Introduction to Real-Time Object Detection

欣子豪

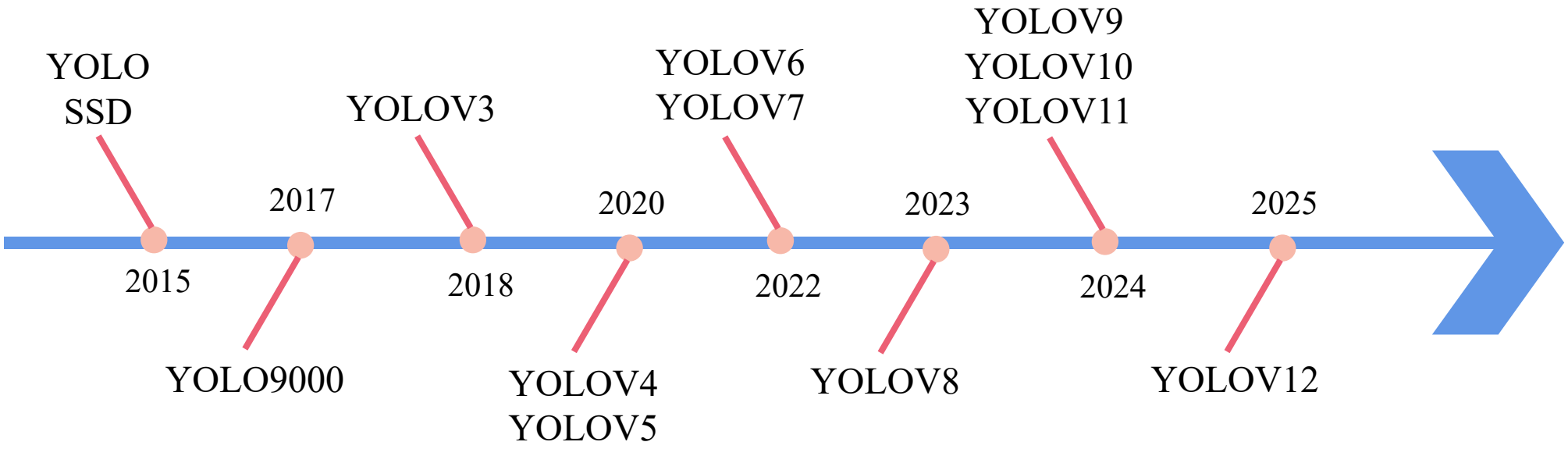
2025.2.24

1. Development of the YOLO series

2. RT-DETR based models

3. Hardware-Software Collaboration -- DeepSeek

Contents

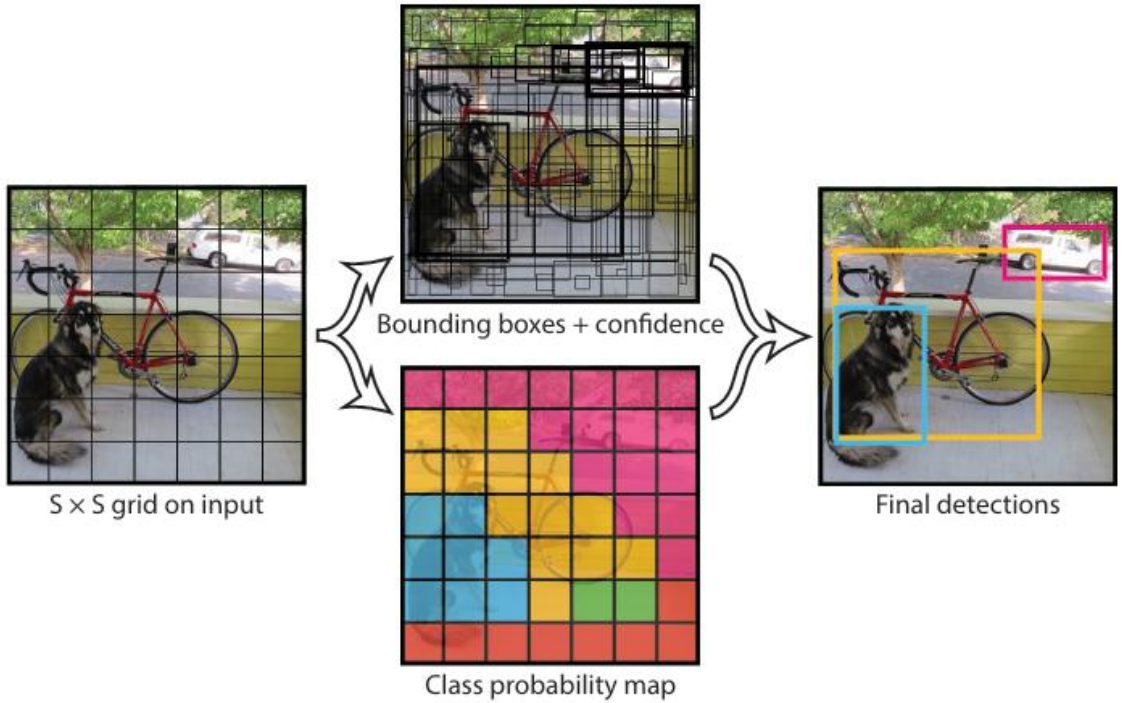


YOLO

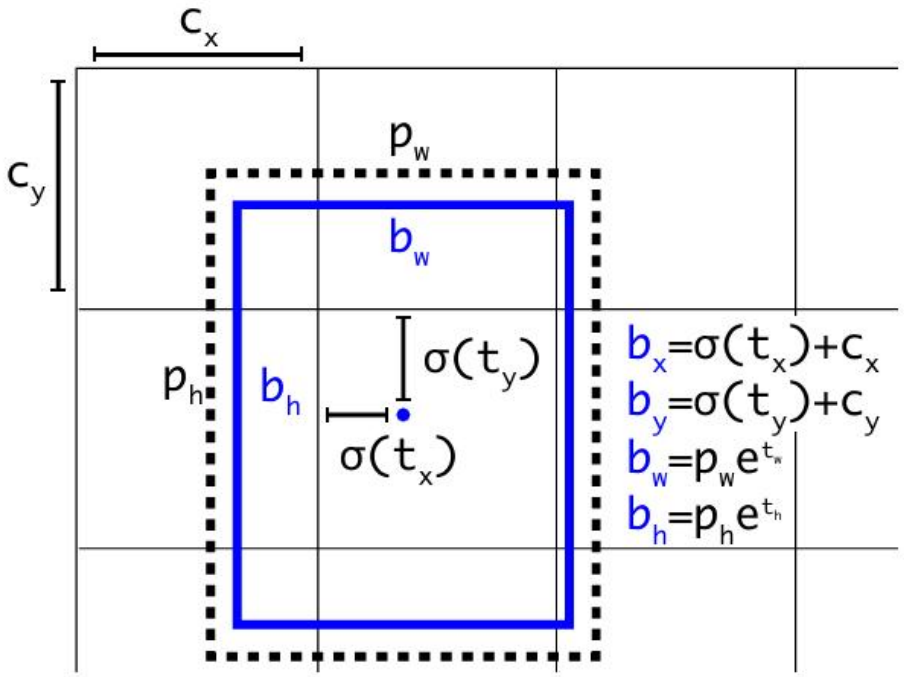
RT-DETR(2024)

RT-DETR
RT-DETR V2
RT-DETR V3
DFINE
DEIM

YOLO



YOLO (2015)
One-stage framework
Divide the image into SxS grid.



YOLO9000 (2017)
Anchor Boxes
Batch Normalization
Multi-Scale Training (320-608 pixel)

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

YOLOV3 (2018)

Darknet-19 -> Darknet-53

FPN

Focal loss, GIoU loss

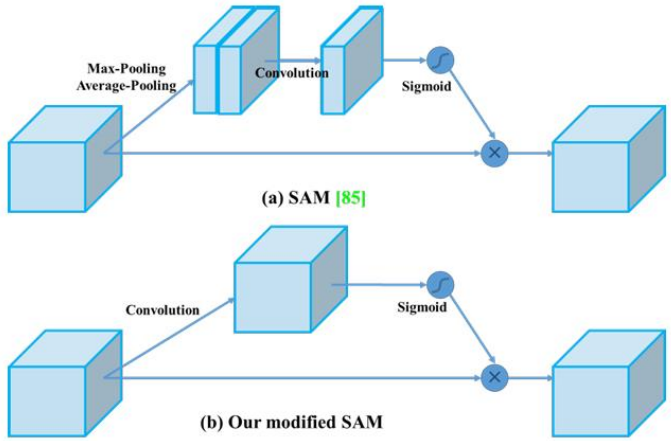


Figure 5: Modified SAM.

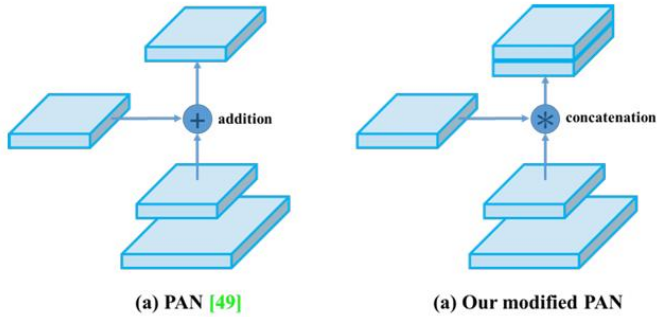


Figure 6: Modified PAN.

YOLOV4 (2020)

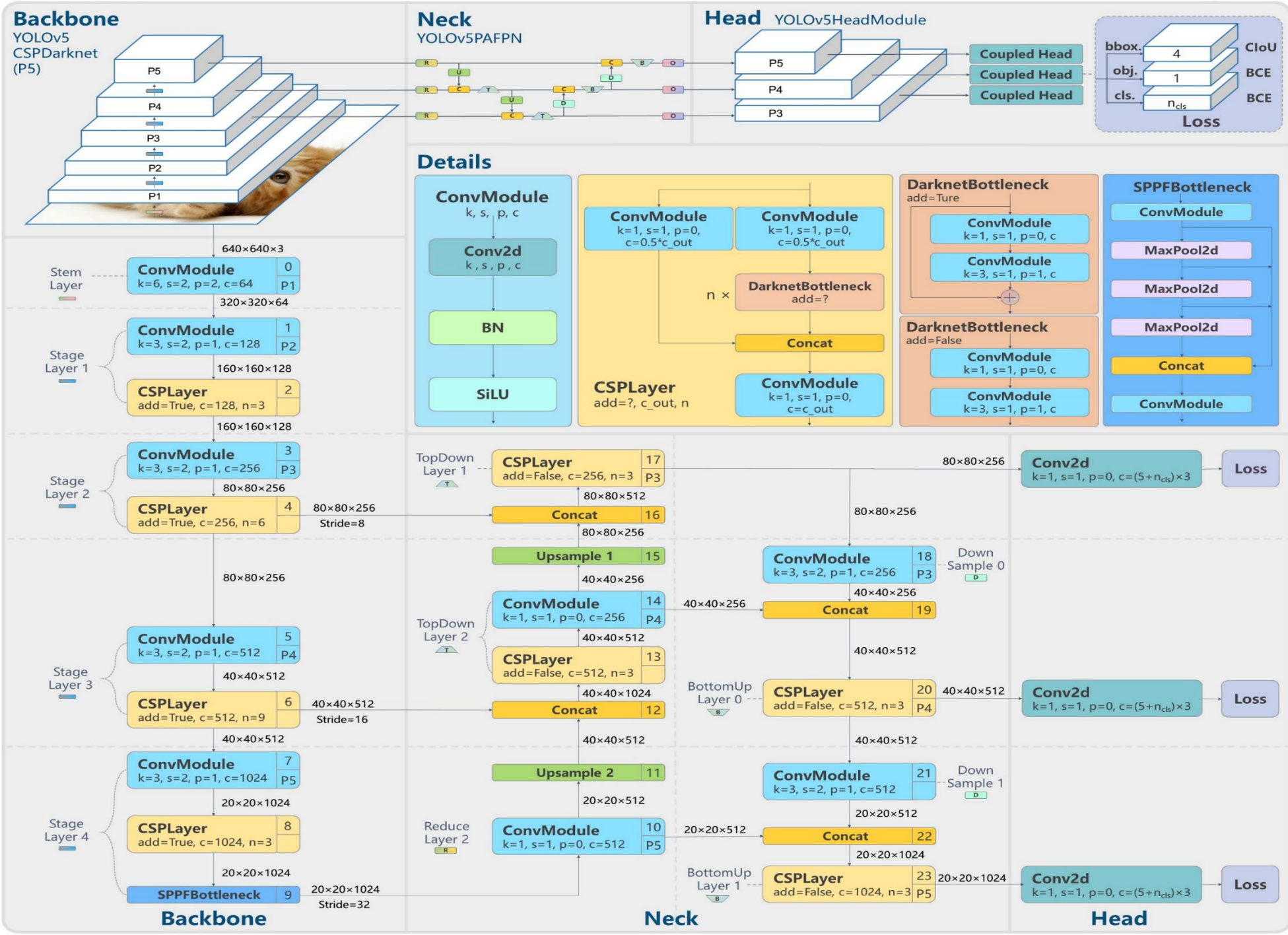
CSPDarknet-53

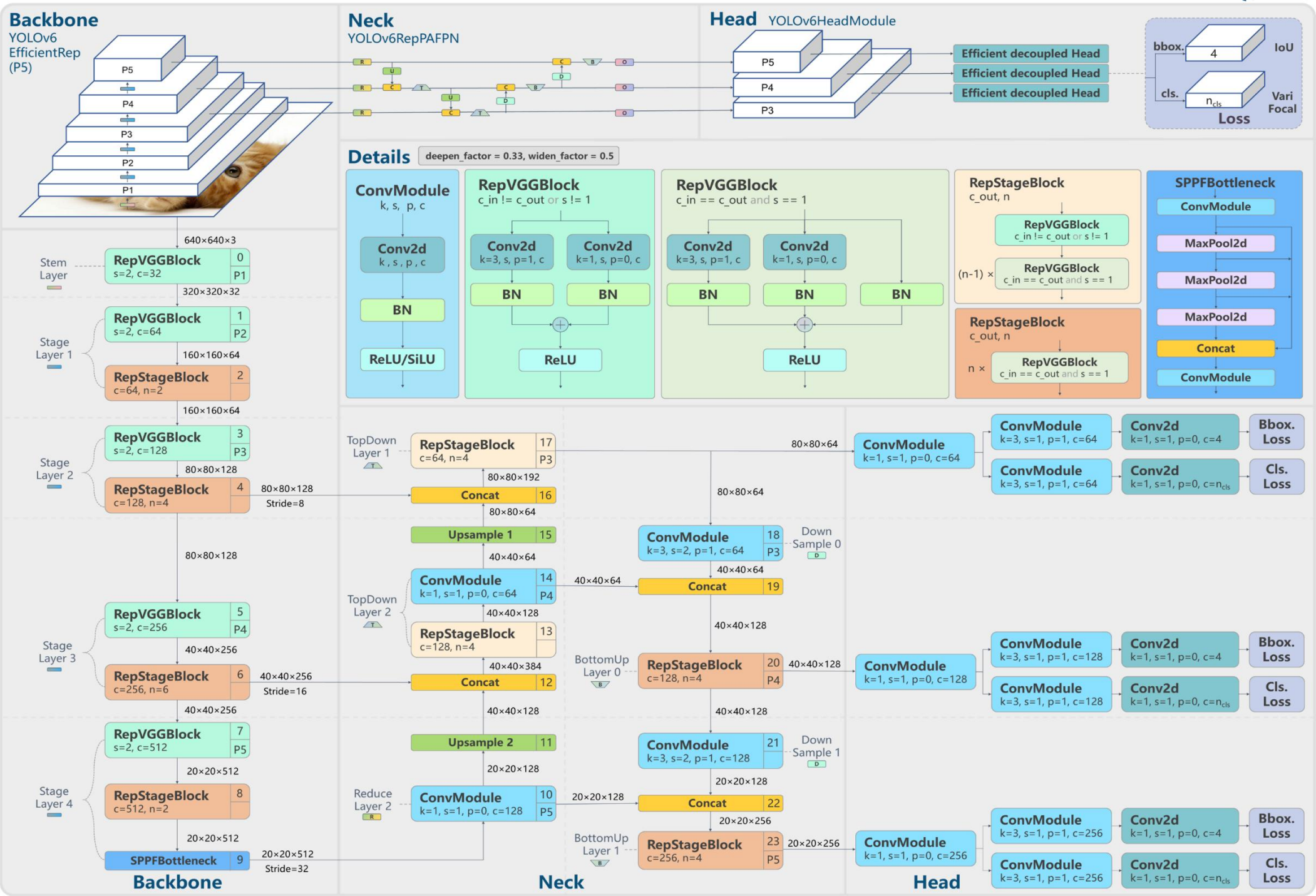
PAN, SAM

Mosaic Aug

Mish action

CIoU loss


Backbone
YOLOv5
CSPDarknet
(P5)
P5
P4
P3
P2
P1
Neck
YOLOv5PAFPN
Head
YOLOv5HeadModule
Coupled Head
Coupled Head
Coupled Head
bbox.
obj.
cls.
4
1
n_{cls}
CIoU
BCE
BCE
Loss
Details
ConvModule
k, s, p, c
Conv2d
k , s , p , c
BN
SiLU
ConvModule
k=1, s=1, p=0, c=0.5*c_out
ConvModule
k=1, s=1, p=0, c=0.5*c_out
n ×
DarknetBottleneck
add=?
Concat
CSPLayer
add=?, c_out, n
ConvModule
k=1, s=1, p=0, c=c_out
DarknetBottleneck
add=Ture
ConvModule
k=1, s=1, p=0, c
ConvModule
k=3, s=1, p=1, c
DarknetBottleneck
add=False
ConvModule
k=1, s=1, p=0, c
ConvModule
k=3, s=1, p=1, c
SPPFBottleneck
ConvModule
MaxPool2d
MaxPool2d
MaxPool2d
Concat
ConvModule
640×640×3
Stem Layer
ConvModule
k=6, s=2, p=2, c=64
0
P1
320×320×64
Stage Layer 1
ConvModule
k=3, s=2, p=1, c=128
1
P2
160×160×128
CSPLayer
add=True, c=128, n=3
2
160×160×128
Stage Layer 2
ConvModule
k=3, s=2, p=1, c=256
3
P3
80×80×256
CSPLayer
add=True, c=256, n=6
4
80×80×256
Stride=8
80×80×256
Stage Layer 3
ConvModule
k=3, s=2, p=1, c=512
5
P4
40×40×512
CSPLayer
add=True, c=512, n=9
6
40×40×512
Stride=16
40×40×512
Stage Layer 4
ConvModule
k=3, s=2, p=1, c=1024
7
P5
20×20×1024
CSPLayer
add=True, c=1024, n=3
8
20×20×1024
SPPFBottleneck
9
20×20×1024
Stride=32
Backbone
TopDown Layer 1
CSPLayer
add=False, c=256, n=3
17
P3
80×80×512
Concat
16
80×80×256
Upsample 1
15
40×40×256
TopDown Layer 2
ConvModule
k=1, s=1, p=0, c=256
14
P4
40×40×512
CSPLayer
add=False, c=512, n=3
13
40×40×1024
Concat
12
40×40×512
Upsample 2
11
20×20×512
Reduce Layer 2
ConvModule
k=1, s=1, p=0, c=512
10
P5
Neck
80×80×256
Conv2d
k=1, s=1, p=0, c=(5+n_{cls})×3
Loss
80×80×256
ConvModule
k=3, s=2, p=1, c=256
18
P3
Down Sample 0
40×40×256
40×40×256
Concat
19
40×40×512
BottomUp Layer 0
CSPLayer
add=False, c=512, n=3
20
P4
40×40×512
Conv2d
k=1, s=1, p=0, c=(5+n_{cls})×3
Loss
40×40×512
ConvModule
k=3, s=2, p=1, c=512
21
Down Sample 1
20×20×512
20×20×512
Concat
22
20×20×1024
BottomUp Layer 1
CSPLayer
add=False, c=1024, n=3
23
P5
20×20×1024
Conv2d
k=1, s=1, p=0, c=(5+n_{cls})×3
Loss
Head



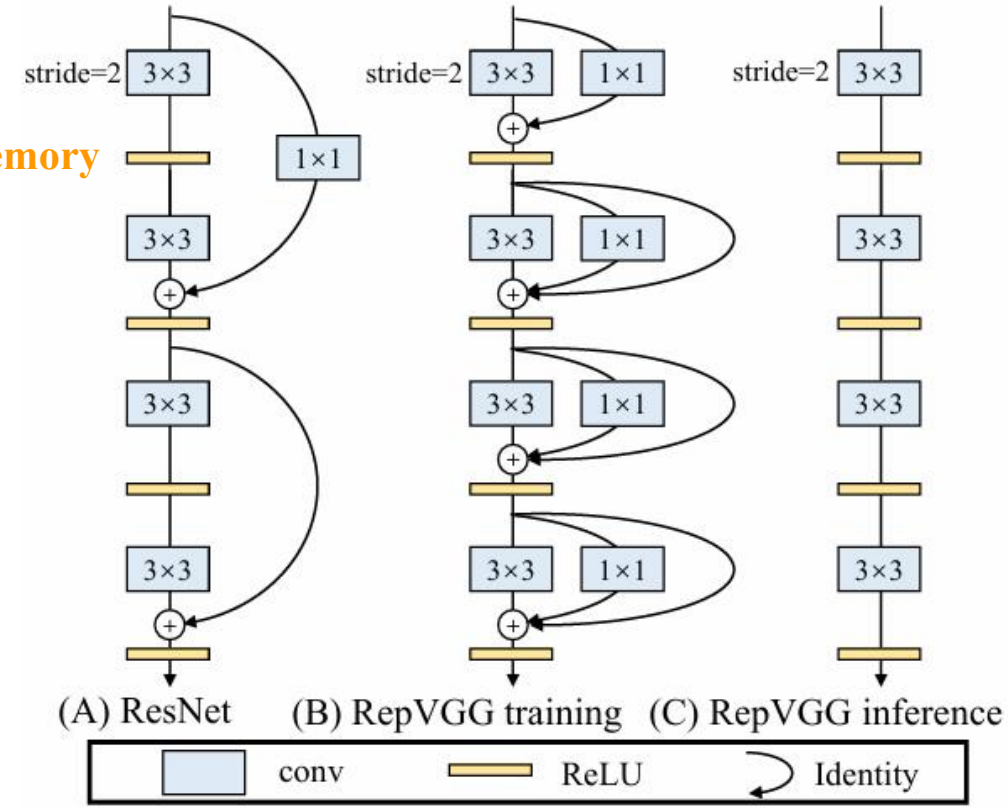
ParNeC 模式识别与神经计算研究组 PAttern Recognition and NEural Computing

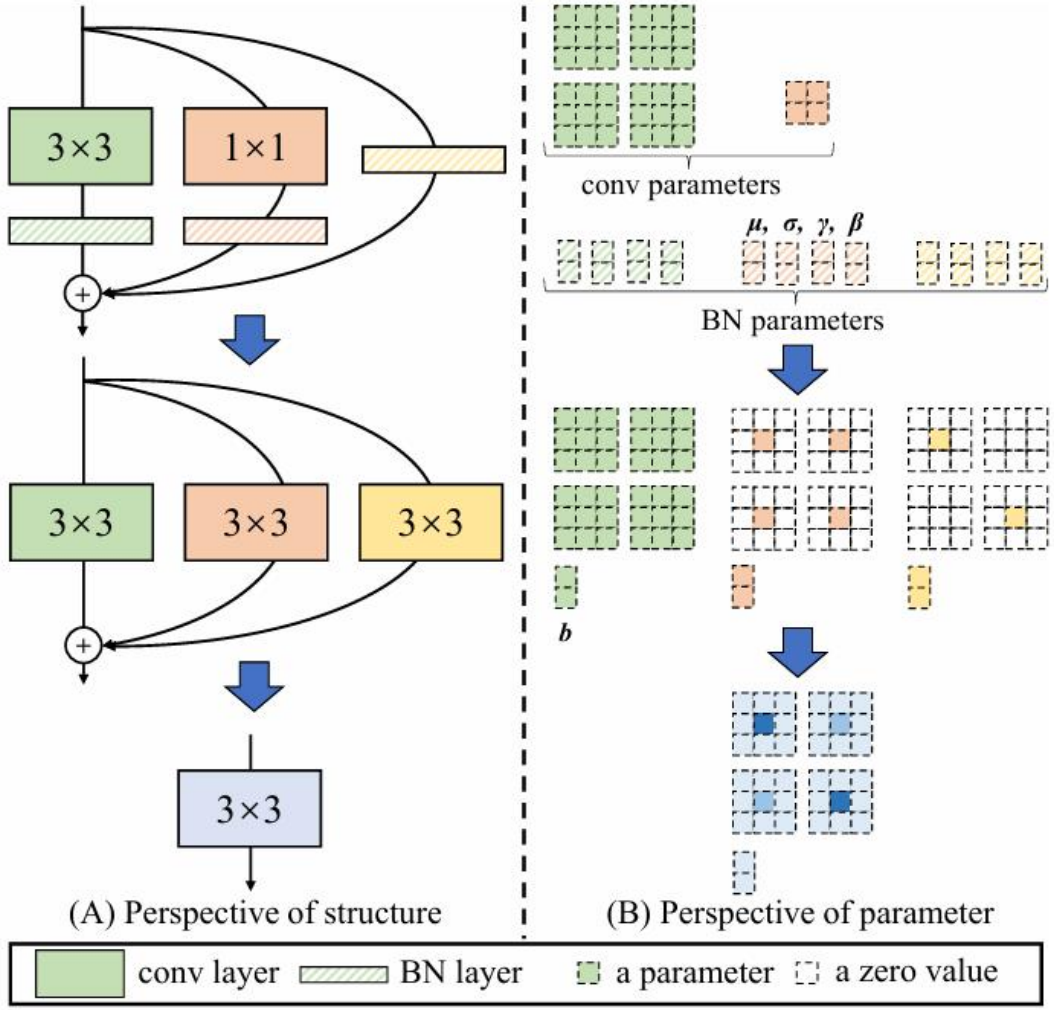
YOLOV6

RepVGGBlock
Decoupled Head
SimOTA Assignment

YOLOV6

2 x memory





Ding X, Zhang X, Ma N, et al. Repvgg: Making vgg-style convnets great again[C]//Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. 2021: 13733-13742.

