ASAG: Building Strong One-Decoder-Layer Sparse Detectors via Adaptive Sparse Anchor Generation

Shenghao Fu[1,3,4], Junkai Yan[1,4], Yipeng Gao[1,4], Xiaohua Xie[1,3,4*], Wei-Shi Zheng[1,2,3,4*]
[1]School of Computer Science and Engineering, Sun Yat-sen University, China, [2]Pengcheng Lab, China,
[3]Guangdong Province Key Laboratory of Information Security Technology, China,
[4]Key Laboratory of Machine Intelligence and Advanced Computing, Ministry of Education, China
{fushh7, yanjk3, gaoyp23}@mail2.sysu.edu.cn, xiexiaoh6@mail.sysu.edu.cn, wszheng@ieee.org

Abstract

Recent sparse detectors with multiple, e.g. *six, decoder layers achieve promising performance but much inference time due to complex heads. Previous works have explored using dense priors as initialization and built one-decoder-layer detectors. Although they gain remarkable acceleration, their performance still lags behind their six-decoder-layer counterparts by a large margin. In this work, we aim to bridge this performance gap while retaining fast speed. We find that the architecture discrepancy between dense and sparse detectors leads to feature conflict, hampering the performance of one-decoder-layer detectors. Thus we propose Adaptive Sparse Anchor Generator (ASAG) which predicts dynamic anchors on patches rather than grids in a sparse way so that it alleviates the feature conflict problem. For each image, ASAG dynamically selects which feature maps and which locations to predict, forming a fully adaptive way to generate image-specific anchors. Further, a simple and effective Query Weighting method eases the training instability from adaptiveness. Extensive experiments show that our method outperforms dense-initialized ones and achieves a better speed-accuracy trade-off. The code is available at https://github.com/iSEE-Laboratory/ASAG.*

1. Introduction

Object detection is a fundamental and challenging computer vision task. Different from traditional CNN-based dense object detectors [27, 10, 20, 21, 32, 40] using sliding-window paradigm, query-based sparse detectors [2, 43, 31, 9] use hundreds of object queries to search through the whole image, each representing an object or background. They get rid of some traditional hand-crafted components and procedures, *e.g.*, anchors and Non-Maximum Suppression (NMS), which greatly simplifies the detection pipeline and makes the detector fully end-to-end trainable. With the help of powerful transformer encoder-decoder architecture, sparse detectors show promising performance.

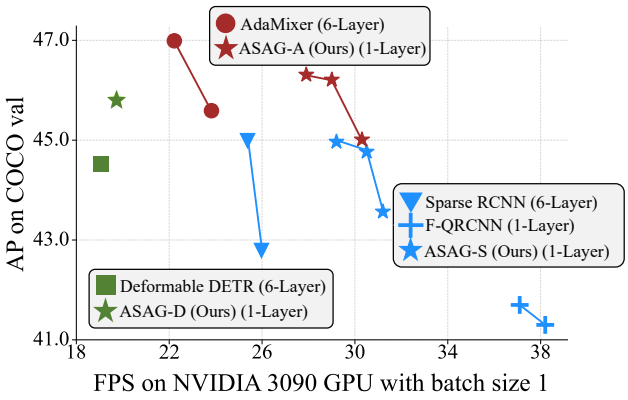

Figure 1: Comparing one-decoder-layer detectors with six-decoder-layer counterparts on FPS and AP with various decoder types. ASAGs achieve a better speed-accuracy trade-off. Best viewed in color.

However, sparse detectors cost more inference time since they need n decoder layers (typically $n = 6$) to progressively refine bounding boxes, leading to a more complex head. Inference with only one decoder layer achieves a much faster speed, such as AdaMixer [9] (43 V.S. 22 FPS) and Sparse RCNN [31] (43 V.S. 25 FPS). The more complex the decoder is, the larger the FPS gap is. Unfortunately, simply discarding the extra decoder layers results in a significant performance drop.

Recently, several methods [36, 41] have attempted to bridge the performance gap between one- and six-decoder-layer detectors. Efficient DETR [36] finds that image-agnostic box and content queries should be blamed for a severe drop in performance when using one decoder layer.

* denotes the corresponding authors.

Thus, both methods utilize an extra dense box prediction step before the decoder to provide proper query initialization. However, they fail to achieve comparable results to detectors with six decoder layers, for example Featurized Query RCNN [41] is lower than Sparse RCNN [31] by 1.5AP with 100 queries. We find that the features corresponding to dense and sparse detectors are significantly different. Such feature conflict hampers the performance of one-decoder-layer detectors, demonstrated in Section 3. Thus, although these methods achieve remarkable acceleration, it still has much room to narrow the performance gap.

