81]	96.9M	226.8G	640	87	53.0% / 52.7%	**71.4%**	**57.9%**
YOLOv7-W6	70.4M	360.0G	1280	84	**54.9% / 54.6%**	**72.6%**	**60.1%**
YOLOv5-X (r6.1) [23]	86.7M	205.7G	640	83	- / 50.7%	-	-
YOLOX-M [21]	25.3M	73.8G	640	81	47.2% / 46.9%	-	-
PPYOLOE-L [85]	52.2M	110.1G	640	78	51.4% / 50.9%	68.9%	55.6%
YOLOR-P6 [81]	37.2M	325.6G	1280	76	53.9% / 53.5%	71.4%	58.9%
YOLOX-L [21]	54.2M	155.6G	640	69	50.1% / 49.7%	-	-
YOLOR-W6 [81]	79.8G	453.2G	1280	66	**55.2% / 54.8%**	**72.7%**	**60.5%**
YOLOv5-L6 (r6.1) [23]	76.8M	445.6G	1280	63	- / 53.7%	-	-
YOLOX-X [21]	99.1M	281.9G	640	58	51.5% / 51.1%	-	-
YOLOv7-E6	97.2M	515.2G	1280	56	**56.0% / 55.9%**	**73.5%**	**61.2%**
YOLOR-E6 [81]	115.8M	683.2G	1280	45	55.8% / 55.7%	73.4%	61.1%
PPYOLOE-X [85]	98.4M	206.6G	640	45	52.2% / 51.9%	69.9%	56.5%
YOLOv7-D6	154.7M	806.8G	1280	44	**56.6% / 56.3%**	**74.0%**	**61.8%**
YOLOv5-X6 (r6.1) [23]	140.7M	839.2G	1280	38	- / 55.0%	-	-
YOLOv7-E6E	151.7M	843.2G	1280	36	**56.8% / 56.8%**	**74.4%**	**62.1%**
YOLOR-D6 [81]	151.7M	935.6G	1280	34	56.5% / 56.1%	**74.1%**	**61.9%**
F-RCNN-R101-FPN+ [5]	60.0M	246.0G	1333	20	- / 44.0%	-	-
Deformable DETR [100]	40.0M	173.0G	-	19	- / 46.2%	-	-
Swin-B (C-M-RCNN) [52]	145.0M	982.0G	1333	11.6	- / 51.9%	-	-
DETR DC5-R101 [5]	60.0M	253.0G	1333	10	- / 44.9%	-	-
EfficientDet-D7x [74]	77.0M	410.0G	1536	6.5	55.1% / 54.4%	72.4%	58.4%
Dual-Swin-T (C-M-RCNN) [47]	113.8M	836.0G	1333	6.5	- / 53.6%	-	-
ViT-Adapter-B [7]	122.0M	997.0G	-	4.4	- / 50.8%	-	-
Dual-Swin-B (HTC) [47]	235.0M	-	1600	2.5	**58.7% / 58.4%**	-	-
Dual-Swin-L (HTC) [47]	453.0M	-	1600	1.5	**59.4% / 59.1%**	-	-

Model	#Param.	FLOPs	Size	FPS^{A100}	AP^{test} / AP^{val}	AP^{test}_{50}	AP^{test}_{75}
DN-Deformable-DETR [41]	48.0M	265.0G	1333	23.0	- / 48.6%	-	-
ConvNeXt-B (C-M-RCNN) [53]	-	964.0G	1280	11.5	- / 54.0%	73.1%	58.8%
Swin-B (C-M-RCNN) [52]	-	982.0G	1280	10.7	- / 53.0%	71.8%	57.5%
DINO-5scale (R50) [89]	47.0M	860.0G	1333	10.0	- / 51.0%	-	-
ConvNeXt-L (C-M-RCNN) [53]	-	1354.0G	1280	10.0	- / 54.8%	73.8%	59.8%
Swin-L (C-M-RCNN) [52]	-	1382.0G	1280	9.2	- / 53.9%	72.4%	58.8%
ConvNeXt-XL (C-M-RCNN) [53]	-	1898.0G	1280	8.6	- / 55.2%	74.2%	59.9%