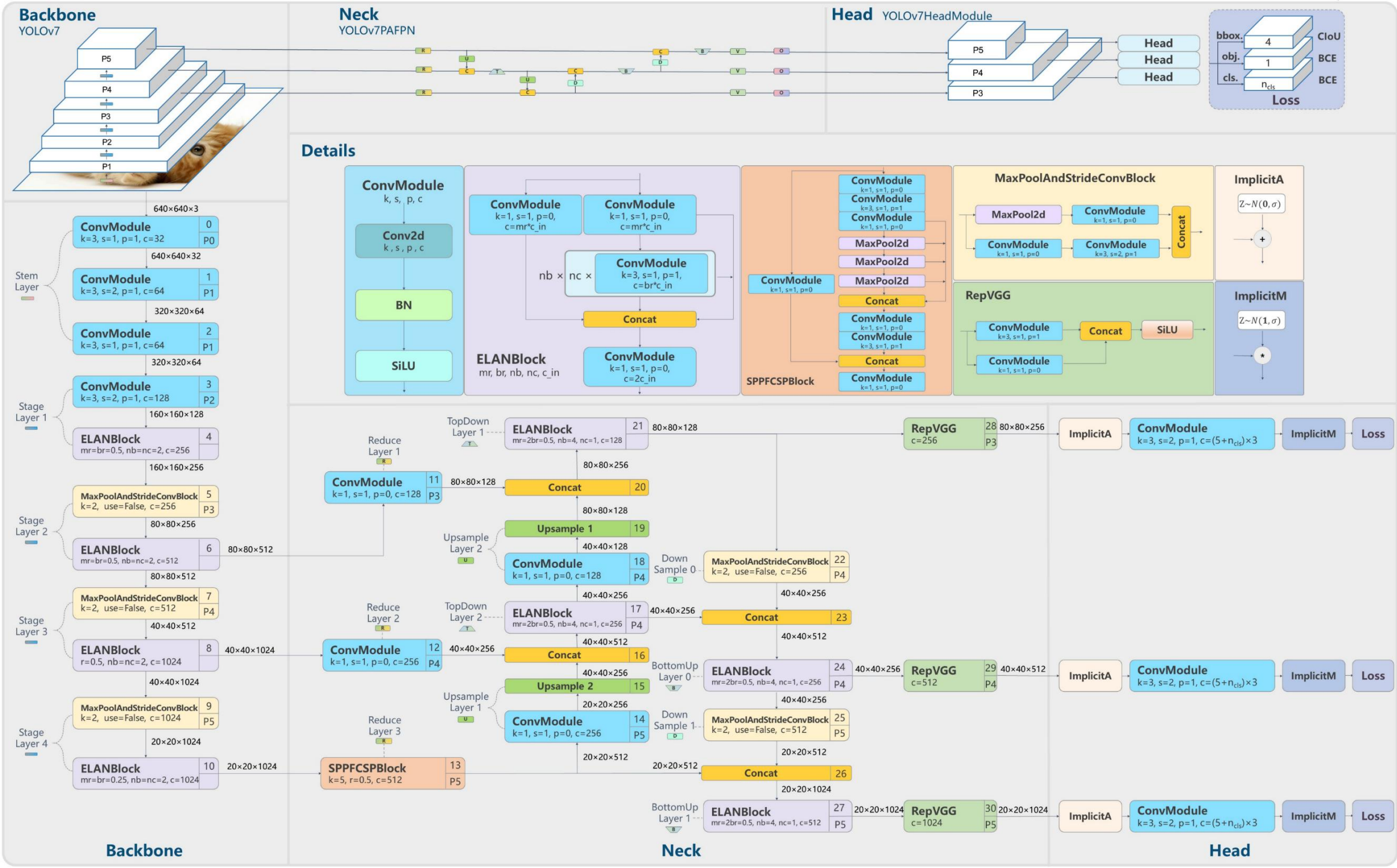
YOLOV6

Method	Input Size	AP^{val}	AP^{val}_{50}	FPS (bs=1)	FPS (bs=32)	Latency (bs=1)	Params	FLOPs
YOLOv5-N [10]	640	28.0%	45.7%	602	735	1.7 ms	1.9 M	4.5 G
YOLOv5-S [10]	640	37.4%	56.8%	376	444	2.7 ms	7.2 M	16.5 G
YOLOv5-M [10]	640	45.4%	64.1%	182	209	5.5 ms	21.2 M	49.0 G
YOLOv5-L [10]	640	49.0%	67.3%	113	126	8.8 ms	46.5 M	109.1 G
YOLOX-Tiny [7]	416	32.8%	50.3%*	717	1143	1.4 ms	5.1 M	6.5 G
YOLOX-S [7]	640	40.5%	59.3%*	333	396	3.0 ms	9.0 M	26.8 G
YOLOX-M [7]	640	46.9%	65.6%*	155	179	6.4 ms	25.3 M	73.8 G
YOLOX-L [7]	640	49.7%	68.0%*	94	103	10.6 ms	54.2 M	155.6 G
PPYOLOE-S [45]	640	43.1%	59.6%	327	419	3.1 ms	7.9 M	17.4 G
PPYOLOE-M [45]	640	49.0%	65.9%	152	189	6.6 ms	23.4 M	49.9 G
PPYOLOE-L [45]	640	51.4%	68.6%	101	127	10.1 ms	52.2 M	110.1 G
YOLOv7-Tiny [42]	416	33.3%*	49.9%*	787	1196	1.3 ms	6.2 M	5.8 G
YOLOv7-Tiny [42]	640	37.4%*	55.2%*	424	519	2.4 ms	6.2 M	13.7 G*
YOLOv7 [42]	640	51.2%	69.7%	110	122	9.0 ms	36.9 M	104.7 G
YOLOv6-N	640	35.9%	51.2%	802	1234	1.2 ms	4.3 M	11.1 G
YOLOv6-T	640	40.3%	56.6%	449	659	2.2 ms	15.0 M	36.7 G
YOLOv6-S	640	43.5%	60.4%	358	495	2.8 ms	17.2 M	44.2 G
YOLOv6-M‡	640	49.5%	66.8%	179	233	5.6 ms	34.3 M	82.2 G
YOLOv6-L-ReLU‡	640	51.7%	69.2%	113	149	8.8 ms	58.5 M	144.0 G
YOLOv6-L‡	640	52.5%	70.0%	98	121	10.2 ms	58.5 M	144.0 G

Kernel size	Theoretical FLOPs (B)	Time usage (ms)	Theoretical TFLOPS
1×1	420.9	84.5	9.96
3×3	3788.1	198.8	**38.10**
5×5	10522.6	2092.5	10.57
7×7	20624.4	4394.3	9.38

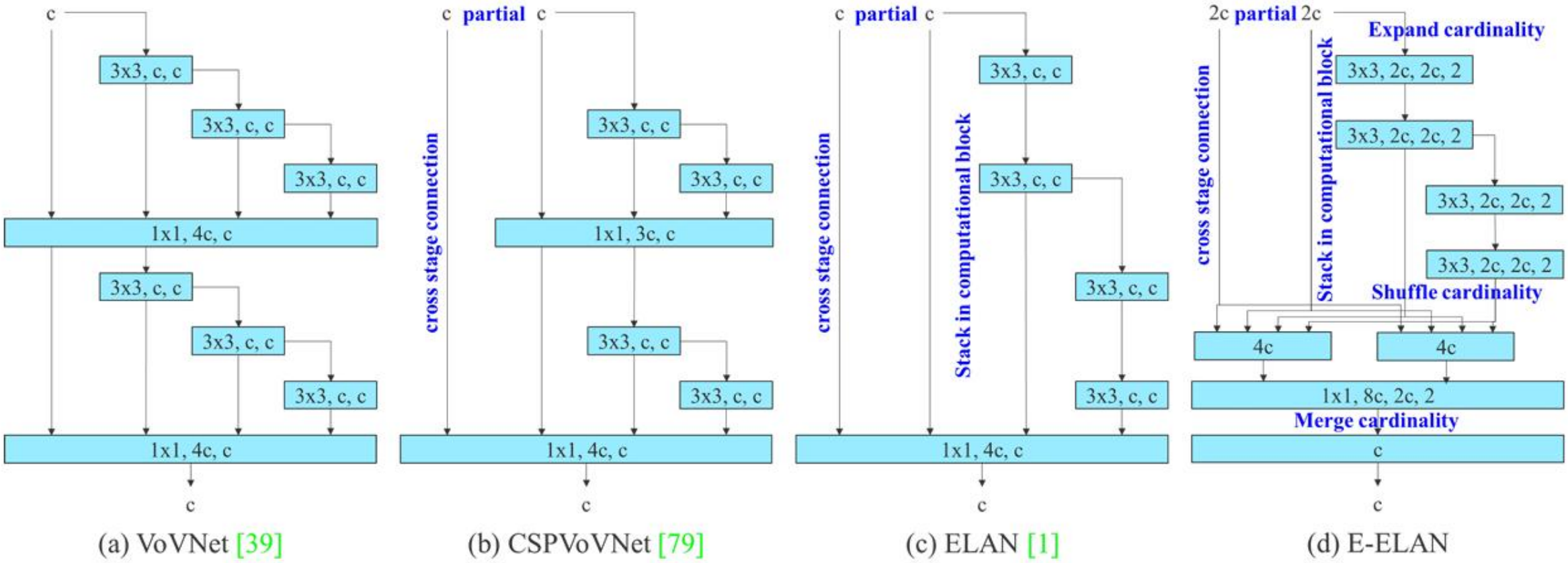
bs=32
in=out=2048
r = 56x56

Lavin A, Gray S. Fast algorithms for convolutional neural networks[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2016: 4013-4021.



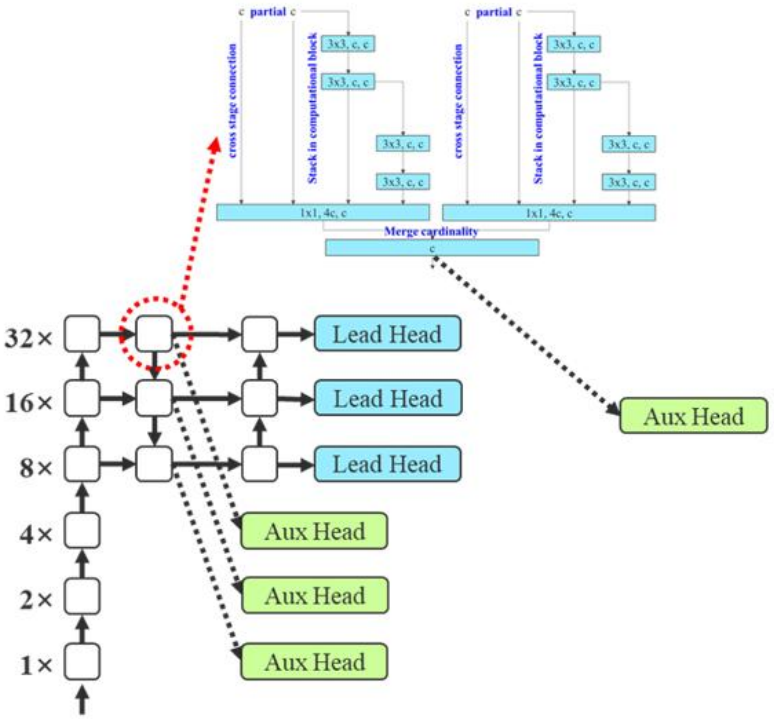
YOLOV7

E-ELAN
Aux Head
Anchor-free

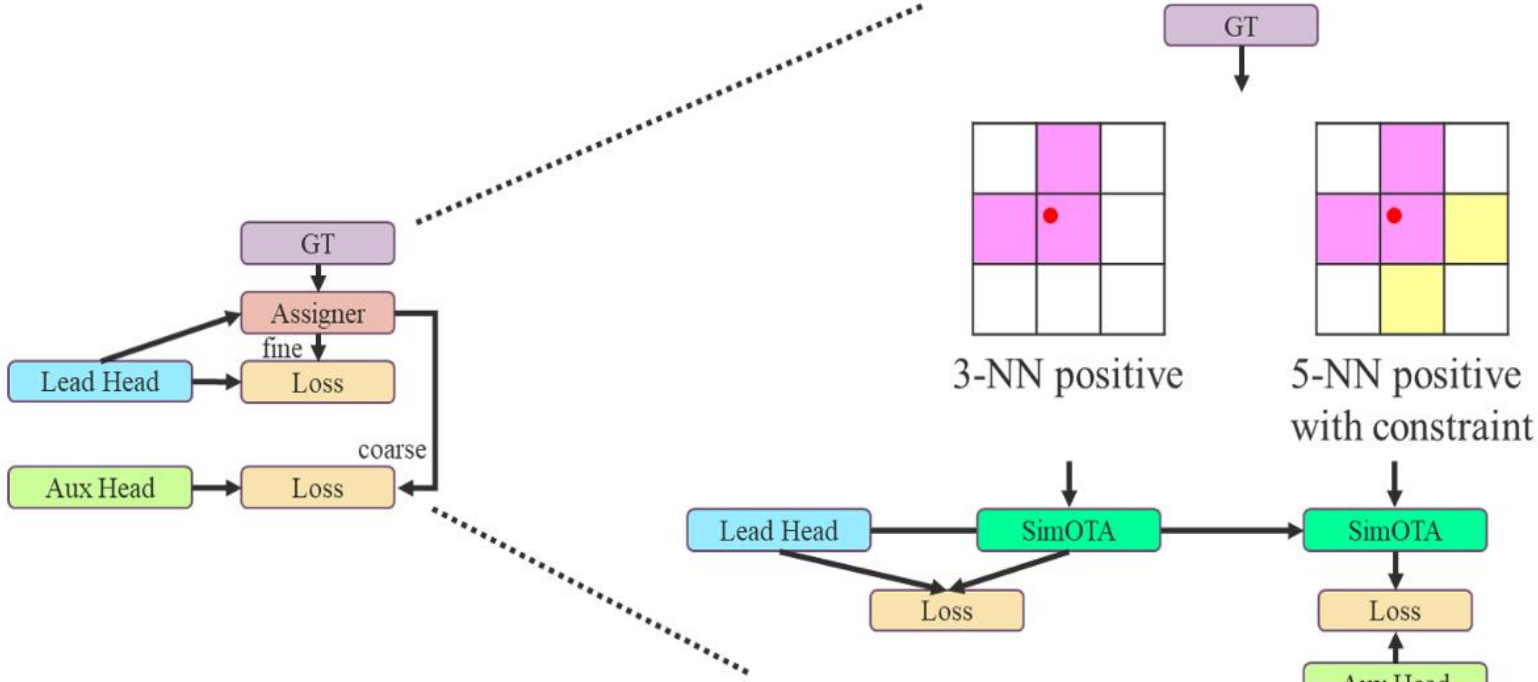


Using expand, shuffle, merge cardinality to enhance the learning ability of the network.

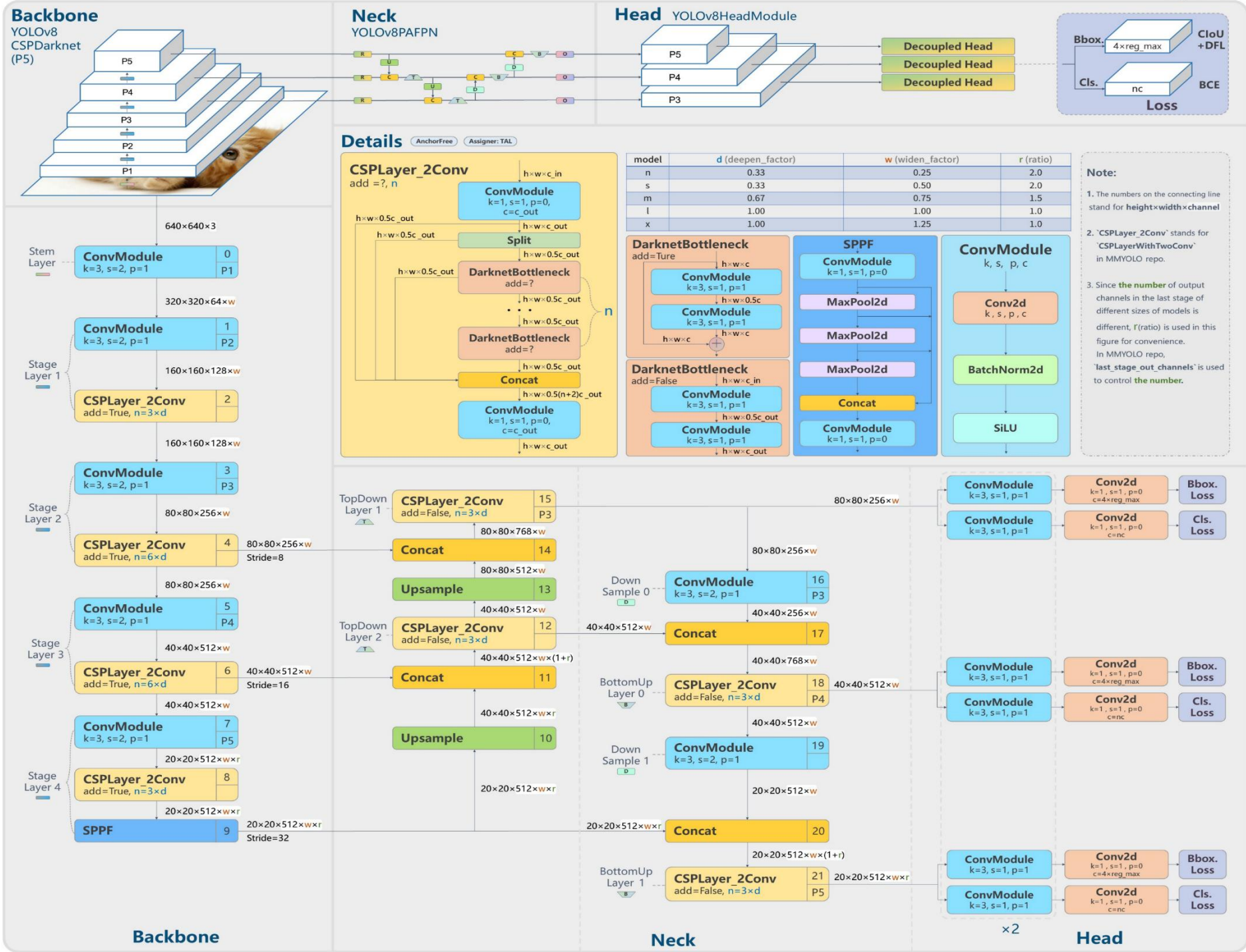
YOLOV7



(a) Model with auxiliary head



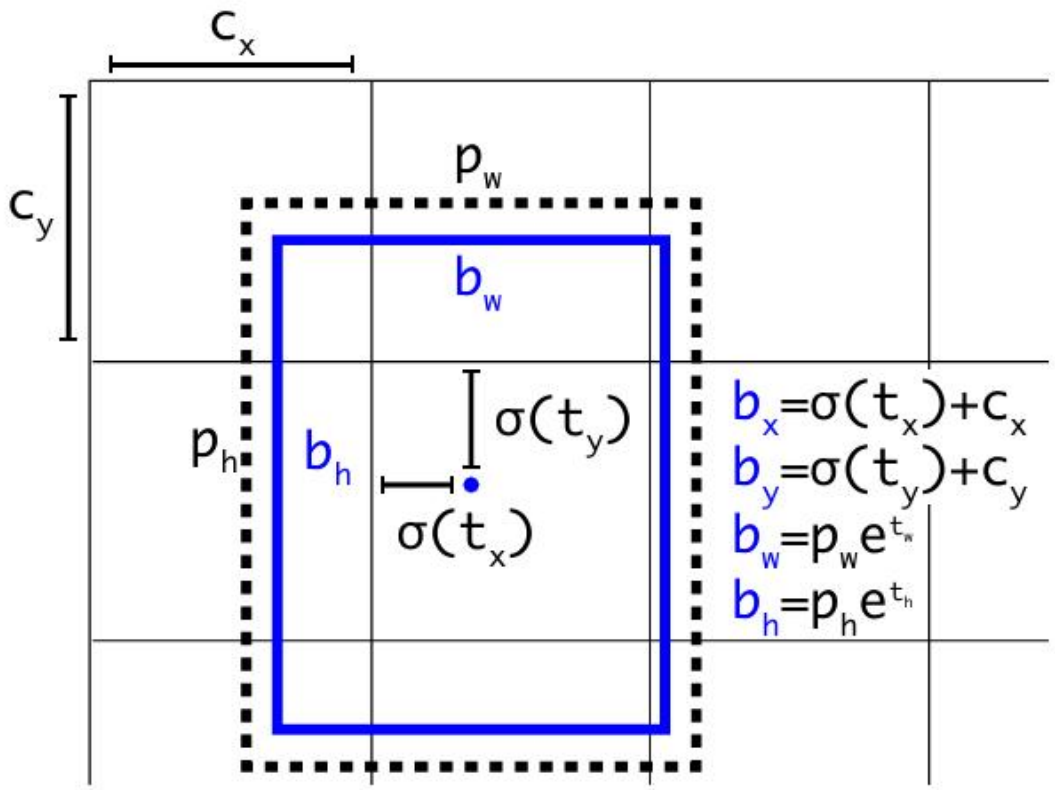
Coarse-to-fine constraint lead head guided assigner