In this work, we aim to build a fully sparse one-decoder-layer detector, narrowing the performance gap between one- and six-decoder-layer detectors and retaining the fast speed. The key difference between our method and other dense initialization methods is that we use patches as the basic prediction units, which can be the whole or part of an image. Sparsely predicting on patches alleviate the feature discrepancy caused by predicting on grids and enjoys global receptive fields. Further, we propose to loose the constraint that each image should use a fixed number of queries to detect objects so that more complex images detect objects with more queries and vice versa. Based on these two preconditions, we propose initializing image-specific queries using Adaptive Sparse Anchor Generator (ASAG), which is fully adaptive to each image in both anchors' locations and numbers. We further design Adaptive Probing to adaptively crop patches on possible locations on different feature map levels. It runs in a top-down and coarse-to-fine way and greatly enhances the ability to detect small objects. Finally, an effective Query Weighting method is proposed to handle the instability coming from adaptiveness.

We conduct extensive experiments on the COCO [22] dataset with various decoder types. As shown in Figure 1, our model ASAG-S outperforms dense-initialized Query RCNN [41] by 2.6 AP with fewer FLOPs and the same decoder. We also retain the fast speed of one-decoder-layer detectors, thus achieving a better speed-accuracy trade-off.

2. Related Works

Improving Sparse Detectors. Recently, DETR [2] views object detection as a set-prediction problem and achieves promising performance. But it still has some apparent disadvantages and lacks interpretability. Many following works have solved the problems like slow convergence and relatively low performance on small objects by utilizing multi-scale feature pyramids [43, 9, 38], introducing more spatial priors [43, 8, 25, 34, 23, 9], stabilizing bipartite matching [16, 39], aligning feature space [37, 38], increasing positive samples [13, 4, 14, 44, 26], using knowledge distillation [12, 3, 5], initializing queries [43, 39, 11], *etc*. Although achieving outstanding performance and fast convergence speed, transformer-based sparse detectors still require more inference time than CNN-based dense detectors thus limiting their practical applications.

NO.	Detector	Init.	AP	AP_s	AP_m	AP_l
(a)	Deformable DETR+ [43]	learned	46.2	28.3	49.2	61.5
(b)	Deformable DETR+†		37.9	23.1	41.9	49.1
(c)	Deformable DETR++ [43]	dense	46.9	29.4	50.1	61.6
(d)	Deformable DETR++†		33.7	23.1	38.4	41.1

Table 1: Effect of dense query initialization on one- and six-decoder-layer detectors. The first decoder layer with dense initialization even underperforms the one without image-specific initialization. †: Inference with the first layer.

Accelerating Sparse Detectors. As a well-known experience, most computation for a detector lies in the backbone due to high-resolution feature maps and dense computation. However, as sparse detectors have more complicated operators in neck and head, *e.g*., attention and grid sampling, the FLOPs cannot directly reflect the FPS. For example, the decoder of AdaMixer [9] takes 31% of the total FLOPs but nearly half of the FPS. Different from works [42, 33, 29] focusing on reducing computation in neck by approximating self-attention in the encoder, we aim to simplify the decoder. Recently, Li *et al*. [19] find that some unimportant queries are not worth being computed equally. However, decreasing the number of queries brings minor acceleration as Sparse RCNN [31] increasing the number of queries from 100 to 300 only uses an extra 1 FPS. Thus decreasing the number of decoder stages is a more promising way to speed up sparse detectors.

3. Why Should We Use Sparse Initialization for One-Decoder-Layer Detectors?

Some previous works [43, 39] have found that utilizing a dense box prediction step as a query initialization can be helpful to six-decoder-layer detectors. For example, Deformable DETR++ [43] outperforms Deformable DETR+ [43] by 0.7 AP as shown in Table 1. However, we surprisingly find that the first decoder layer with better initialization even performs worse than the one with image-agnostic initialization, as shown by (b) and (d) in Table 1. This phenomenon shows that the benefit from the dense box prediction step for six-decoder-layer detectors does not lie in better initialization but in more supervision signals to the encoder, as many works [4, 14, 44, 26] find that one-to-one matching is not sufficient for feature learning.