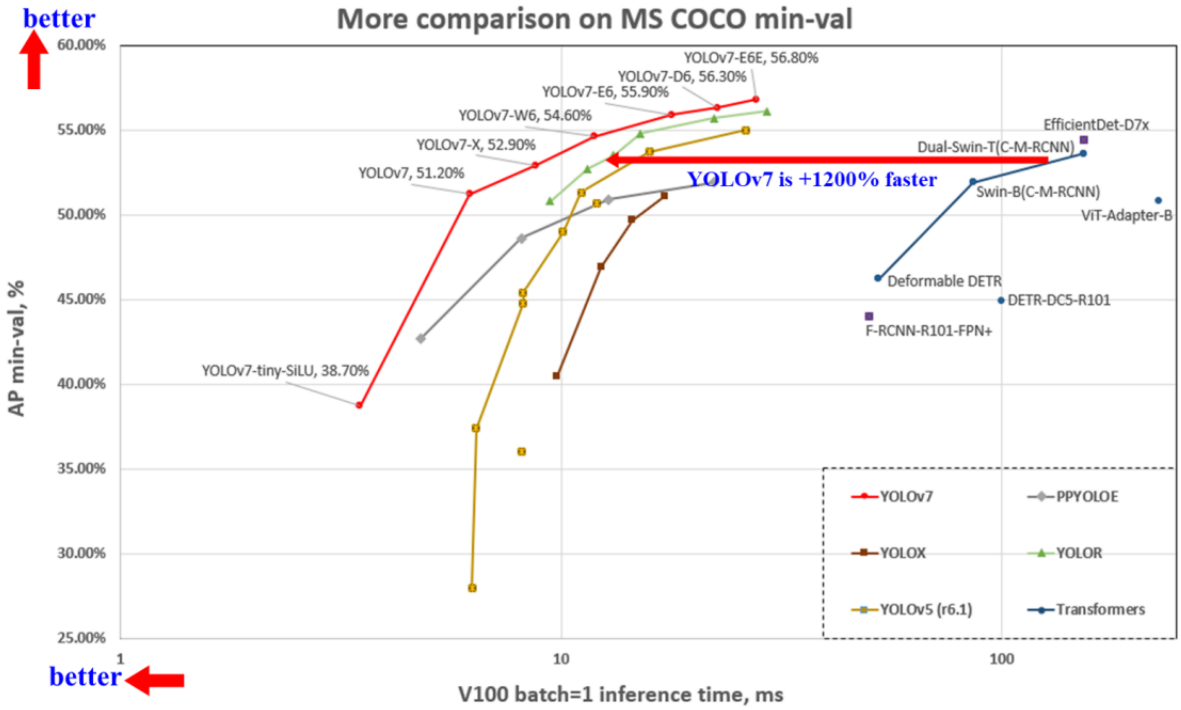


Figure 9: Comparison with other object detectors.

Comparison to DINO

- Dual-Swin-L (HTC) 1600x1600 - 59.1% AP - 1.5 FPS V100 - **isn't real-time** - is 2000% FSP slower than YOLOv7-e6e
- Dual-Swin-L(HTC, multi-scale) - 60.1% AP - 0.3 FPS V100 - **isn't real-time** - is 12000% FPS slower than YOLOv7-e6e
- DINO-5scale-R50 (10 FPS, 51.0% AP) is less accurate and 1500% FPS **slower than YOLOv7** (161 FPS, 51.2% AP)
- DINO(Swim-L,multi-scale) with 63.3 box AP - additional training datasets are used (so no fair comparison), no publicly available code and models, it is slower than 1 FPS - **isn't real-time** is ~10000% slower than YOLOv7-e6e