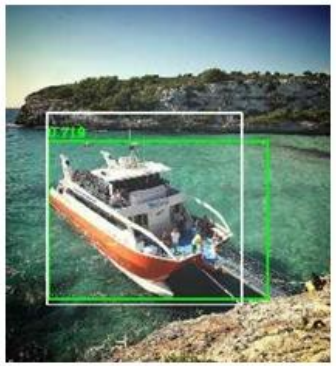


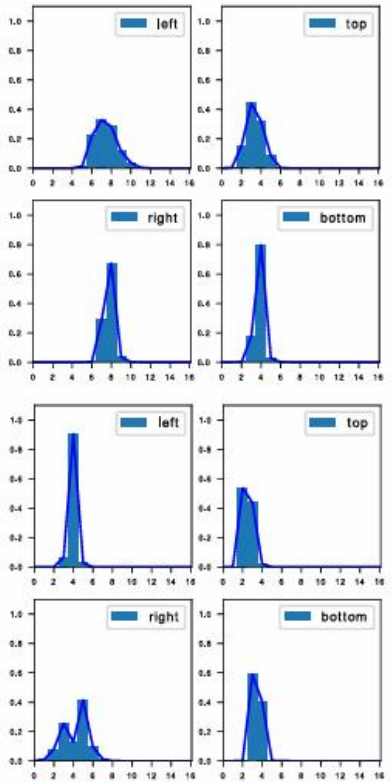
YOLOV8

DFL loss
CSPLayer2

YOLOV8

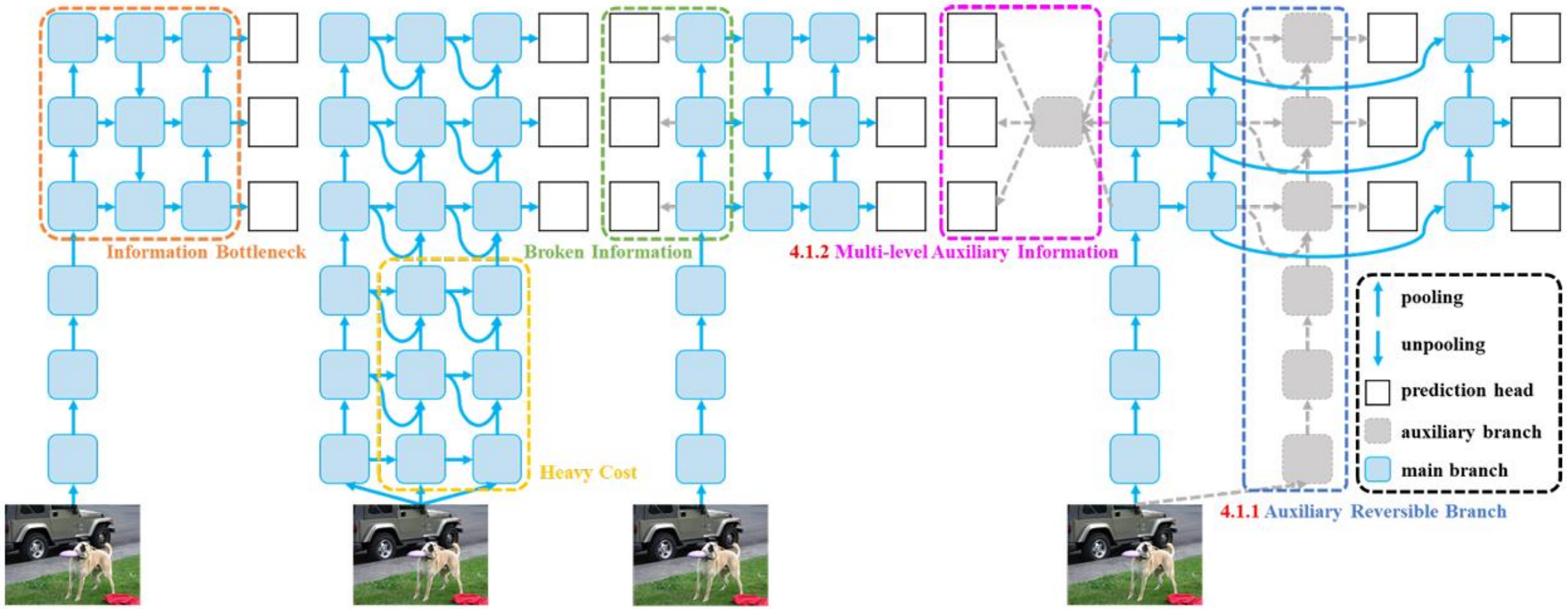




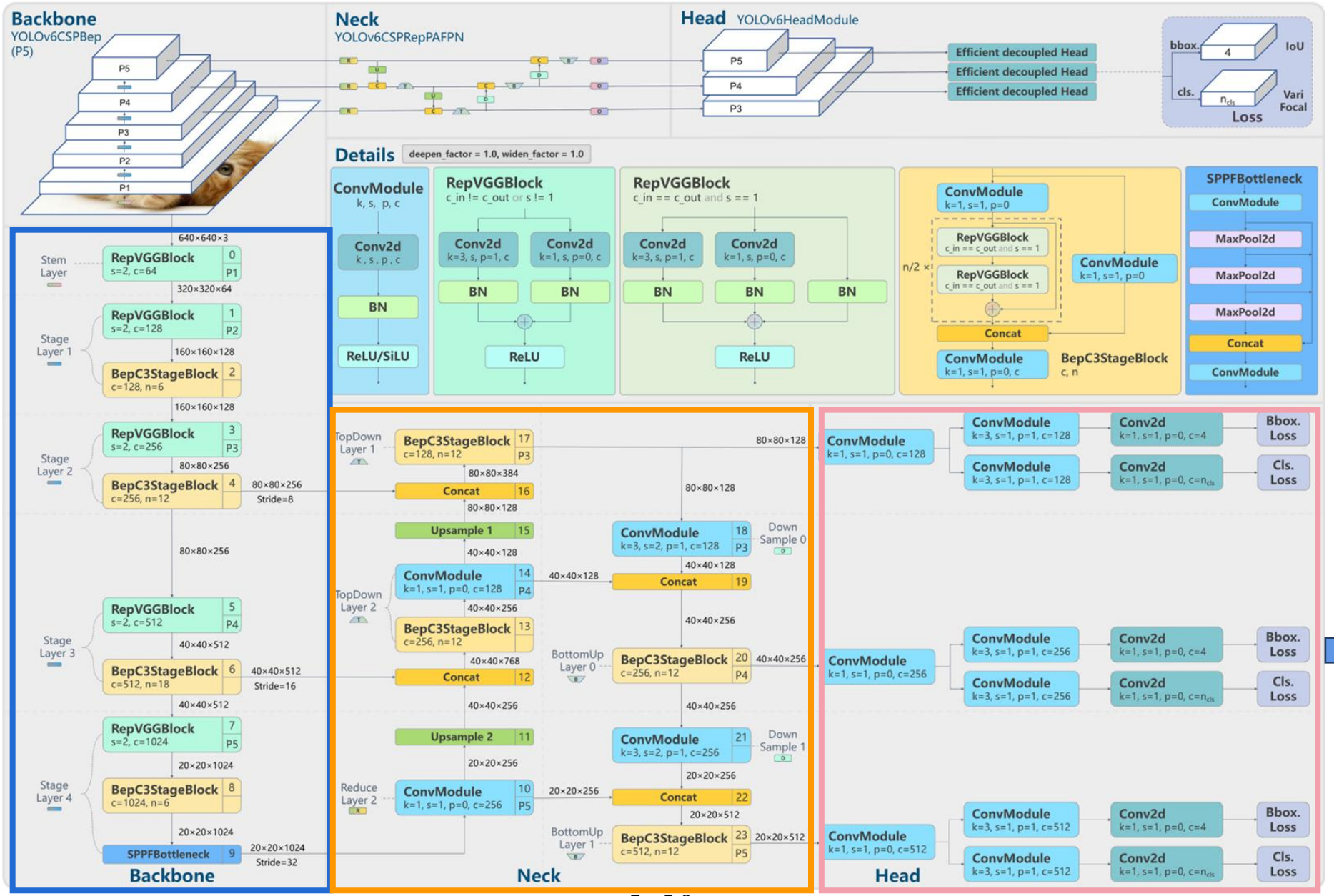


Li X, Wang W, Wu L, et al. Generalized focal loss: Learning qualified and distributed bounding boxes for dense object detection[J]. Advances in Neural Information Processing Systems, 2020, 33: 21002-21012.

YOLOV9


Information Bottleneck
Broken Information
4.1.2 Multi-level Auxiliary Information
Heavy Cost
pooling
unpooling
prediction head
auxiliary branch
main branch
4.1.1 Auxiliary Reversible Branch
(a) PAN [37]
(b) RevCol [3]
(c) Deep Supervision
(d) Programmable Gradient Information

Glance at YOLOs



5~20 ms

post-process (NMS)

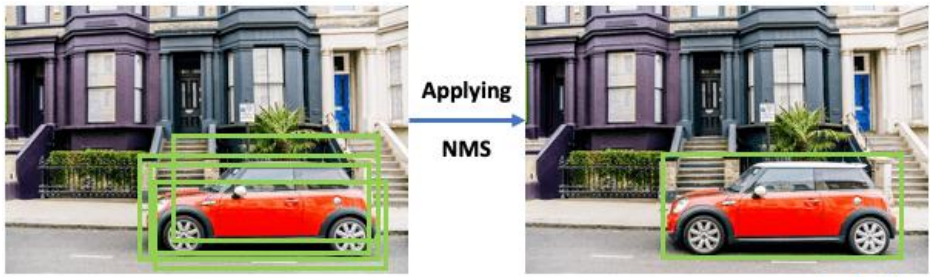
2~3 ms

github.com/open-mmlab/mmyolo

Glance at YOLOs

Model	Backbone	#Epochs	#Params (M)	GFLOPs	$FPS_{bs=1}$	AP^{val}	AP^{val}_{50}	AP^{val}_{75}	AP^{val}_{S}	AP^{val}_{M}	AP^{val}_{L}
Real-time Object Detectors											
YOLOv5-L [11]	-	300	46	109	54	49.0	67.3	-	-	-	-
YOLOv5-X [11]	-	300	86	205	43	50.7	68.9	-	-	-	-
PPYOLOE-L [40]	-	300	52	110	94	51.4	68.9	55.6	31.4	55.3	66.1
PPYOLOE-X [40]	-	300	98	206	60	52.3	69.9	56.5	33.3	56.3	66.4
YOLOv6-L [16]	-	300	59	150	99	52.8	70.3	57.7	34.4	58.1	70.1
YOLOv7-L [38]	-	300	36	104	55	51.2	69.7	55.5	35.2	55.9	66.7
YOLOv7-X [38]	-	300	71	189	45	52.9	71.1	57.4	36.9	57.7	68.6
YOLOv8-L [12]	-	-	43	165	71	52.9	69.8	57.5	35.3	58.3	69.8
YOLOv8-X [12]	-	-	68	257	50	53.9	71.0	58.7	35.7	59.3	70.7
End-to-end Object Detectors											
DINO-Deformable-DETR [44]	R50	36	47	279	5	50.9	69.0	55.3	34.6	54.1	64.6
Real-time End-to-end Object Detector (ours)											
RT-DETR	R50	72	42	136	**108**	**53.1**	**71.3**	**57.7**	34.8	58.0	70.0
RT-DETR	R101	72	76	259	**74**	**54.3**	**72.7**	58.6	36.0	58.8	**72.1**

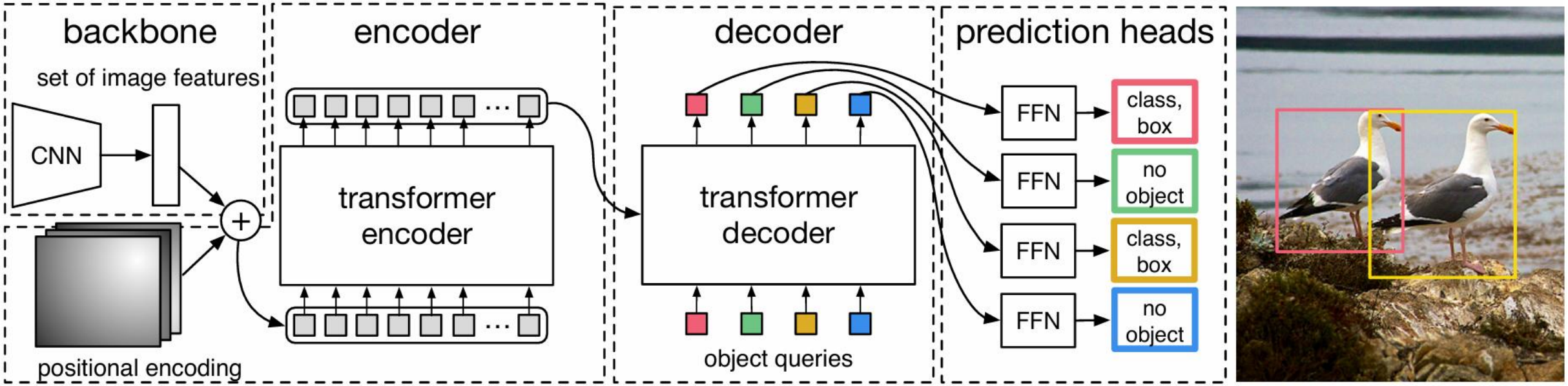
TensorRT FP16 and the input size is (800, 1333)



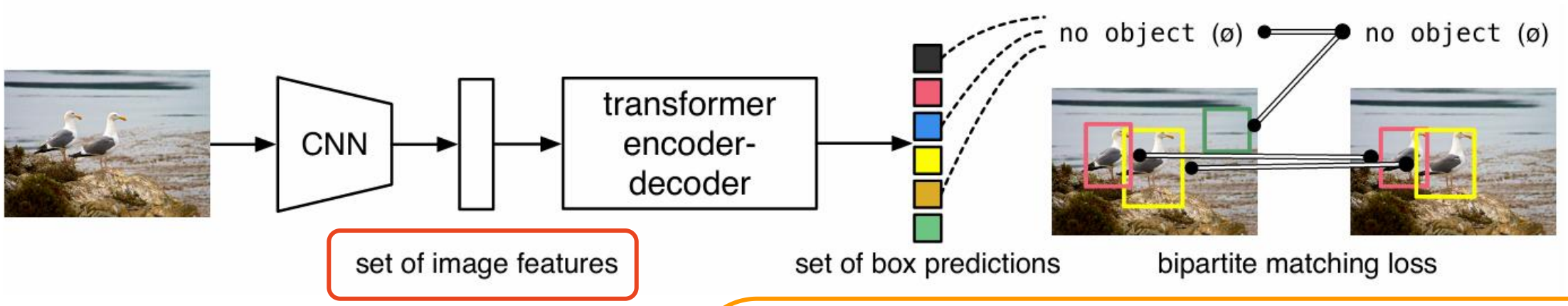
IoU thr. (Conf=0.001)	AP (%)	NMS (ms)	Conf thr. (IoU=0.7)	AP (%)	NMS (ms)
0.5	52.1	2.24	0.001	52.9	2.36
0.6	52.6	2.29	0.01	52.4	1.73
0.8	52.8	2.46	0.05	51.2	1.06

NMS execution time of YOLOv8

DETR


backbone
set of image features
CNN
positional encoding
encoder
transformer encoder
decoder
transformer decoder
object queries
prediction heads
FFN
FFN
FFN
FFN
class, box
no object
class, box
no object

DETR

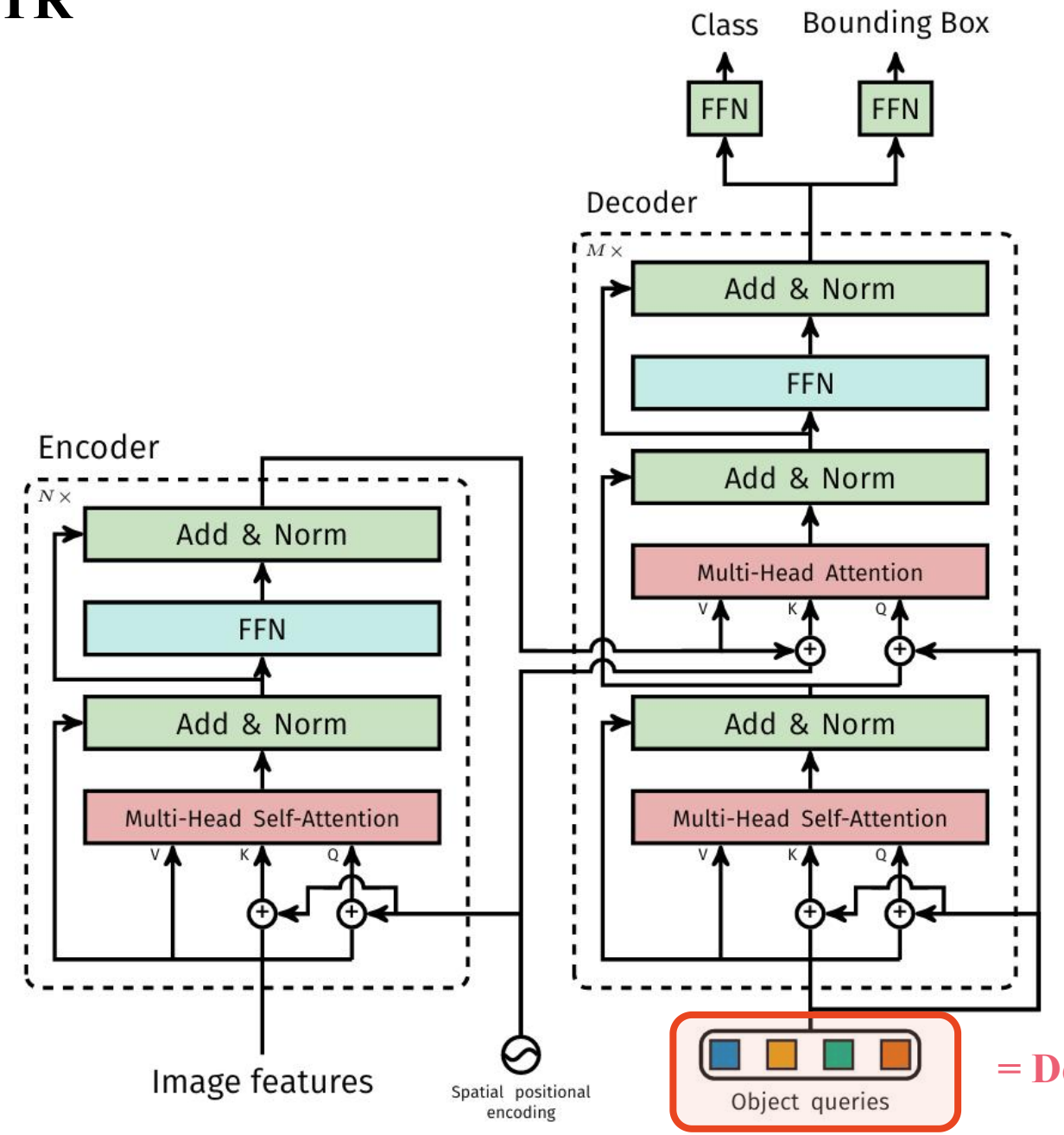


Set Prediction

$$\hat{\sigma} = \arg\min_{\sigma \in \mathfrak{S}_N} \sum_{i}^{N} \mathcal{L}_{\text{match}}(y_i, \hat{y}_{\sigma(i)})$$

$$\mathcal{L}_{\text{Hungarian}}(y, \hat{y}) = \sum_{i=1}^{N} \left[-\log \hat{p}_{\hat{\sigma}(i)}(c_i) + \mathbb{1}_{\{c_i \neq \varnothing\}} \mathcal{L}_{\text{box}}(b_i, \hat{b}_{\hat{\sigma}}(i)) \right]$$