To better illustrate the phenomenon above, we visualized discriminability scores in the encoder in Figure 3 following [44], which are l^2-norm of the corresponding grid features. Objects with higher discriminability scores can be better detected. Since dense detectors predict objects based on grid features while sparse detectors detect objects using queries, the architecture discrepancy makes the needed

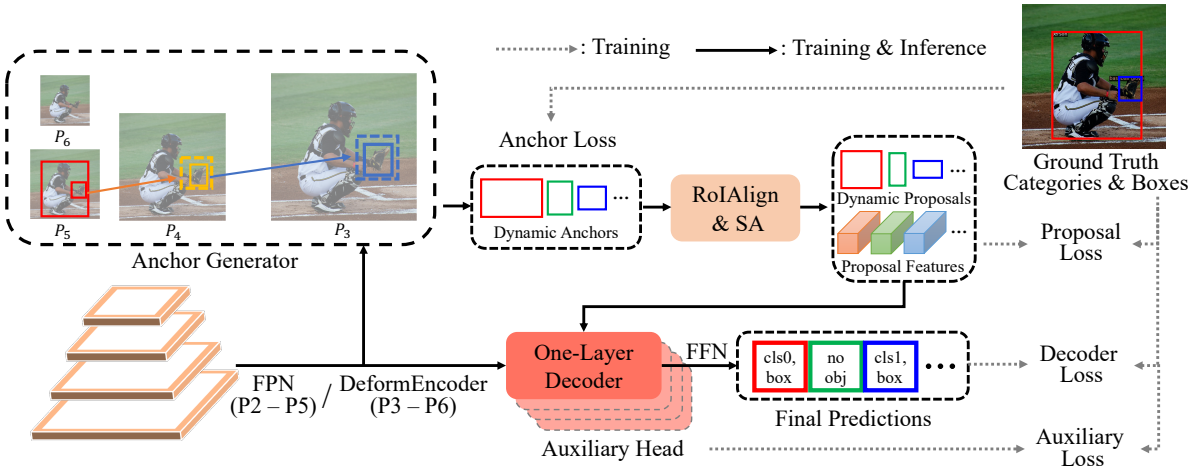

Figure 2: Overview. Our model first uses Anchor Generator to predict dynamic anchors. Then, query features are extracted by RoIAlign [10] and refined by a Self-Attention layer (SA). One decoder layer of any kind is used for final prediction. The neck is changed following the decoder. We additionally add three auxiliary heads using the same proposals to provide more supervision signals, which are discarded during inference. Each component is supervised under one-to-one matching losses.

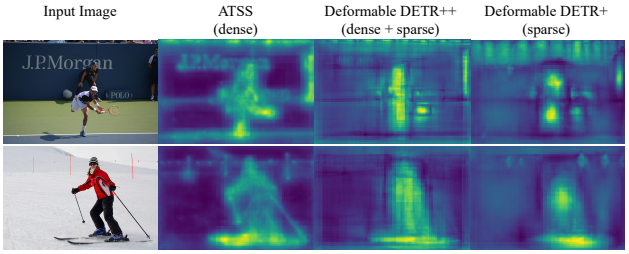

Figure 3: Comparison on feature maps (discriminability scores [44]) of dense and sparse detectors. There exists a clear inconsistency between dense and sparse features.

features totally different. As shown in Figure 3, sparse detectors pay more attention to background information than dense detectors [3]. Moreover, dense detectors prefer to activate the whole object uniformly since each grid has equal chances to predict the object, while sparse detectors tend to highlight some discriminative parts. The score maps of sparse detectors with dense initialization fall between dense and sparse detectors. Due to the powerful decoders with six layers, such detectors with dense initialization can tolerate the discrepancy of features and benefit from more positive signals. However, one-decoder-layer detectors with limited representation ability suffer from conflicting features. We hypothesize that this is why one-decoder-layer detectors with well-initialized queries still lag behind their six-decoder-layer counterparts. This phenomenon also demonstrates that there is redundancy in six-decoder-layer detectors, which can be reduced for acceleration. Thus, we provide a sparse way to further narrow the performance gap between one- and six-decoder-layer detectors and retain the fast speed of one-decoder-layer detectors.