DETR

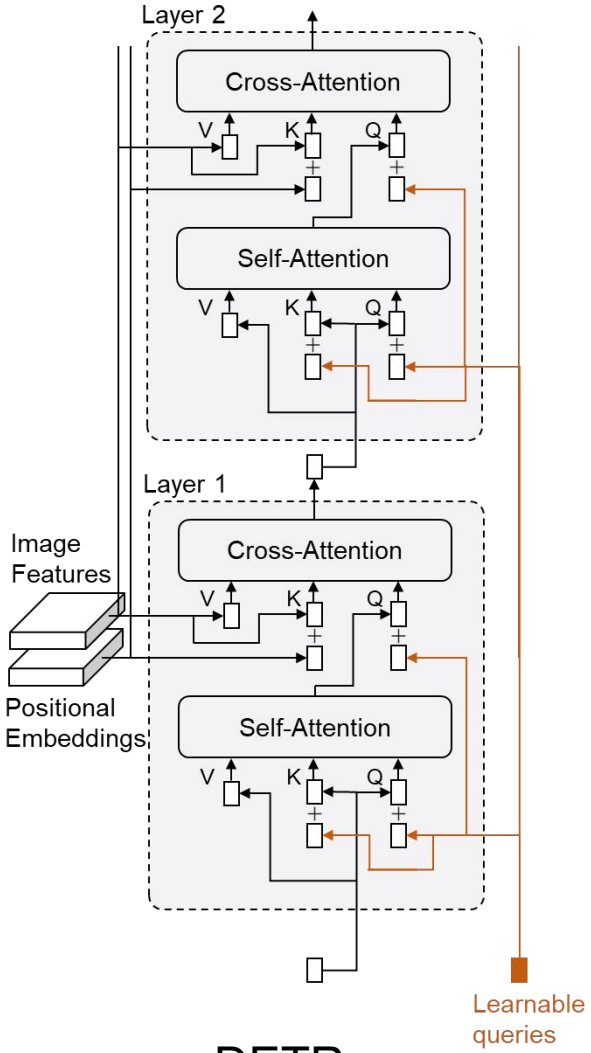


FFN

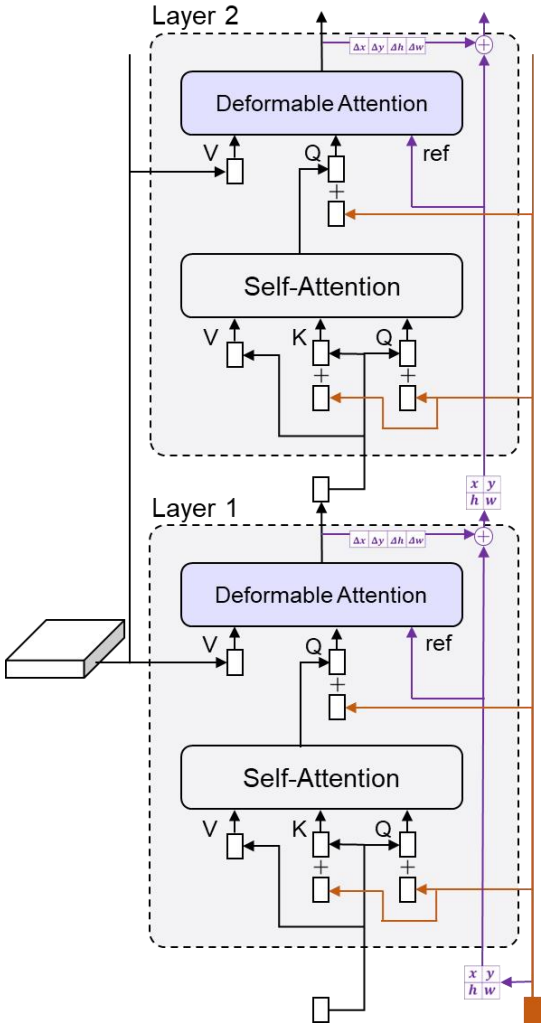
FC + Act + Dropout + FC + Dropout

= Decoder Embedding + Learnable queries

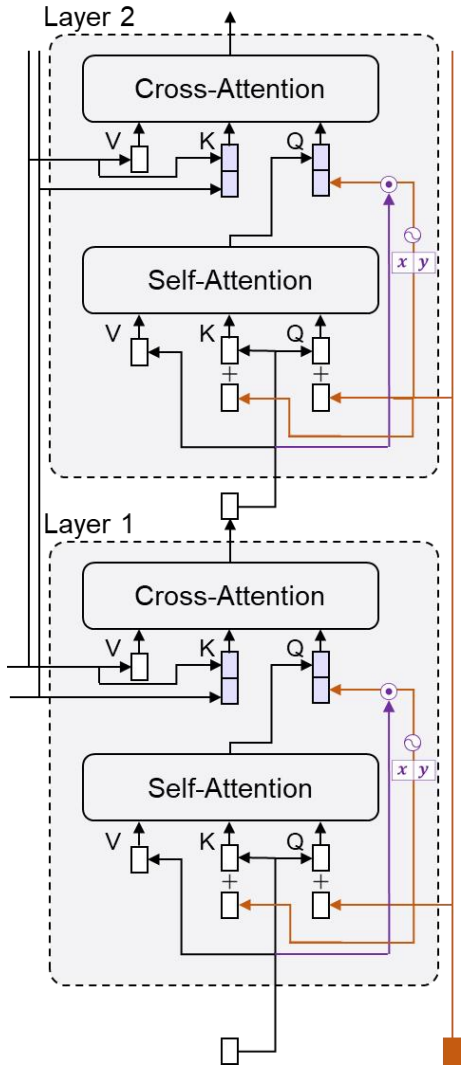
Understanding the queries



DETR

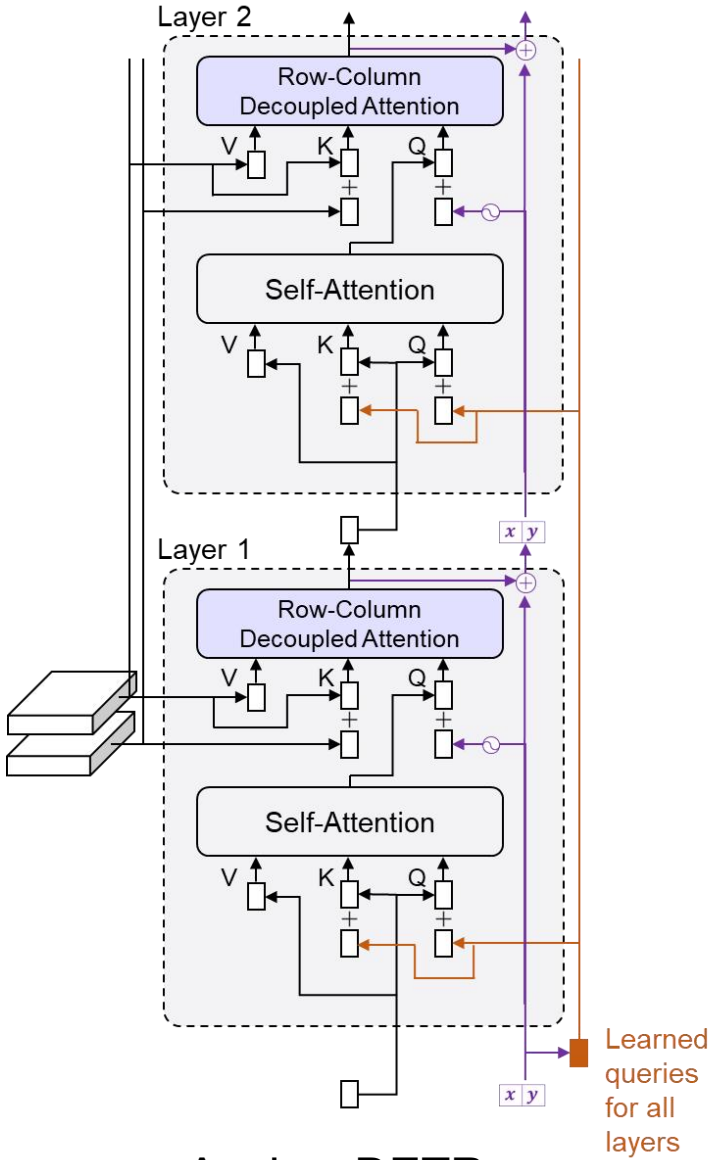


Deformable DETR

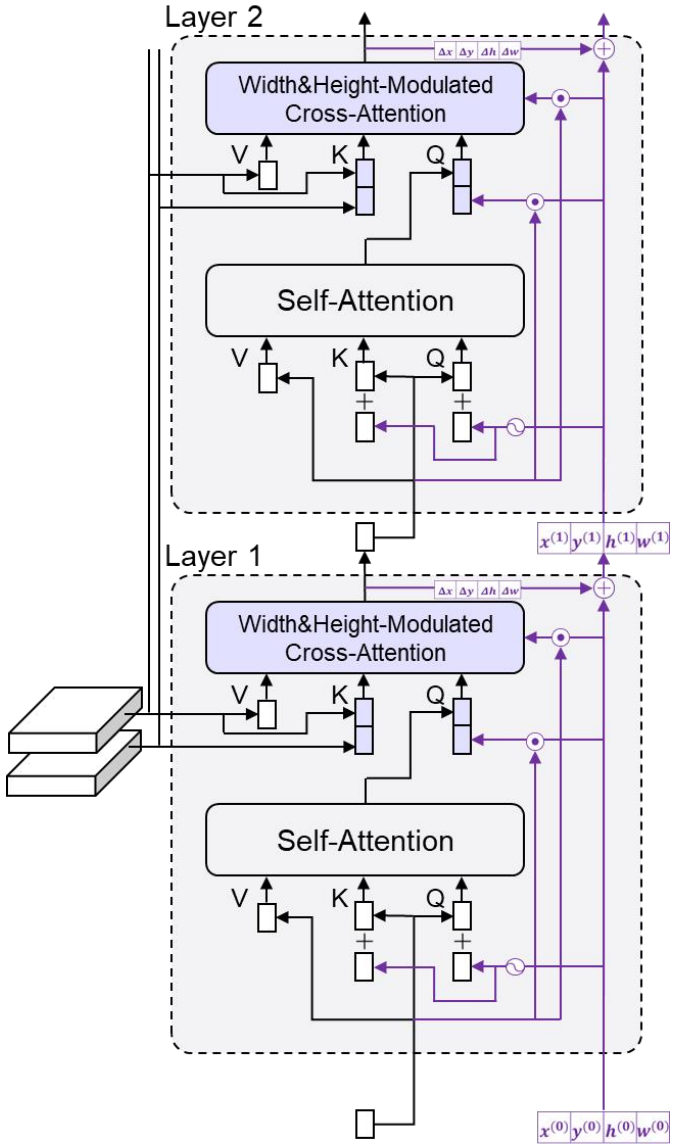


Conditional DETR

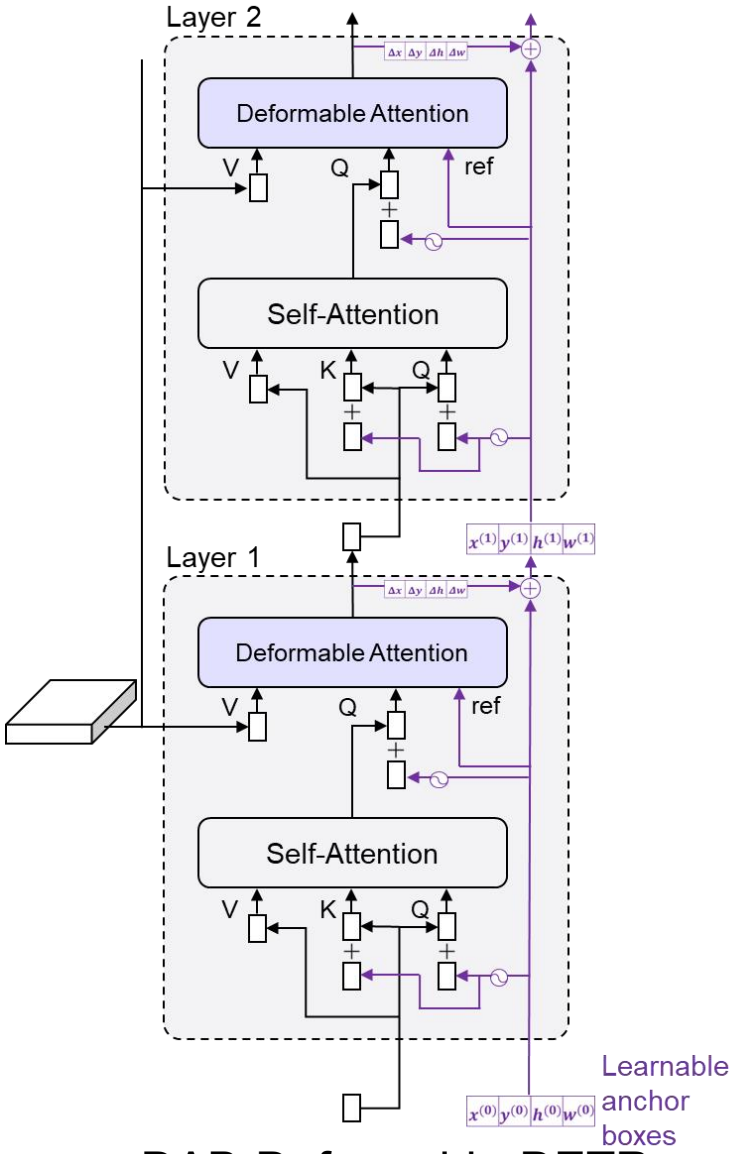
Understanding the queries



Anchor-DETR

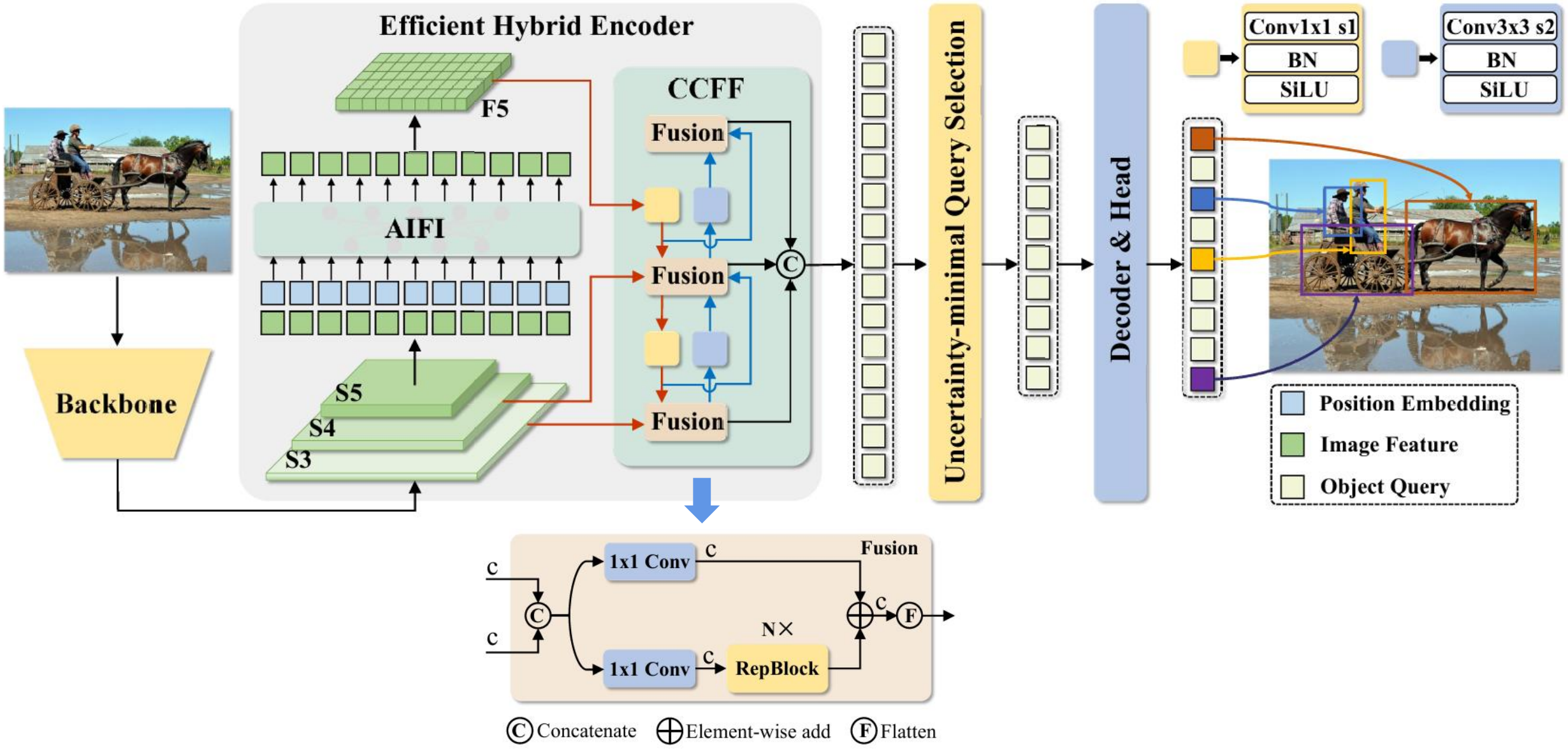


DAB-DETR



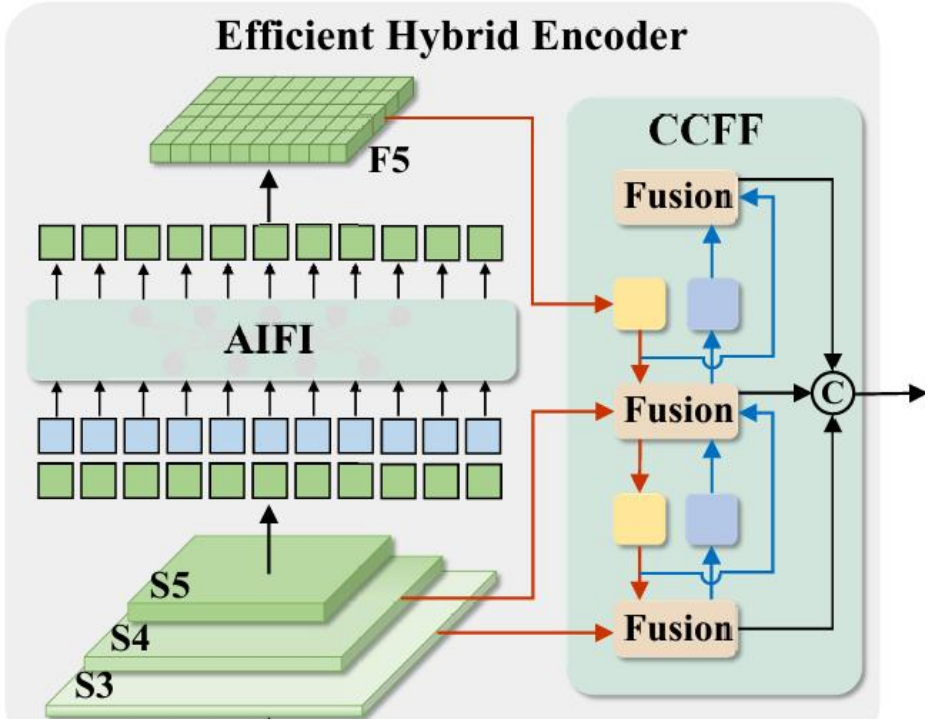
DAB-Deformable-DETR

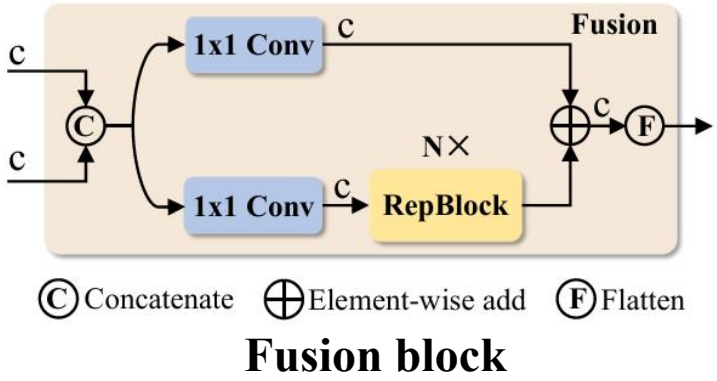
RT-DETR



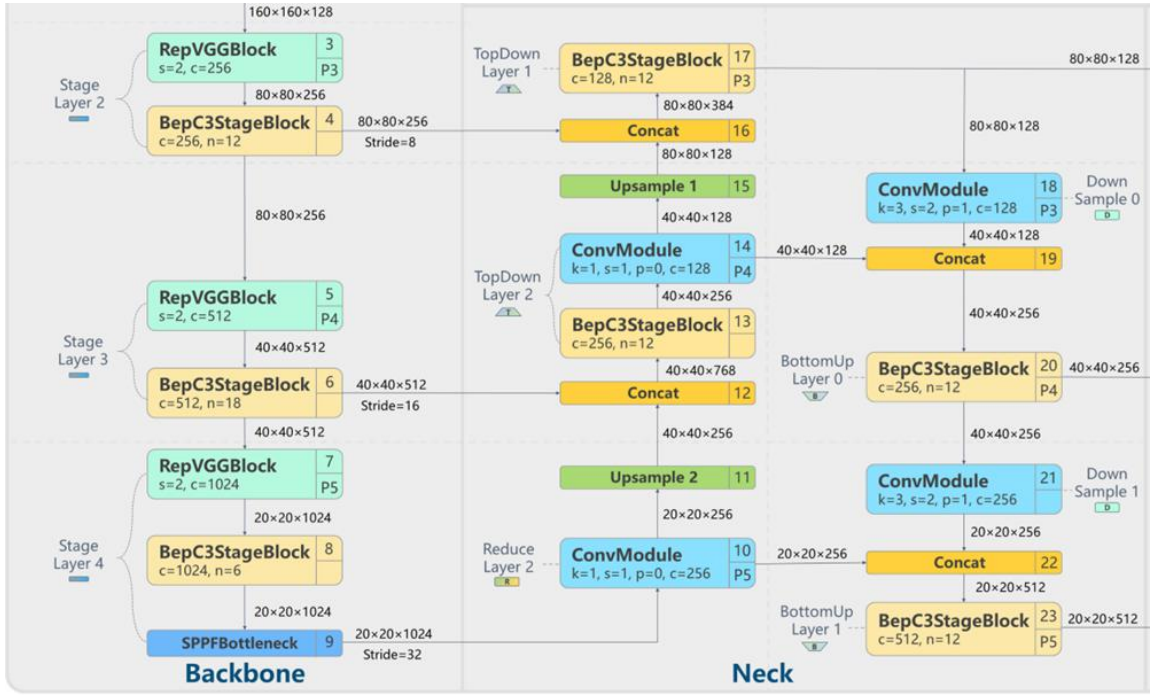
Fusion block

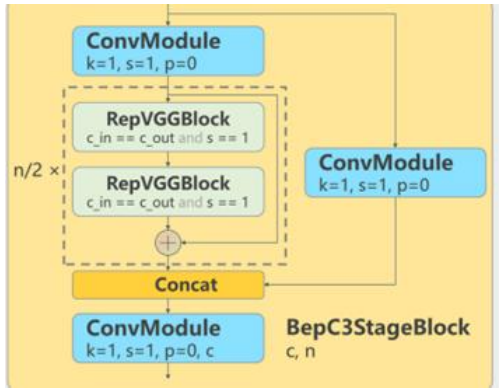
RT-DETR VS YOLOV6



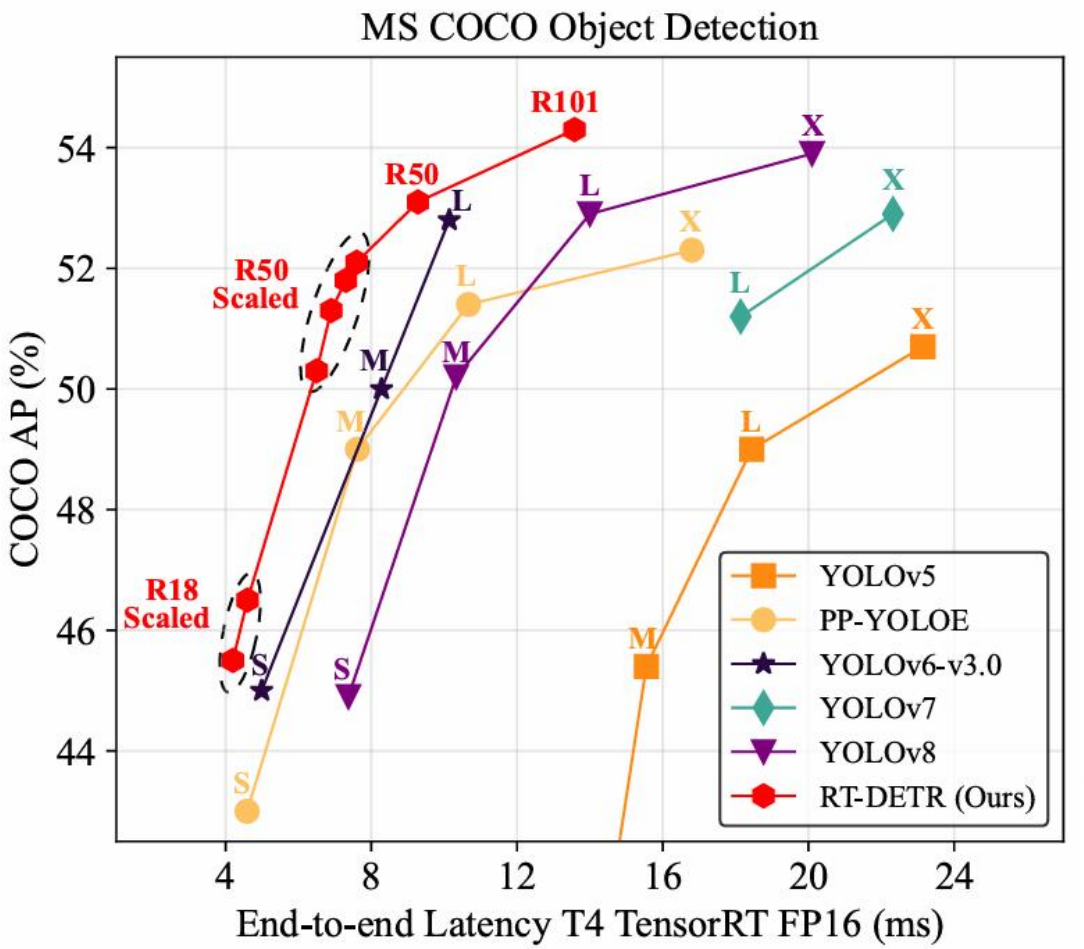


Fusion block



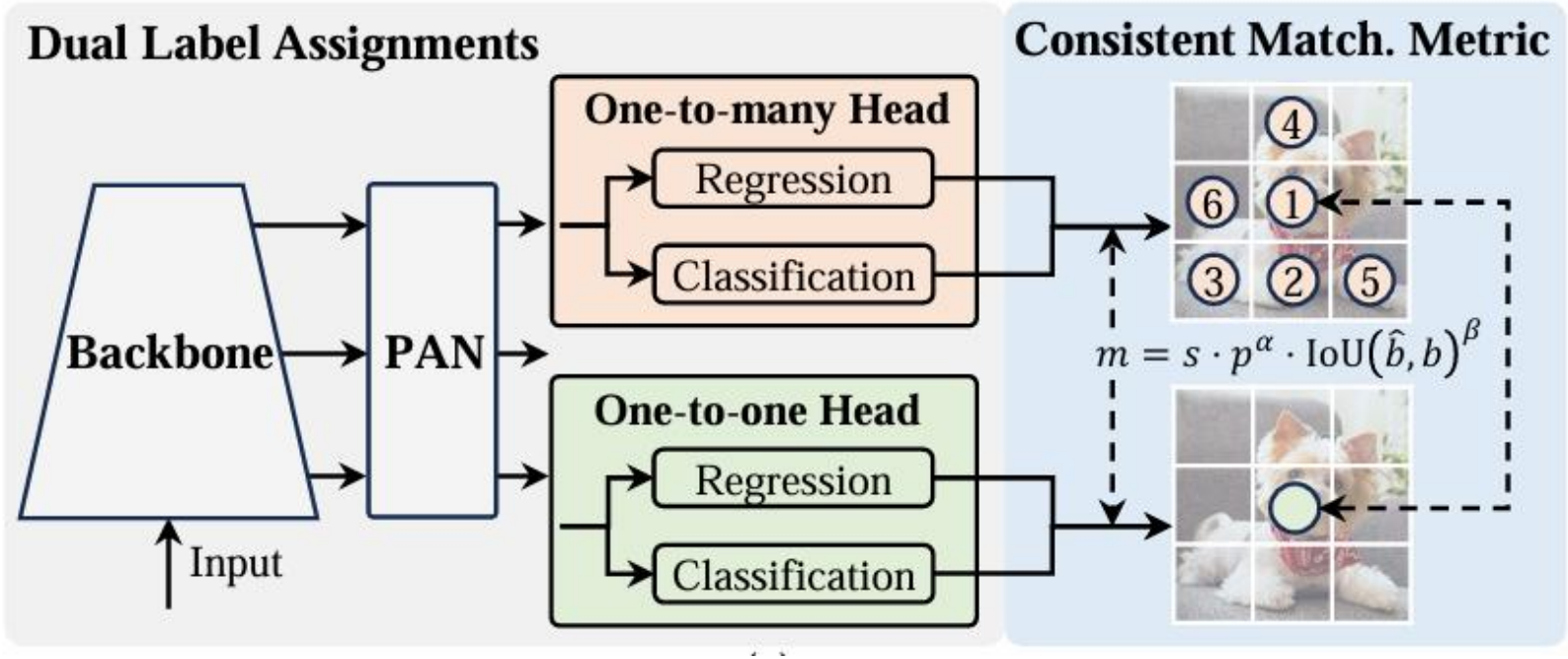


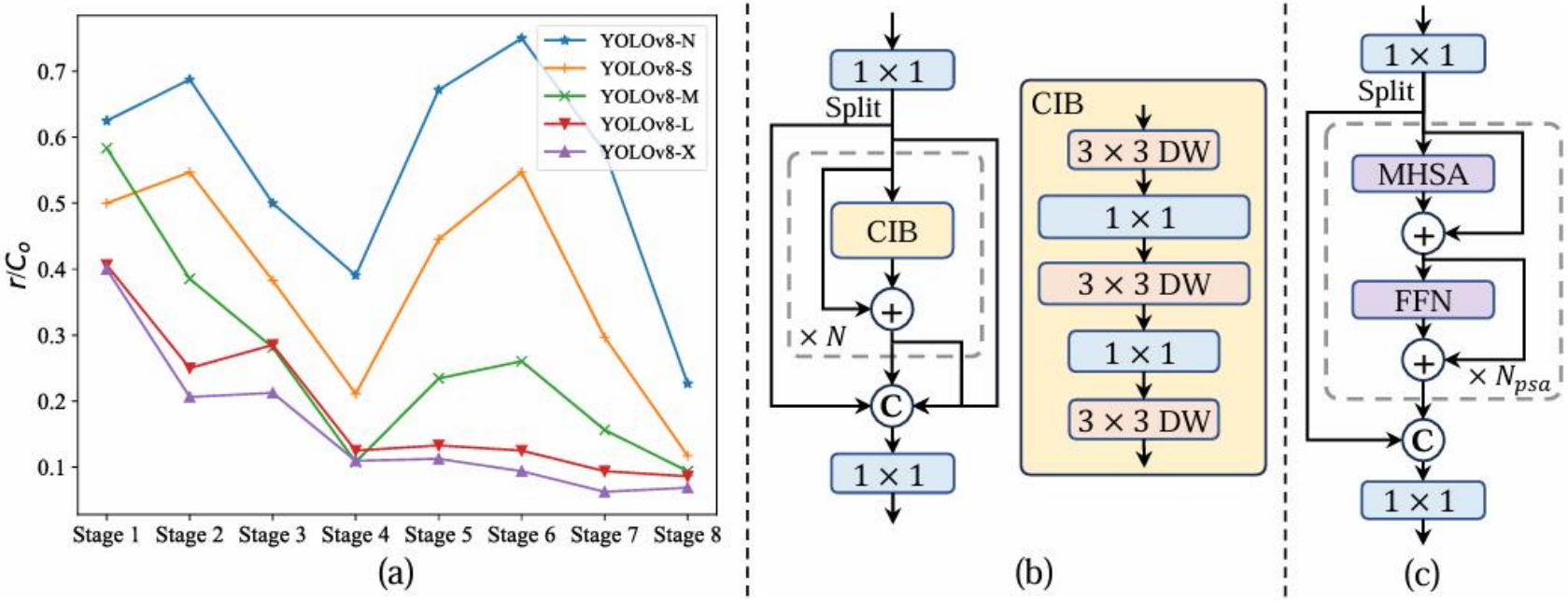
RT-DETR



Model	#Epochs	#Params (M)	GFLOPs	$FPS_{bs=1}$	AP^{val}	AP^{val}_{50}	AP^{val}_{75}	AP^{val}_{S}	AP^{val}_{M}	AP^{val}_{L}
S and M models of YOLO Detectors										
YOLOv5-S[11]	300	7.2	16.5	74	37.4	56.8	-	-	-	-
YOLOv5-M[11]	300	21.2	49.0	64	45.4	64.1	-	-	-	-
PPYOLOE-S[40]	300	7.9	17.4	218	43.0	59.6	47.1	25.9	47.4	58.6
PPYOLOE-M[40]	300	23.4	49.9	131	48.9	65.8	53.7	30.8	53.4	65.3
YOLOv6-S[16]	300	18.5	45.3	201	45.0	61.8	48.9	24.3	50.2	62.7
YOLOv6-M[16]	300	34.9	85.8	121	50.0	66.9	54.6	30.6	55.4	67.3
YOLOv8-S[12]	-	11.2	28.6	136	44.9	61.8	48.6	25.7	49.9	61.0
YOLOv8-M[12]	-	25.9	78.9	97	50.2	67.2	54.6	32.0	55.7	66.4
Scaled RT-DETRs										
Scaled RT-DETR-R50-Dec^2	72	$36^{\dagger}$	98.4	**154**	**50.3**	**68.4**	54.5	**32.2**	55.2	**67.5**
Scaled RT-DETR-R50-Dec^3	72	$36^{\dagger}$	100.1	**145**	**51.3**	**69.6**	**55.4**	**33.6**	**56.1**	**68.6**
Scaled RT-DETR-R50-Dec^4	72	$36^{\dagger}$	101.8	**137**	**51.8**	**70.0**	**55.9**	**33.7**	**56.4**	**69.4**
Scaled RT-DETR-R50-Dec^5	72	$36^{\dagger}$	103.5	**132**	**52.1**	**70.5**	**56.2**	**34.3**	**56.9**	**69.9**
Scaled RT-DETR-R50-Dec^6	72	36	105.2	125	**52.2**	**70.6**	**56.4**	**34.4**	**57.0**	**70.0**
Scaled RT-DETR-R34-Dec^2	72	$31^{\dagger}$	89.3	185	**47.4**	**64.7**	**51.3**	**28.9**	**51.0**	**64.2**
Scaled RT-DETR-R34-Dec^3	72	$31^{\dagger}$	91.0	172	**48.5**	**66.2**	**52.3**	**30.2**	**51.9**	**66.2**