4. Adaptive Sparse Anchor Generation for Query Initialization

4.1. Overview

In this section, we introduce our Adaptive Sparse Anchor Generator (ASAG for short), which initializes queries sparsely and is more suitable for sparse decoders while retaining fast speed. As shown in Figure 2, ASAG generates image-specific anchors adaptively from the aspect of locations and numbers, without using predefined spatial priors. Unlike complex decoders, our ASAG is lightweight, using only 0.06G FLOPs. With dynamic anchors, we use RoIAlign [10] to generate content queries since initializing both box and content queries is vital for one-decoder-layer detectors [36]. Further, an extra Self-Attention layer is utilized to model relationships between objects and reduce redundancy for NMS-free, similar to Featurized QR-CNN [41]. Lastly, a one-layer decoder of any kind [9, 31, 43] is used for final refinements.

Recent works [13, 4, 14, 44, 26] show that sparse detectors benefit from sufficient supervision signals. Considering that dense methods provide supervision to each grid and other six-decoder-layer detectors have more auxiliary losses, we additionally add three auxiliary parallel one-layer decoders for a fair comparison. Thus, our models are also supervised by six one-to-one matching losses. Different from Group DETR [4] that uses different groups of queries and the shared decoder by each group, we use different decoders with the same proposals.

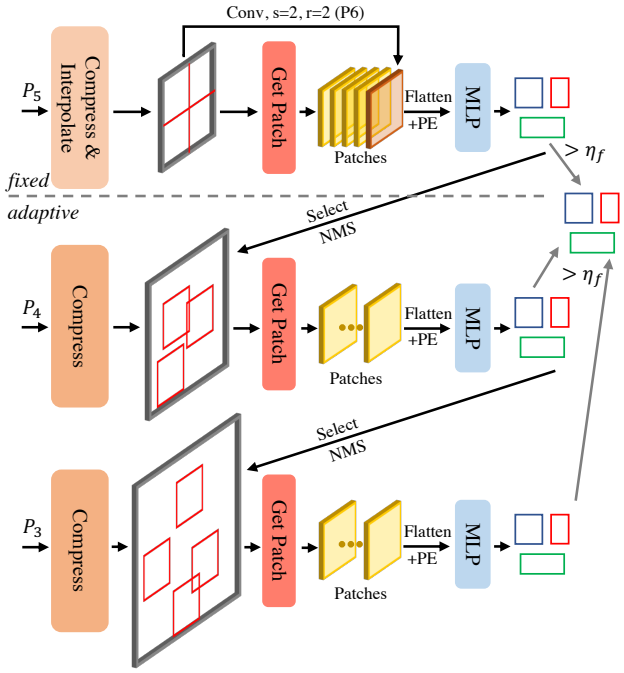

Figure 4: Adaptive Sparse Anchor Generator (ASAG). ASAG starts predicting dynamic anchors from fixed feature maps and then adaptively explores larger feature maps using Adaptive Probing, which runs top-down and coarse-to-fine. Learnable position embeddings (PE) are added to patches after flattening to keep spatial structures.

4.2. Adaptive Sparse Anchor Generator

Patches are better prediction units. In this work, we use **patches** as the basic prediction units, from which possible anchors will be predicted. The patch is the whole or part of an image, which may contain lots of objects, and is much larger than grids or regions of interest. Thus, predicting from patches alleviates the feature discrepancy caused by predicting from grids and enjoys global receptive fields.

Inference with the fixed number of feature maps. We start demonstrating our method from using a fixed number of feature maps to predict all objects, *i.e.* P_5[1] and P_6. As shown in the upper part of Figure 4, taking the P_5 feature map as input, we first compress the feature map to a few channels along the channel dimension to save computation and parameters. To handle images with different sizes, we interpolate P_5 to a fixed size and then evenly split it into four patches from top-left to bottom-right to narrow the predictor's search space. If we directly view the whole P_5 feature map as a single patch, the model tends to overlook some small objects. Considering that some large objects may appear across the patches, P_6 feature map is used to handle the problem, which is downsampled from P_5 after interpolating by a factor of 2, and viewed as a single patch (the brown patch). We use an MLP as the predictor to predict anchors of a fixed number on each patch simultaneously, each described by four coordinates and a location score. The location score can be seen as the class probability so that it is class-agnostic and is supervised by IoU [18] as a soft label.

Inference with a dynamic number of feature maps. Since P_5 is not sufficient to predict small objects, using feature maps with larger sizes can obtain more precise anchors. Motivated by QueryDet [35], we propose to sparsely compute on large feature maps using **Adaptive Probing** to correct the anchors for small objects with low confidence.