Scaled RT-DETR-R34-Dec^4	72	31	92.7	161	**48.9**	**66.8**	**52.9**	**30.6**	**52.4**	**66.3**
Scaled RT-DETR-R18-Dec^2	72	$20^{\dagger}$	59.0	**238**	**45.5**	**62.5**	**49.4**	**27.8**	48.7	61.7
Scaled RT-DETR-R18-Dec^3	72	20	60.7	217	**46.5**	**63.8**	**50.4**	**28.4**	49.8	**63.0**

YOLOV10

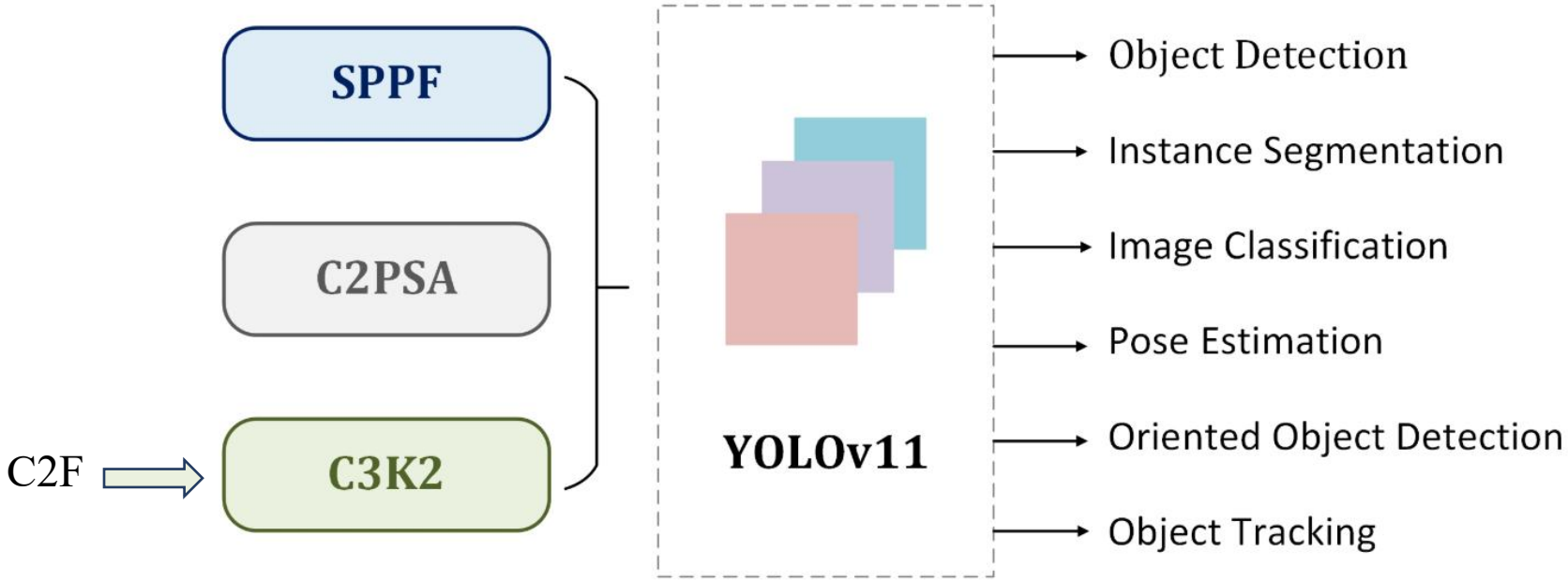

Dual Label Assignments
One-to-many Head
Regression
Classification
Backbone
PAN
Input
One-to-one Head
Regression
Classification
Consistent Match. Metric
$m = s \cdot p^{\alpha} \cdot \mathrm{IoU}(\hat{b}, b)^{\beta}$

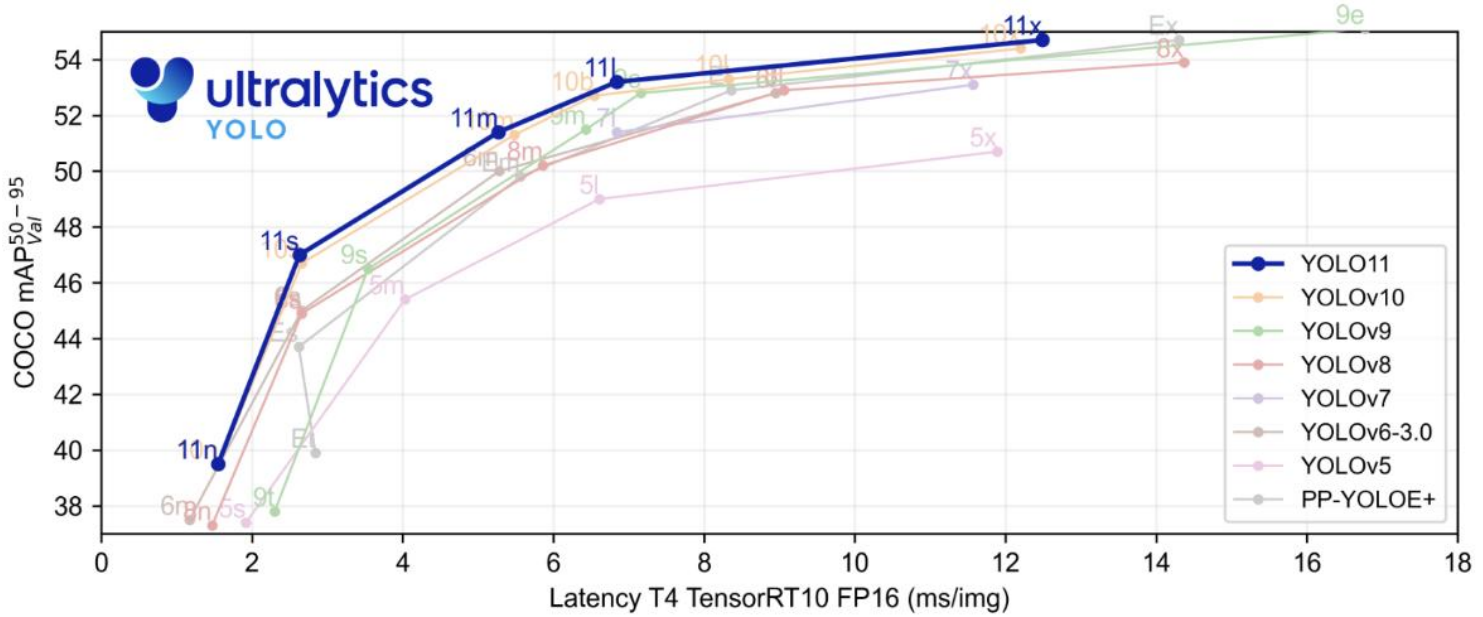

YOLOv8-N
YOLOv8-S
YOLOv8-M
YOLOv8-L
YOLOv8-X
r/C_o
Stage 1 Stage 2 Stage 3 Stage 4 Stage 5 Stage 6 Stage 7 Stage 8
(a)
1 × 1
Split
CIB
× N
C
1 × 1
(b)
CIB
3 × 3 DW
1 × 1
3 × 3 DW
1 × 1
3 × 3 DW
1 × 1
Split
MHSA
FFN
× N_{psa}
C
1 × 1
(c)

YOLOV10

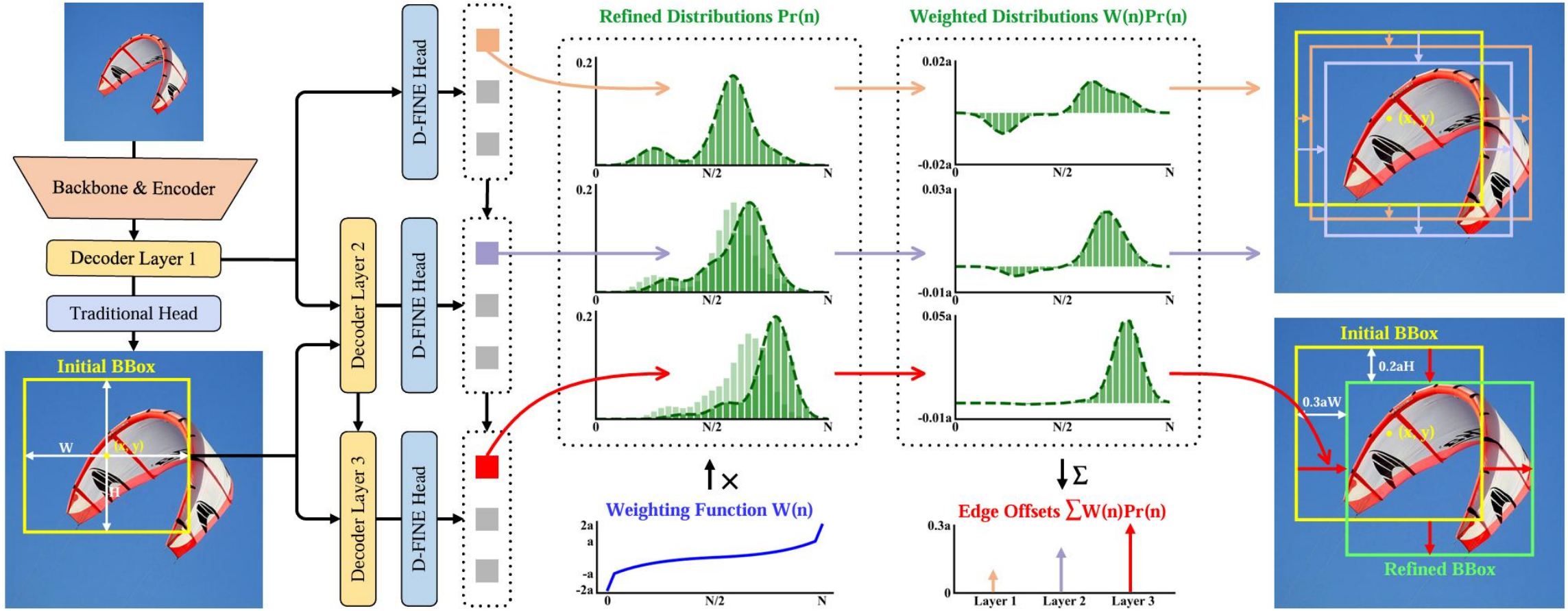
Model	#Param.(M)	FLOPs(G)	AP^{val}(%)	Latency(ms)	$Latency^{f}$(ms)
YOLOv6-3.0-N [27]	4.7	11.4	37.0	2.69	1.76
Gold-YOLO-N [54]	5.6	12.1	39.6	2.92	1.82
YOLOv8-N [20]	3.2	8.7	37.3	6.16	1.77
YOLOv10-N (Ours)	**2.3**	**6.7**	**38.5 / 39.5**†	**1.84**	**1.79**
YOLOv6-3.0-S [27]	18.5	45.3	44.3	3.42	2.35
Gold-YOLO-S [54]	21.5	46.0	45.4	3.82	2.73
YOLO-MS-XS [7]	4.5	17.4	43.4	8.23	2.80
YOLO-MS-S [7]	8.1	31.2	46.2	10.12	4.83
YOLOv8-S [20]	11.2	28.6	44.9	7.07	2.33
YOLOv9-S [59]	7.1	26.4	46.7	-	-
RT-DETR-R18 [71]	20.0	60.0	46.5	4.58	4.49
YOLOv10-S (Ours)	**7.2**	**21.6**	**46.3 / 46.8**†	**2.49**	**2.39**
YOLOv6-3.0-M [27]	34.9	85.8	49.1	5.63	4.56
Gold-YOLO-M [54]	41.3	87.5	49.8	6.38	5.45
YOLO-MS [7]	22.2	80.2	51.0	12.41	7.30
YOLOv8-M [20]	25.9	78.9	50.6	9.50	5.09
YOLOv9-M [59]	20.0	76.3	51.1	-	-
RT-DETR-R34 [71]	31.0	92.0	48.9	6.32	6.21
RT-DETR-R50m [71]	36.0	100.0	51.3	6.90	6.84
YOLOv10-M (Ours)	**15.4**	**59.1**	**51.1 / 51.3**†	**4.74**	**4.63**
YOLOv6-3.0-L [27]	59.6	150.7	51.8	9.02	7.90
Gold-YOLO-L [54]	75.1	151.7	51.8	10.65	9.78
YOLOv9-C [59]	25.3	102.1	52.5	10.57	6.13
YOLOv10-B (Ours)	**19.1**	**92.0**	**52.5 / 52.7**†	**5.74**	**5.67**
YOLOv8-L [20]	43.7	165.2	52.9	12.39	8.06
RT-DETR-R50 [71]	42.0	136.0	53.1	9.20	9.07
YOLOv10-L (Ours)	**24.4**	**120.3**	**53.2 / 53.4**†	**7.28**	**7.21**
YOLOv8-X [20]	68.2	257.8	53.9	16.86	12.83
RT-DETR-R101 [71]	76.0	259.0	54.3	13.71	13.58
YOLOv10-X (Ours)	**29.5**	**160.4**	**54.4 / 54.4**†	**10.70**	**10.60**

YOLO11

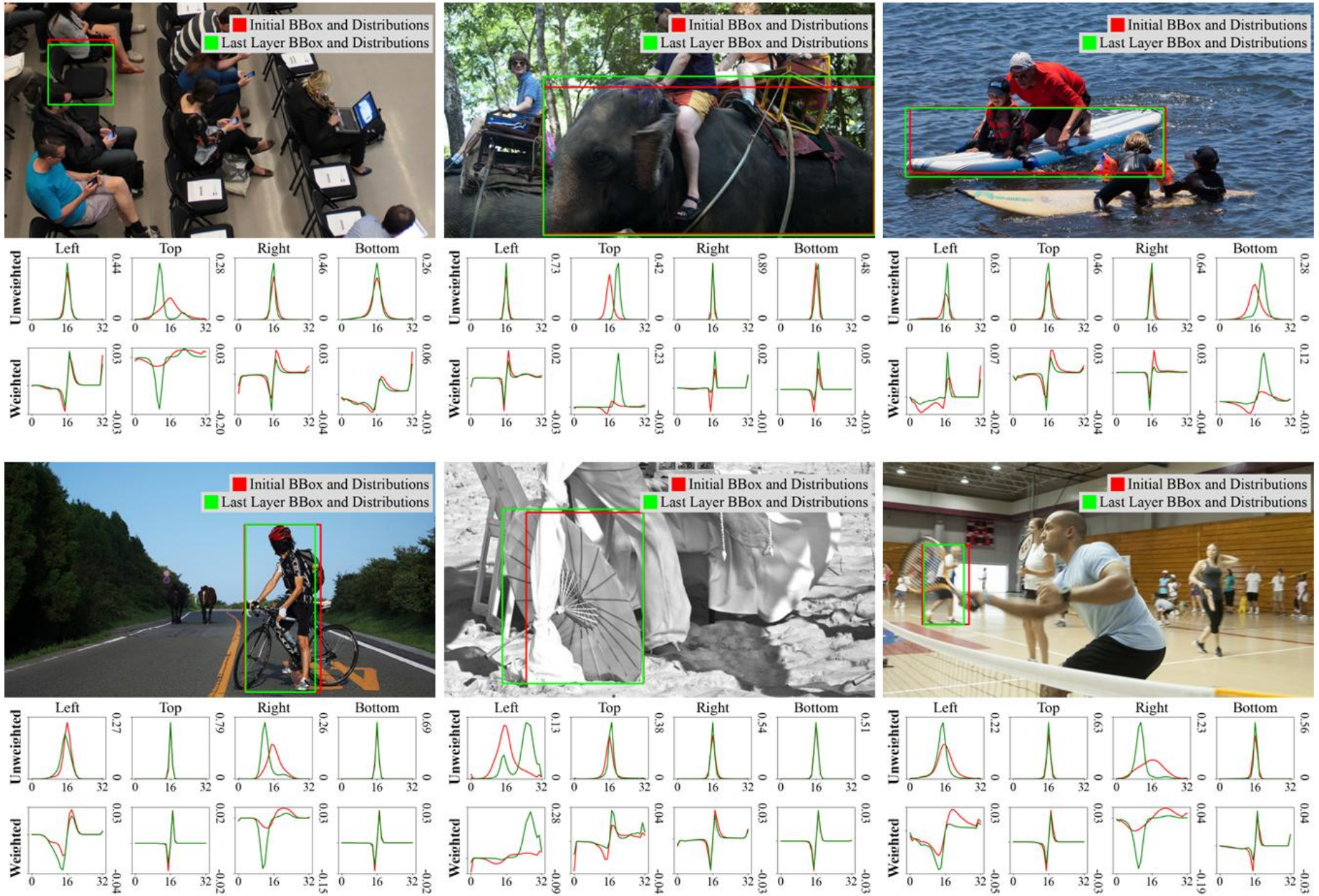

SPPF
C2PSA
C2F
C3K2
YOLOv11
Object Detection
Instance Segmentation
Image Classification
Pose Estimation
Oriented Object Detection
Object Tracking


ultralytics
YOLO
COCO mAP$^{50-95}_{val}$
Latency T4 TensorRT10 FP16 (ms/img)
YOLO11
YOLOv10
YOLOv9
YOLOv8
YOLOv7
YOLOv6-3.0
YOLOv5
PP-YOLOE+

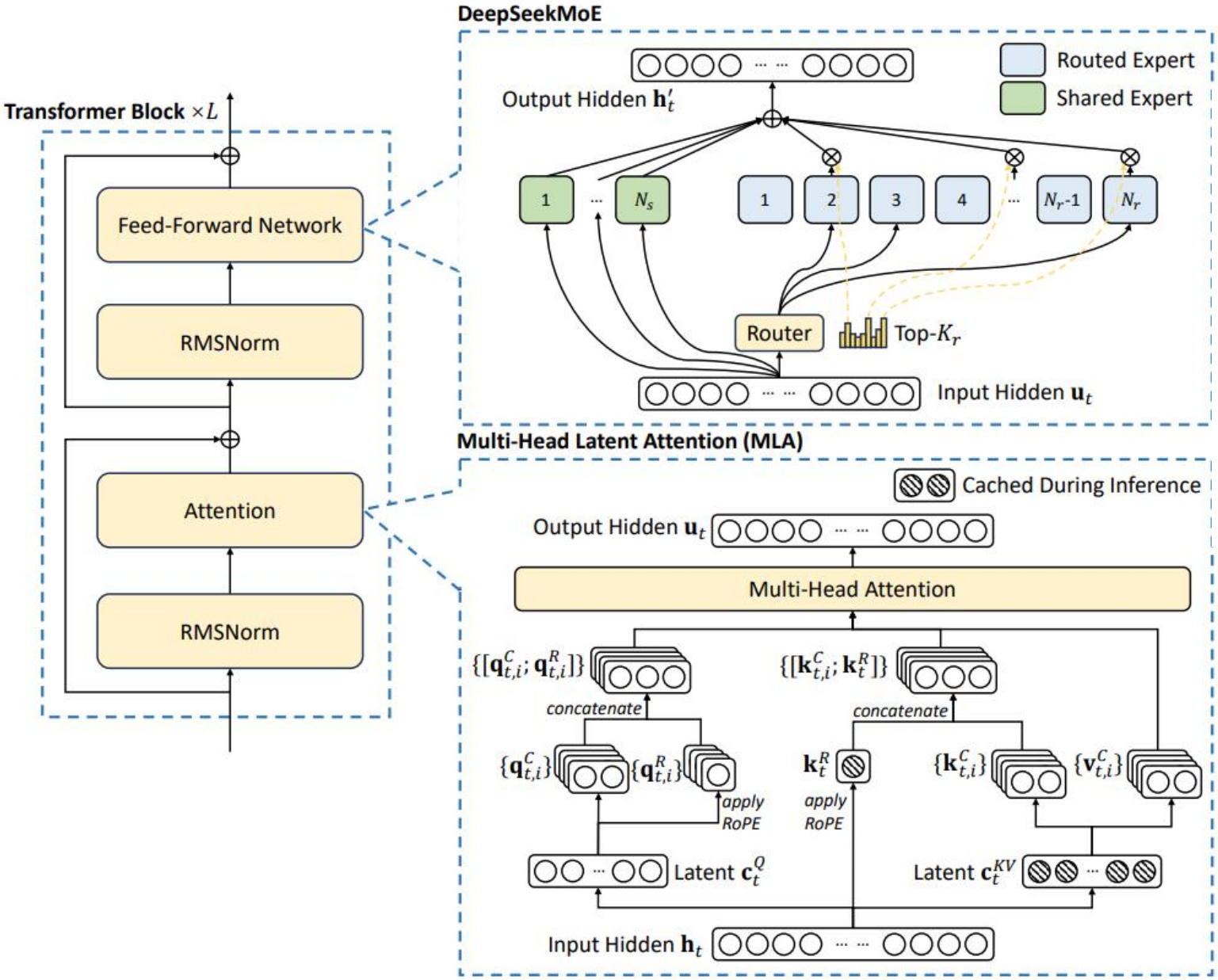
D-FINE（ICLR2025)

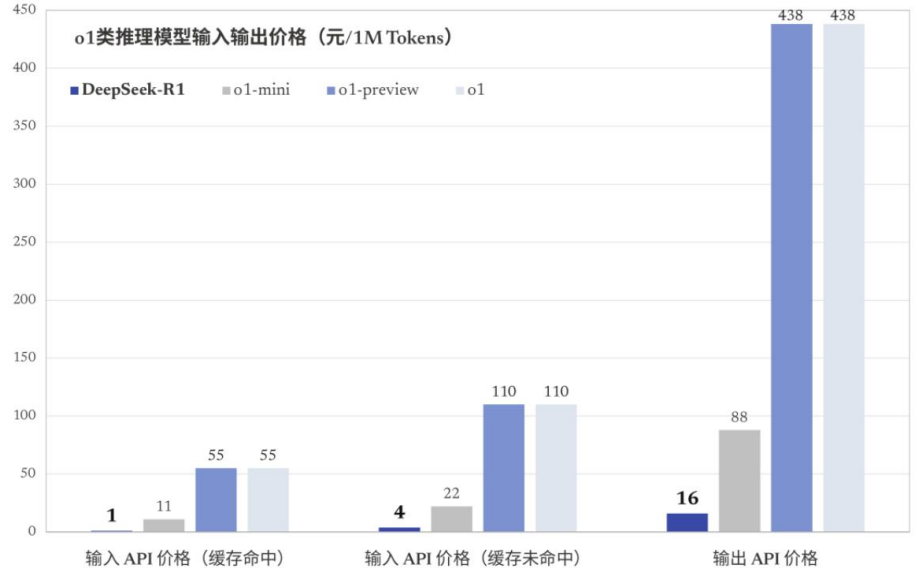


D-FINE（ICLR2025）



DeepSeek Architecture



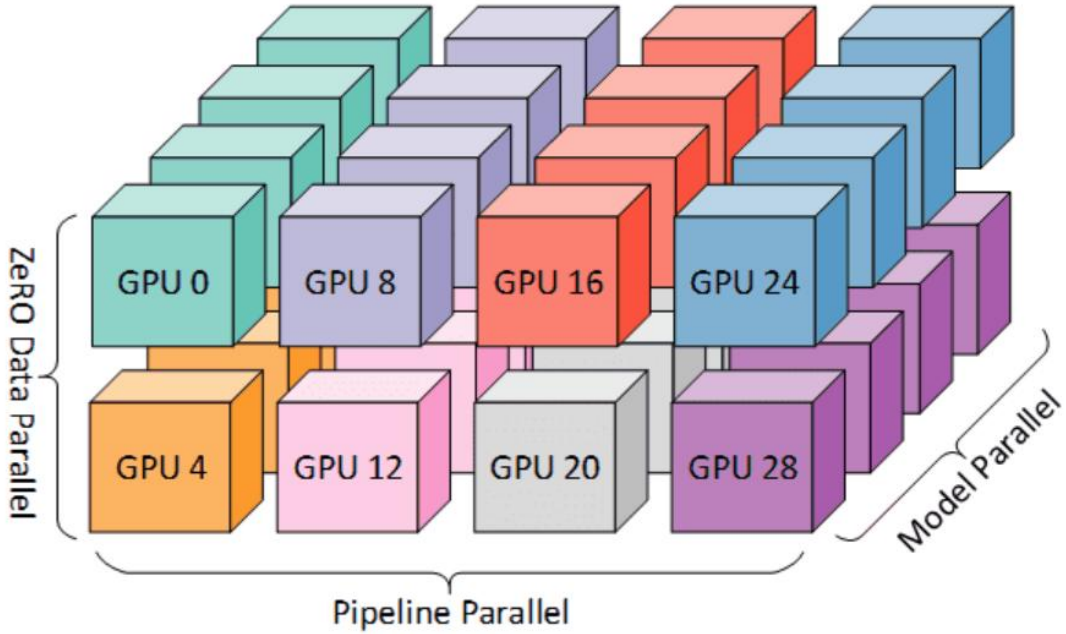


MLA (Multi-Head Latent Attention)
+
MoE (Mixture of experts)

DeepSeek Architecture

The training of DeepSeek-V3 is supported by the HAI-LLM framework, an efficient and lightweight training framework crafted by our engineers from the ground up. On the whole, DeepSeek-V3 applies 16-way Pipeline Parallelism (PP) (Qi et al., 2023a), 64-way Expert Parallelism (EP) (Lepikhin et al., 2021) spanning 8 nodes, and ZeRO-1 Data Parallelism (DP) (Rajbhandari et al., 2020).

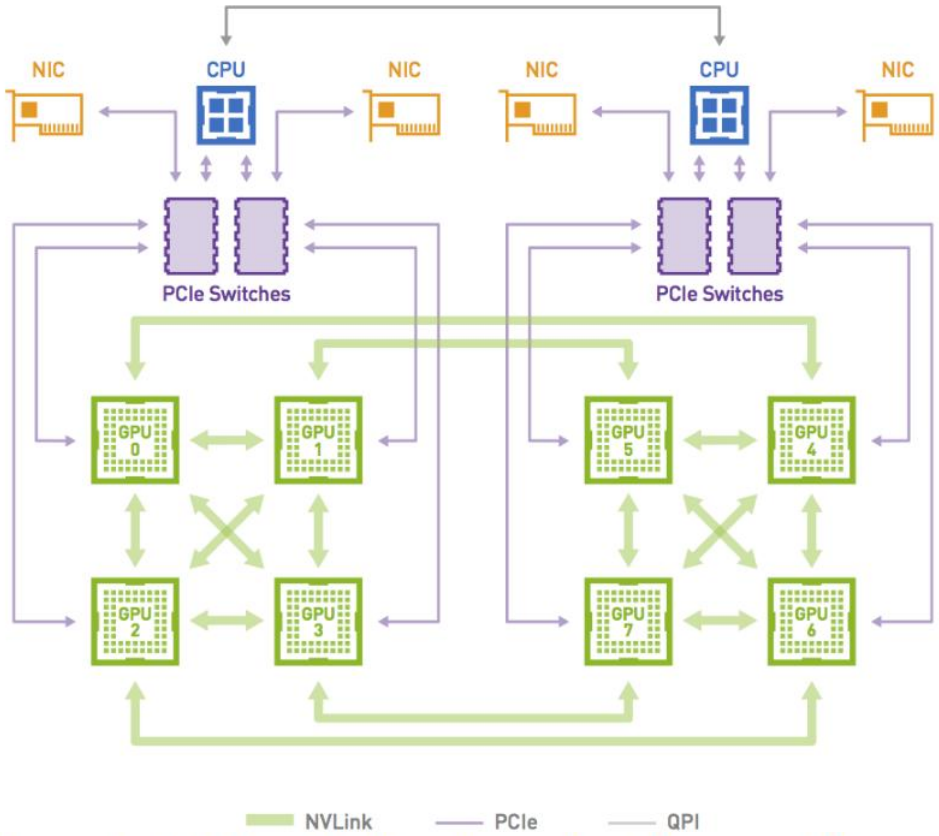
DeepSeek-V3 Technical Repor: https://arxiv.org/pdf/2412.19437



3D Parallel

HAI-LLM(2023): https://www.high-flyer.cn/en/blog/hai-llm/

DeepSeek Architecture (EP)



IB -> NVLink

Each token:
4 nodes
3.2 experts for each node
Maximum of 13 experts

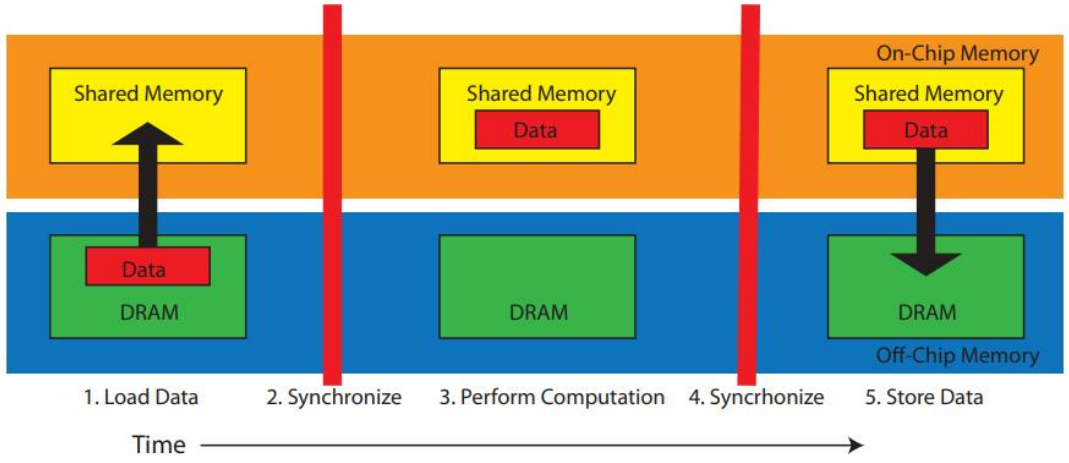
designed with the MoE gating algorithm and the network topology of our cluster. To be specific, in our cluster, cross-node GPUs are fully interconnected with IB, and intra-node communications are handled via NVLink. NVLink offers a bandwidth of 160 GB/s, roughly 3.2 times that of IB (50 GB/s). To effectively leverage the different bandwidths of IB and NVLink, we limit each token to be dispatched to at most 4 nodes, thereby reducing IB traffic. For each token, when its routing decision is made, it will first be transmitted via IB to the GPUs with the same in-node index on its target nodes. Once it reaches the target nodes, we will endeavor to ensure that it is

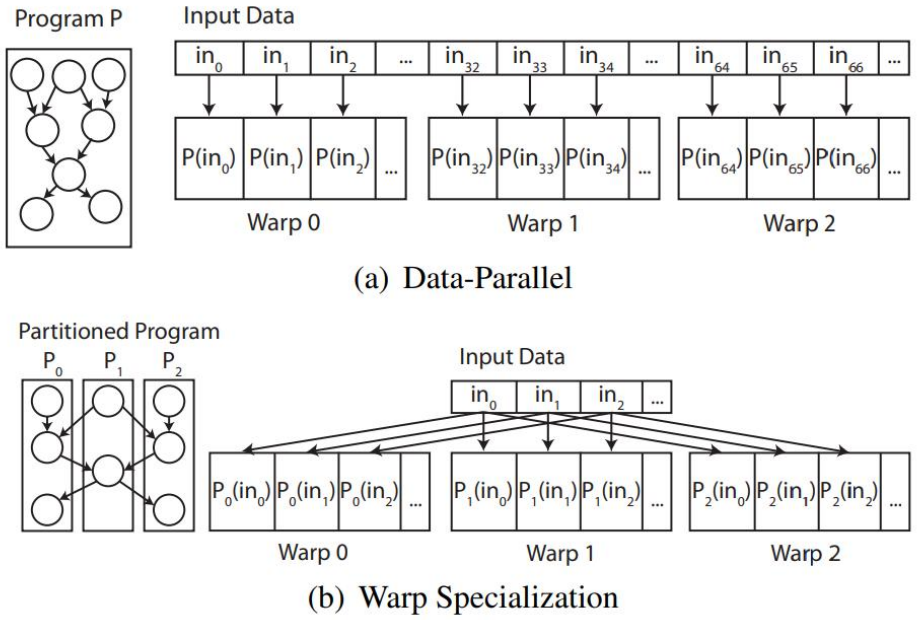
DeepSeek: Warp Specialization

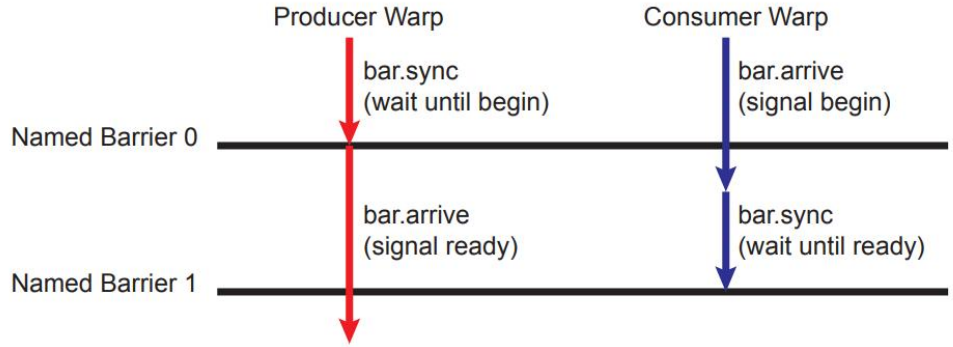
kernels. Specifically, we employ customized PTX (Parallel Thread Execution) instructions and auto-tune the communication chunk size, which significantly reduces the use of the L2 cache and the interference to other SMs.

DeepseekV3: Sec 3.2.2

132 SMs :
20 SMs for communication
112 SMs for computing

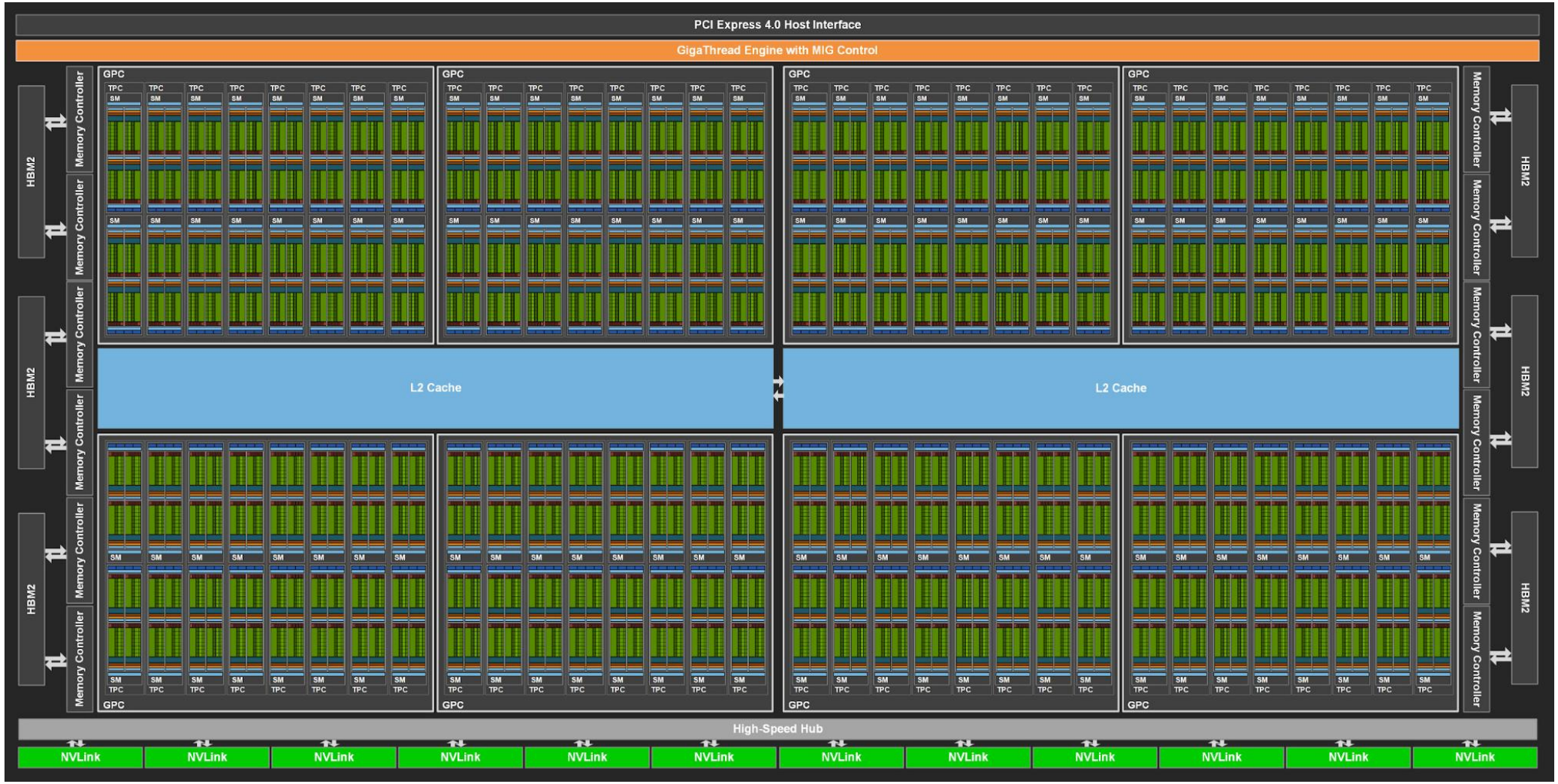






Bauer M, Treichler S, Aiken A. Singe: Leveraging warp specialization for high performance on gpus[C]//Proceedings of the 19th ACM SIGPLAN symposium on Principles and practice of parallel programming. 2014: 119-130.

How the GPU Calculates

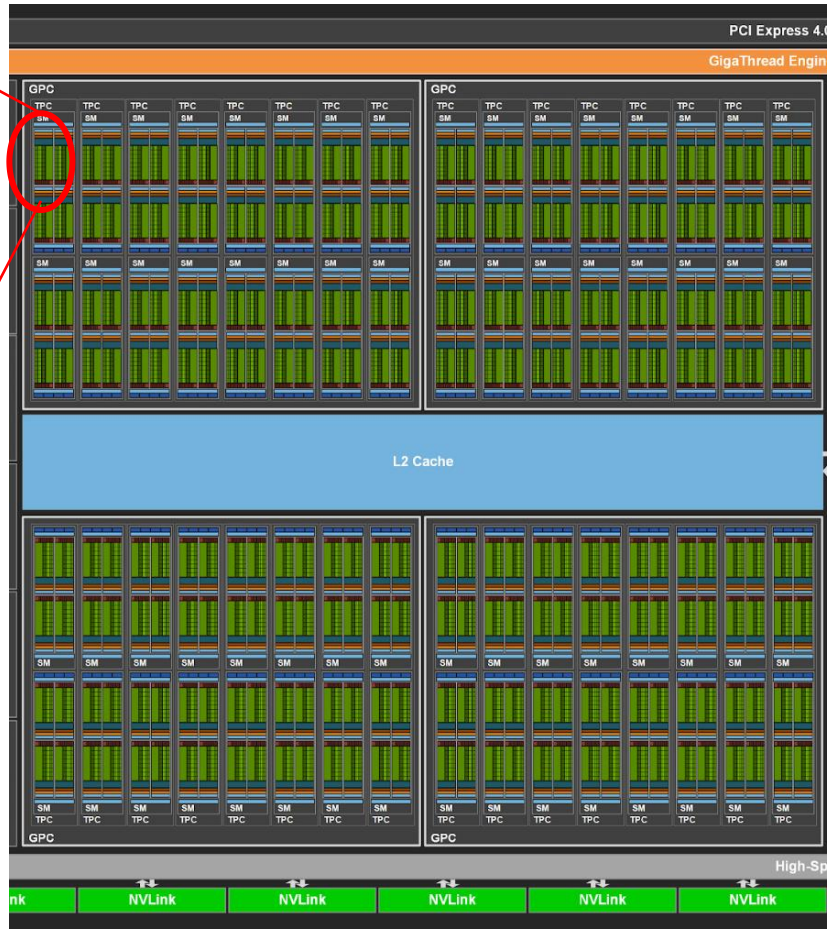


A100 GPU with 128 SMs

https://resources.nvidia.com/en-us-genomics-ep/ampere-architecture-white-paper?xs=169656

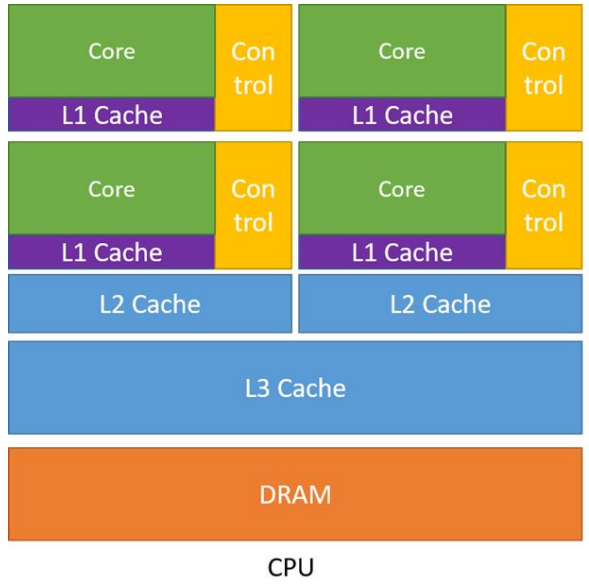
How the GPU Calculates

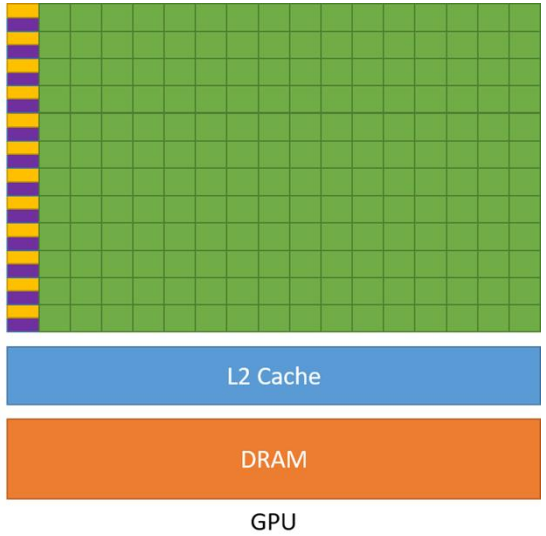
A100 GPU Streaming Multiprocessor(SM)



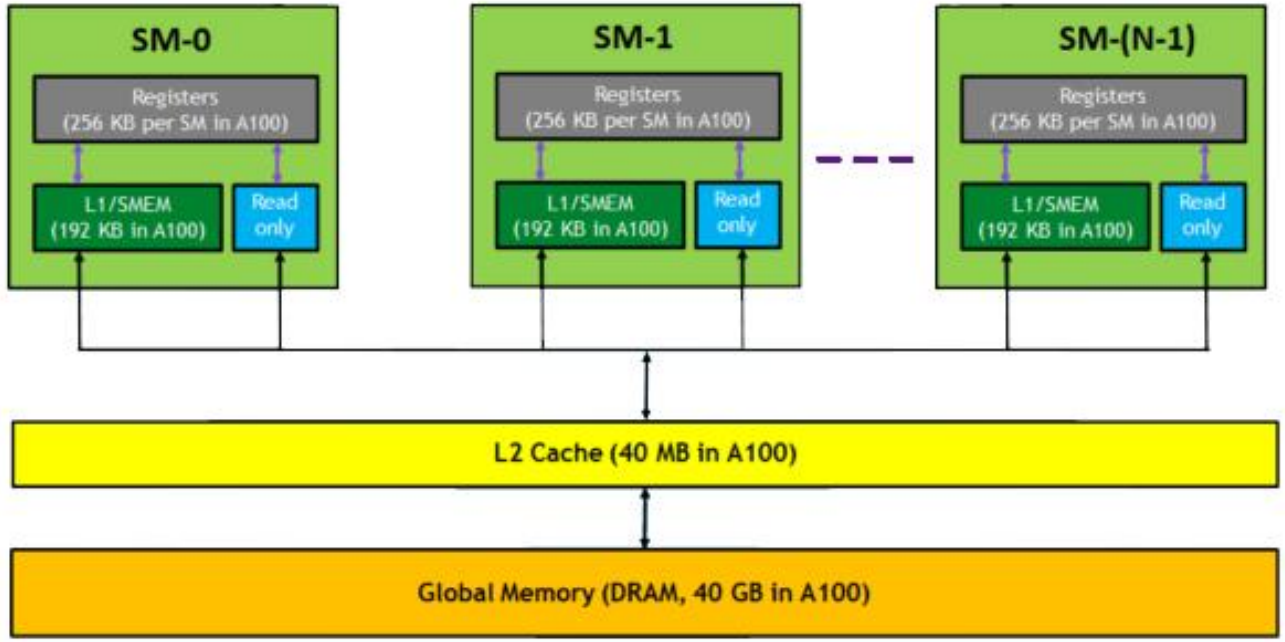
https://resources.nvidia.com/en-us-genomics-ep/ampere-architecture-white-paper?xs=169656

GPU Memory Hierarchy



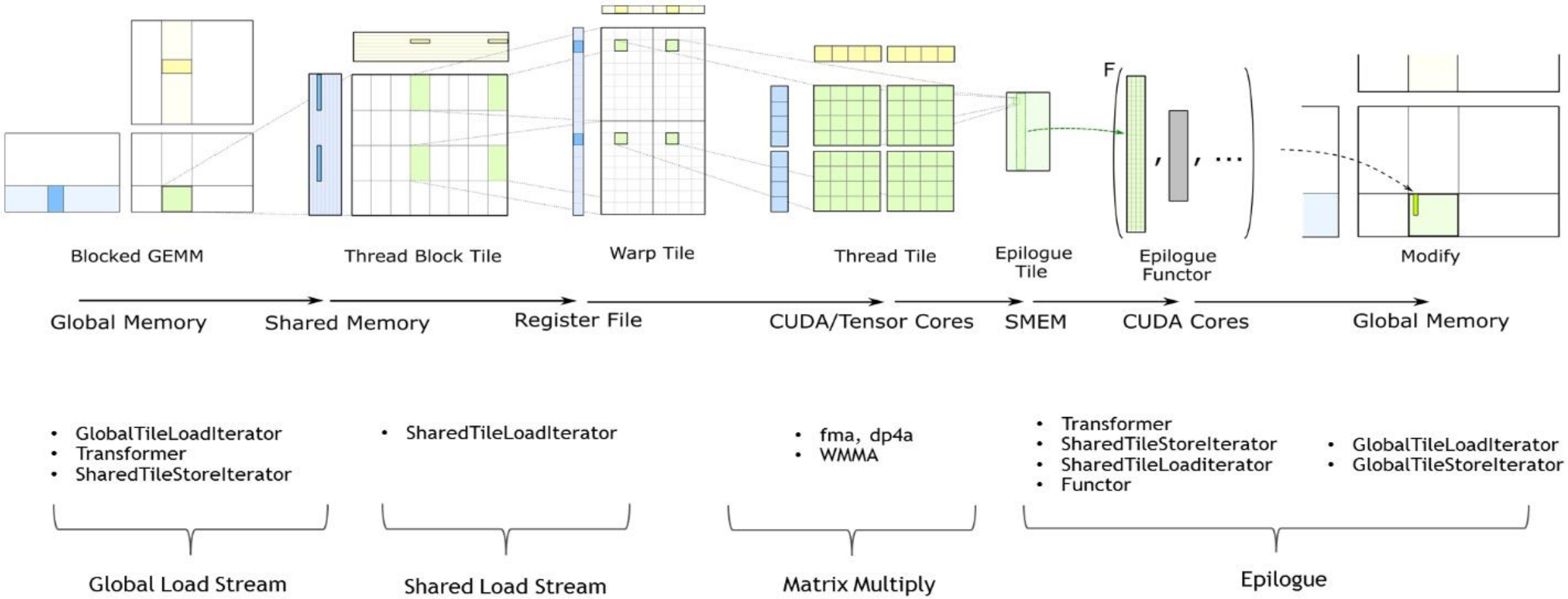


The GPU Devotes More Transistors to Data Processing



https://docs.nvidia.com/cuda/cuda-c-programming-guide/

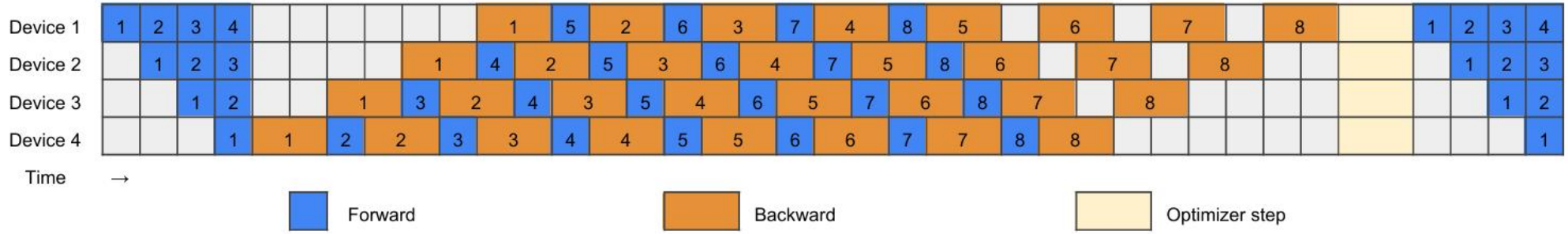
GPU Memory Hierarchy



CUTLASS GEMM Structural Model

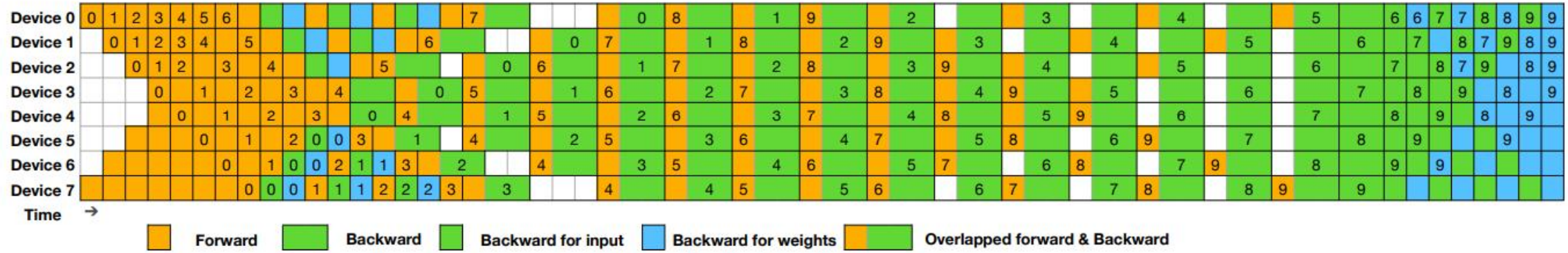
https://developer.nvidia.com/blog/cutlass-linear-algebra-cuda/

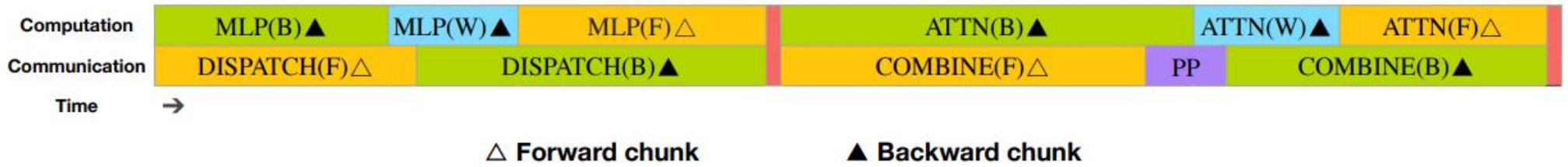
DeepSeek: Pipeline Parallel (DualPipe)



1F1B Pipeline

Qi P, Wan X, Huang G, et al. Zero bubble pipeline parallelism[J]. arXiv preprint arXiv:2401.10241, 2023.





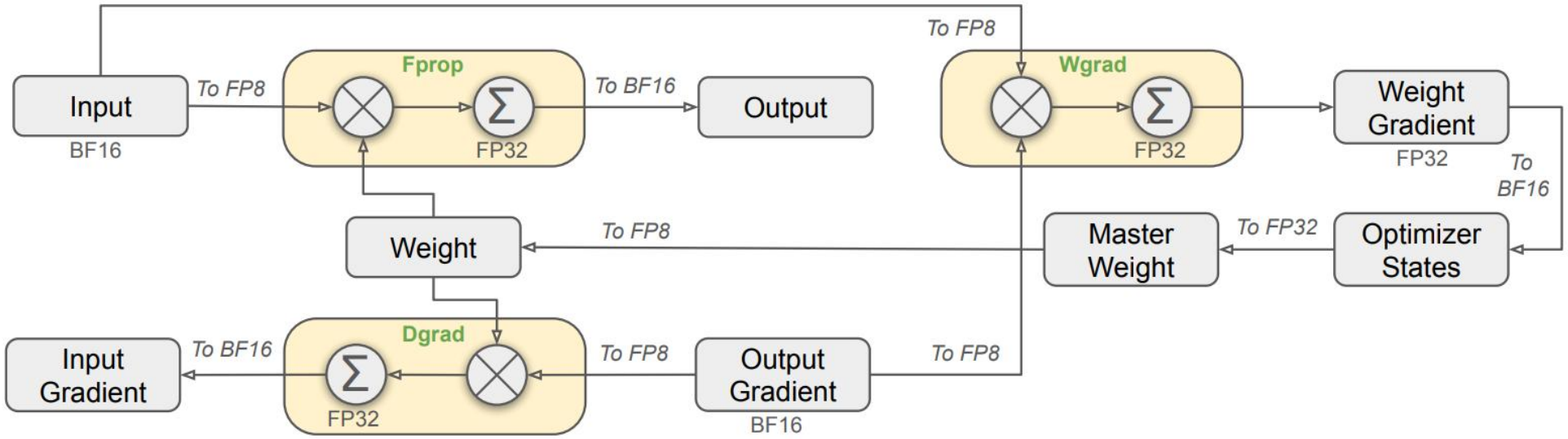
DeepSeek: Memory

Recomputation of RMSNorm and MLA Up-Projection. We recompute all RMSNorm operations and MLA up-projections during back-propagation, thereby eliminating the need to persistently store their output activations. With a minor overhead, this strategy significantly reduces memory requirements for storing activations.

Exponential Moving Average in CPU. During training, we preserve the Exponential Moving Average (EMA) of the model parameters for early estimation of the model performance after learning rate decay. The EMA parameters are stored in CPU memory and are updated asynchronously after each training step. This method allows us to maintain EMA parameters without incurring additional memory or time overhead.

Shared Embedding and Output Head for Multi-Token Prediction. With the DualPipe strategy, we deploy the shallowest layers (including the embedding layer) and deepest layers (including the output head) of the model on the same PP rank. This arrangement enables the physical sharing of parameters and gradients, of the shared embedding and output head, between the MTP module and the main model. This physical sharing mechanism further enhances our memory efficiency.

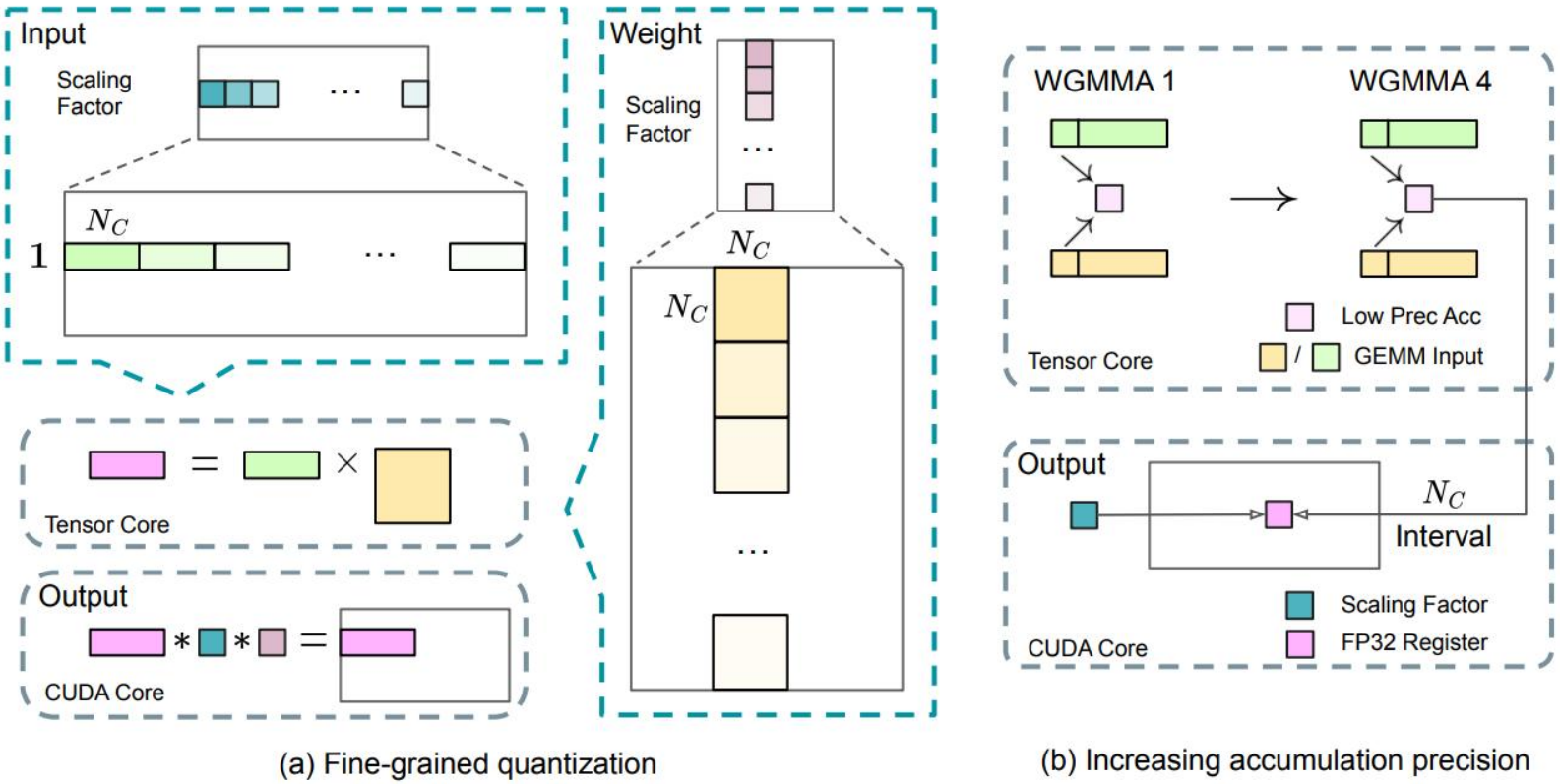
DeepSeek: Mixed Precision



Nvidia: FP8 for GEMM

Outliers in activations, weights, and gradients.

DeepSeek: Mixed Precision



(a) Fine-grained quantization

(b) Increasing accumulation precision

Figure 7 | (a) We propose a fine-grained quantization method to mitigate quantization errors caused by feature outliers; for illustration simplicity, only Fprop is illustrated. (b) In conjunction with our quantization strategy, we improve the FP8 GEMM precision by promoting to CUDA Cores at an interval of $N_C = 128$ elements MMA for the high-precision accumulation.

Activation: Group and scale elements on a 1x128 tile basis(per token per 128 channels)

Weight: 128x128 block basis (per 128 input channels per 128 output channels)

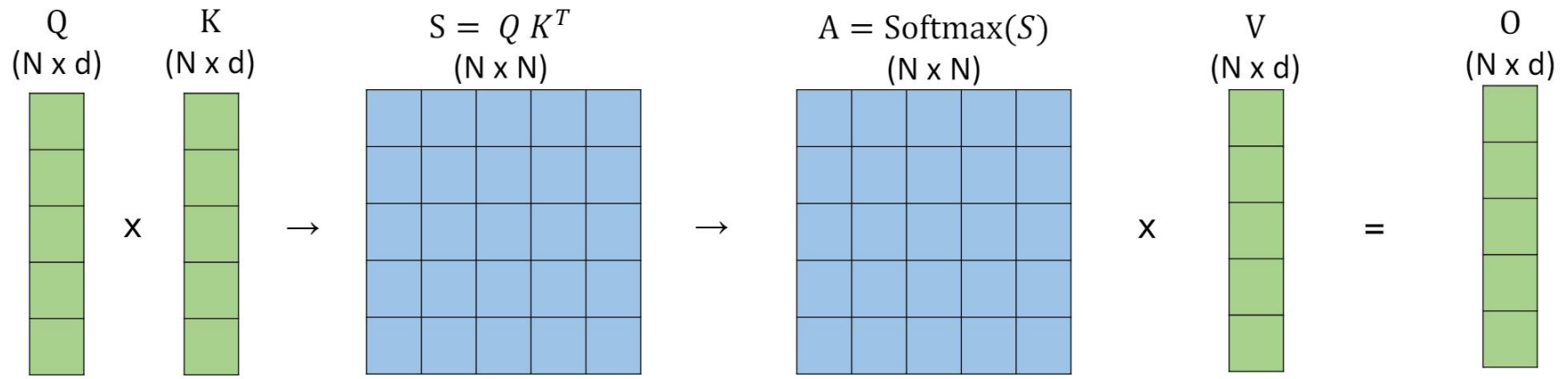
E4M3 (4-bit exponent and 3-bit mantissa) in Fprop.

E5M2 (5-bit exponent and 2-bit mantissa) in Dgrad and Wgrad.

DeepSeek: MoE

1. Expert Parallelism
    1. IB band and NVLink
    2. Warp Specialization (PTX)
2. Pipeline Parallelism: DualPipe
3. Data Parallelism
4. Memory Saving Tips
5. Quantification

MLA: Make Attention Faster



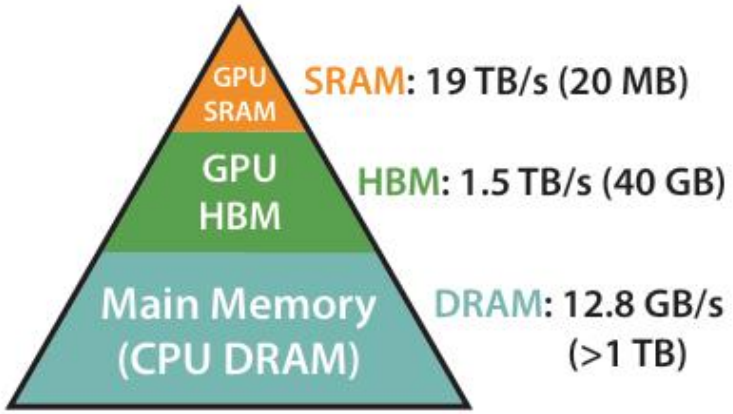
Typical sequence length N: 1K – 8K
Head dimension d: 64 – 128

$$\text{Softmax}([s_1, \cdots, s_N]) = \left[\frac{e^{s_1}}{\sum_i e^{s_i}}, \cdots, \frac{e^{s_N}}{\sum_i e^{s_i}}\right]$$

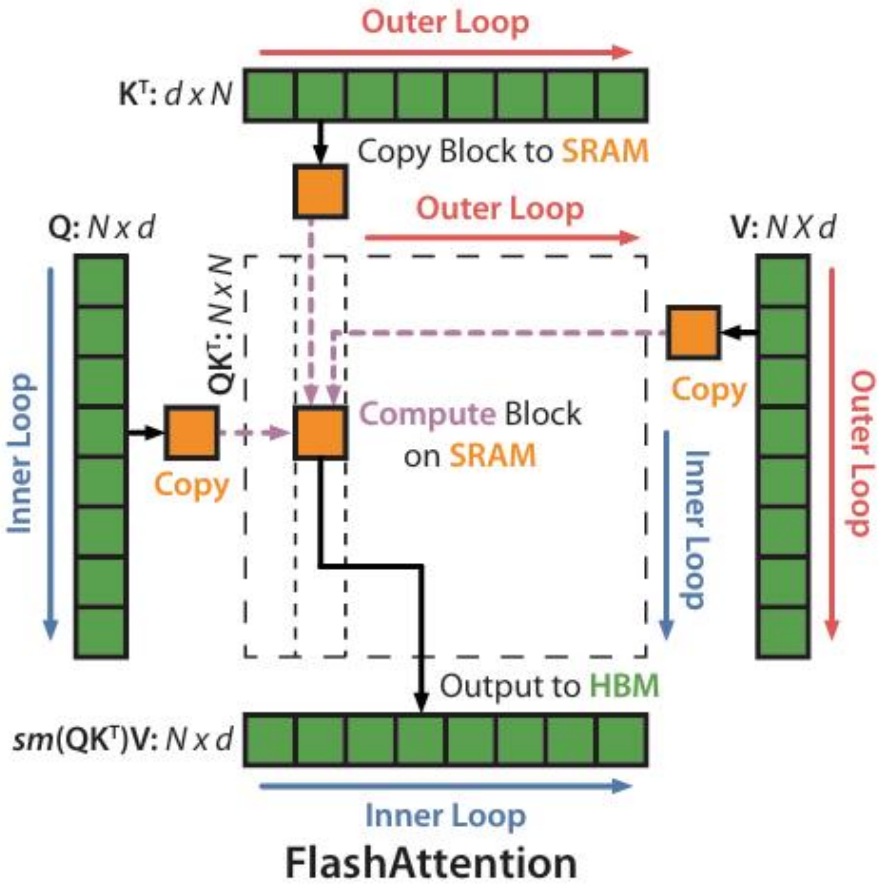
$$O = \text{Softmax}(QK^T)V$$

1: Load $\mathbf{Q}, \mathbf{K}$ by blocks from HBM, compute $\mathbf{S} = \mathbf{Q}\mathbf{K}^\top$, write $\mathbf{S}$ to HBM.
2: Read $\mathbf{S}$ from HBM, compute $\mathbf{P} = \text{softmax}(\mathbf{S})$, write $\mathbf{P}$ to HBM.
3: Load $\mathbf{P}$ and $\mathbf{V}$ by blocks from HBM, compute $\mathbf{O} = \mathbf{P}\mathbf{V}$, write $\mathbf{O}$ to HBM.
4: Return $\mathbf{O}$.

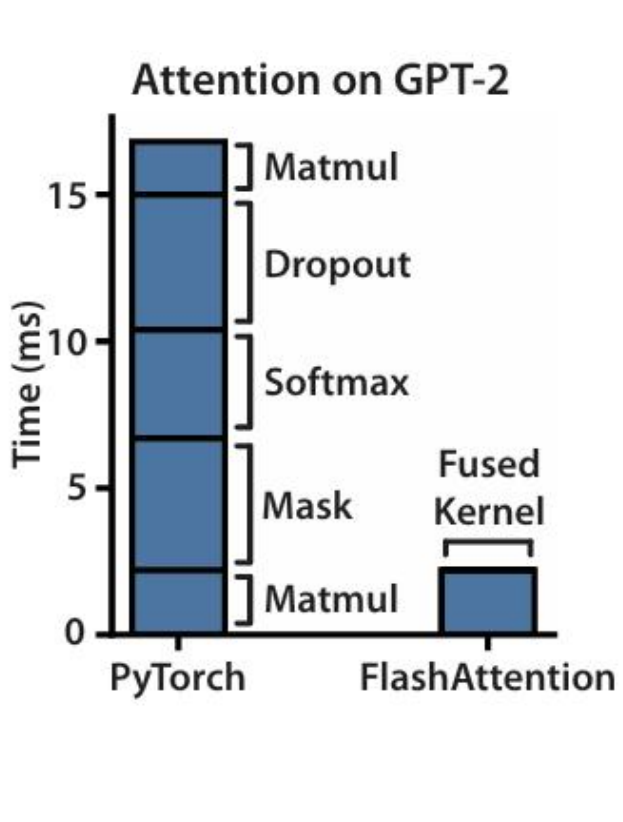
FlashAttention



Memory Hierarchy with Bandwidth & Memory Size



FlashAttention



HBM access

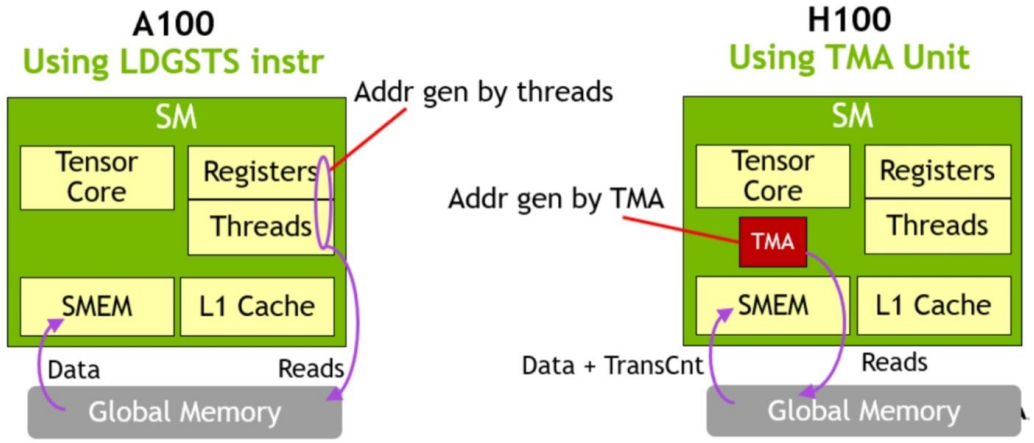
FlashAttention: $\Theta(N^2 d^2 M^{-1})$

StandardAttention: $\Theta(Nd + N^2)$

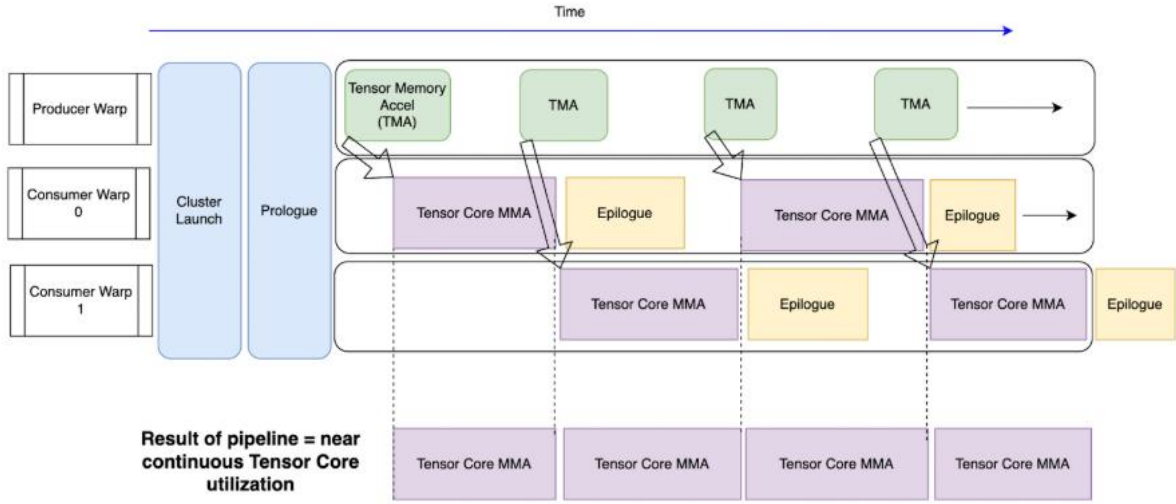
d: 64/ 128
N: ~1024
M: ~100KB (A100)

Attention	Standard	FLASHATTENTION
GFLOPs	66.6	75.2
HBM R/W (GB)	40.3	4.4
Runtime (ms)	41.7	7.3

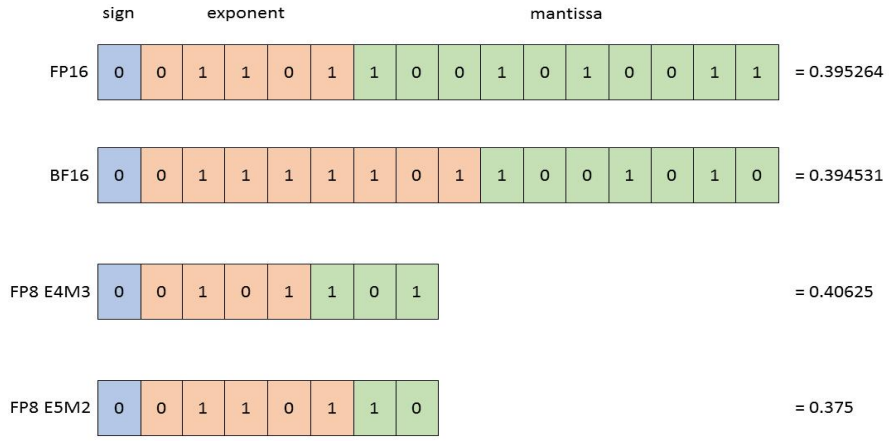
FlashAttention3



Tensor Memory Accelerator is a hardware component introduced with H100's that asynchronously handles the transfer of memory from HBM to shared memory.



PingPong Architecture (1 producer, 2 consumers)



FP8 in Hopper

higher precision

higher dynamic range

https://developer.nvidia.com/blog/nvidia-hopper-architecture-in-depth/

FlashMLA

FlashMLA is an efficient MLA decoding kernel for Hopper GPUs, optimized for variable-length sequences serving.

Currently released:

- BF16
- Paged kvcache with block size of 64

Quick start

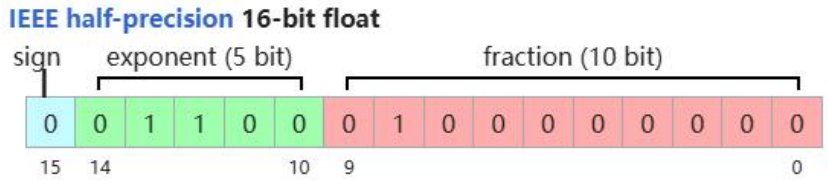
Install

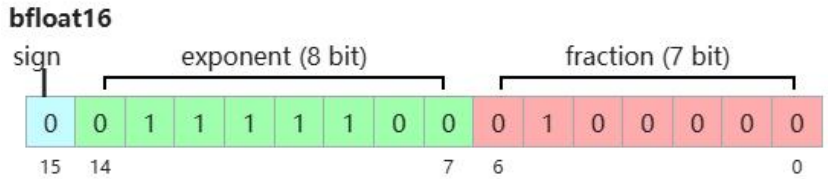
```
python setup.py install
```

Benchmark

```
python tests/test_flash_mla.py
```

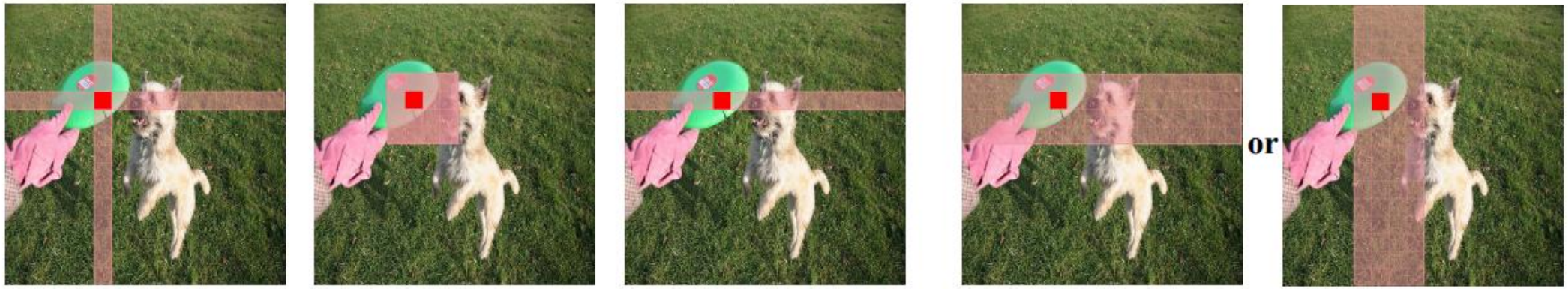
Achieving up to 3000 GB/s in memory-bound configuration and 580 TFLOPS in computation-bound configuration on H800 SXM5, using CUDA 12.6.





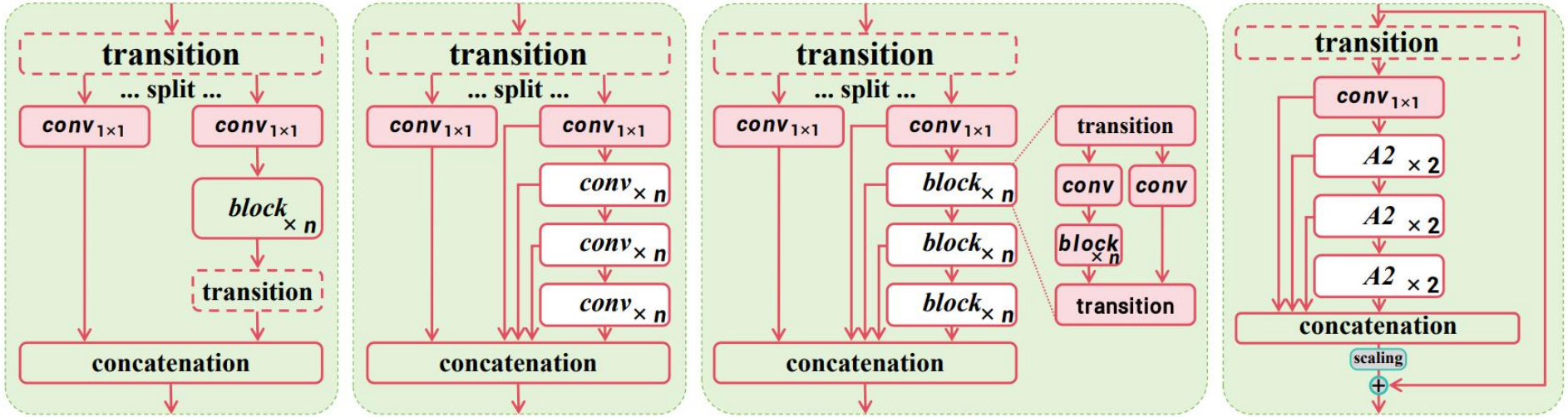
https://github.com/deepseek-ai/FlashMLA

YOLOV12 (25.2.19)



Criss-cross attention | Window attention | Axial attention | Area attention (Ours)

+FlashAttention



(a) CSPNet (b) ELAN (c) C_3K_2 (d) R-ELAN

YOLOV12

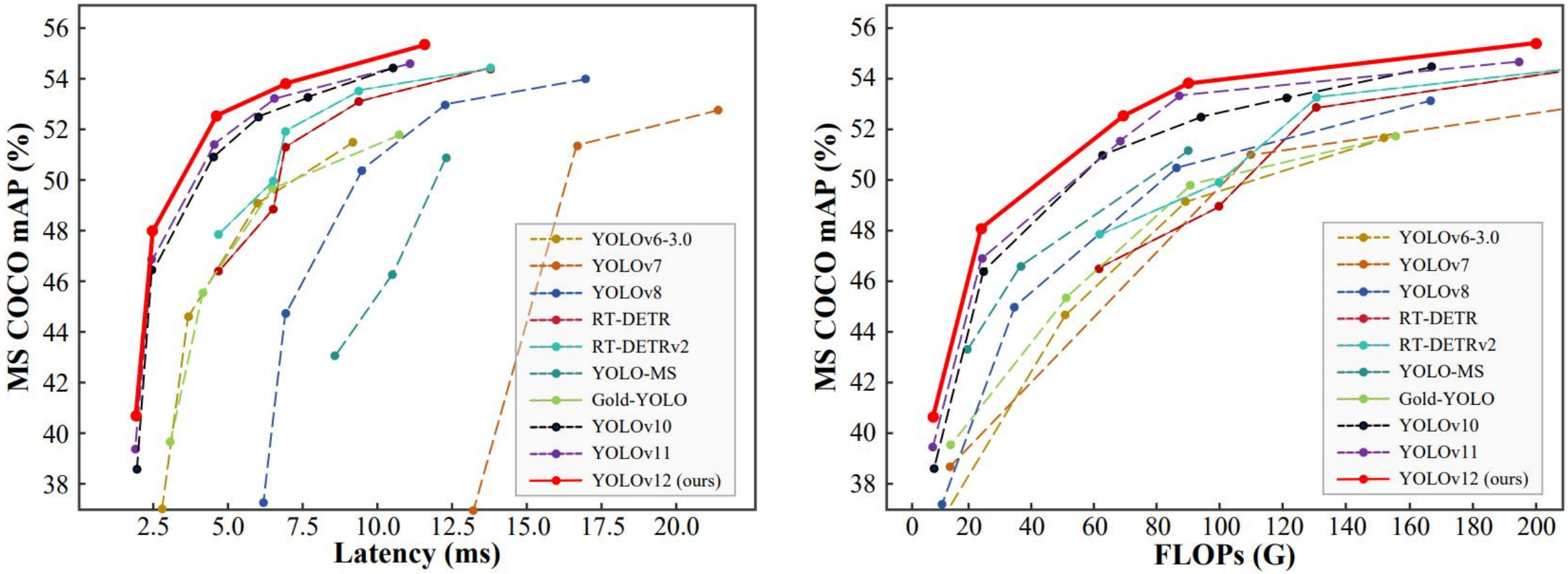


Figure 1. **Comparisons with other popular methods in terms of latency-accuracy (left) and FLOPs-accuracy (right) trade-offs.**

Summary

Next Generation Real-Time Object Detection

Synergistic Optimization of Hardware and Software

Embedded Devices

Vision-Language Model (VLM)