As shown in the lower part of Figure 4, we select some anchors predicted from the fixed part, whose confidences fall in $[\eta_l, \eta_h]$ and sizes are smaller than half of the patch size. Anchors with scores lower than η_l are seen as noisy anchors, and anchors with scores higher than η_h are accurate enough. Thus such anchors are not used in Adaptive Probing. With selected anchors, we crop some patches on the P_4 feature map, whose centers are the corresponding anchors' ones. Since patches may overlap with each other, we use NMS with IoU threshold η_{iou} to reduce the redundancy. Anchors predicted from the higher resolution feature maps are more precise than the original ones, thus we replace them with the newly generated ones. Such probing iterations continue to perform on the following larger feature maps until the largest one P_3. We empirically find that additionally using P_2 brings minor improvement but adds more inference time. Once no anchors are selected, the iterations break with an early-stop mechanism. Thus, the probing is adaptive to the number of iterations, the number of patches, and the locations of patches. Finally, all anchors with scores higher than η_f and not being selected for Adaptive Probing are gathered. Considering that the model needs to deal with pictures with different difficulties, the number of generated patches and anchors varies accordingly. We pad the output anchors to the max size for parallel processing.

Training ASAG. We first define three kinds of patches: 1) *generated patch* which is generated from selected anchors, 2) *GT patch* which is gained by grouping ground truth boxes smaller than half of the patch size, and 3) *random patch* which is generated randomly. To ensure that predictors for each pyramid level are fully and equally trained, we define a minimal training patch number $N_{TP} = 4$ for each level since the lower level tends to receive fewer supervision signals due to the early-stop mechanism. For feature maps from P_5 to P_3, we use the generated patch, GT patch, and random patch in turn until the minimum patch number is met. For P_6, since we view the whole feature map as a single patch, we flip the patch horizontally and vertically to meet the minimum patch number. Only anchors generated from generated patches with confidence scores higher

[1] P_l denotes the feature map downsampled by a factor of 2^l.

than η_f are gathered and sent to the following model parts. The targets for each patch are objects whose centers lie in the patch. Since the output anchors are unordered, bipartite matching is used to get one-to-one matching, similar to other parts of our model and other sparse detectors, except it is class-agnostic. Further, we use IoUs between ground truth boxes and the matched anchors as the soft labels [18].

Relationships with related works. The Adaptive Probing computes sparsely on large feature maps to save computation, similar to PointRend [15] and QueryDet [35]. However, both are dense prediction methods and it is clear where to explore on the larger feature map while we sparsely find the corresponding location. Further, QueryDet predicts objects in a **divide-and-conquer** way, but the Adaptive Probing is in a **correct-and-replace** manner thus we enjoy the early-stop mechanism. We can even discard large feature maps manually for efficient inference, as shown in Table 2.

4.3. Stablizing Training

Through the novel design above, we gain adaptive sparse anchors and proposals efficiently. However, as shown in Figure 5, we find that our dynamic anchors have two characteristics that are significantly different from the traditional hand-crafted anchors: 1) dynamic anchors may not be as precise as predefined anchors in the early training stage, 2) dynamic anchors change both in quality and numbers along the training process, making the detection head hard to optimize. It is unsuitable for treating two anchors with 0.1 and 0.9 IoU equally since it confuses the detectors about the definition of positive samples. Thus, we propose **Query Weighting** to ease the training difficulty by giving high-quality anchors with larger weights and vice versa. Soft labels make detectors pay more attention to precise predictions, stabilizing the training when dynamic anchors change, especially in the early training process. This simple weighting mechanism introduces no inference cost.

Motivated by DW [17] to give diverse positive and negative loss weights, our weighting functions are as follows:

$$\text{Norm}(x_1, x_2) = \sigma((x_1 \times x_2 - 1/3) \times 4.5) \div \sigma(3), \quad (1)$$

$$w_{pos} = \text{Norm}(s^{\gamma_1}, IoU^{\gamma_2}), \quad (2)$$

$$w_{neg} = \text{Norm}(s^{\gamma_1}, P_{neg}(IoU^{\gamma_2})) - \sigma(-1.5), \quad (3)$$

where s and IoU are classification scores s and IoUs, P_{neg} is the same function as in [17], and σ denotes sigmoid function. After normalizing, the positive weights are roughly in [0.2, 1] and the negative weights are in [0, 0.8]. Since the sigmoid function is non-linear, it raises the small values while still keeping them within [0, 1]. Even if the matched anchors do not overlap with the targets, we cannot assign the positive weights to zeros as no other anchors will be assigned to the targets in the one-to-one label assignment.

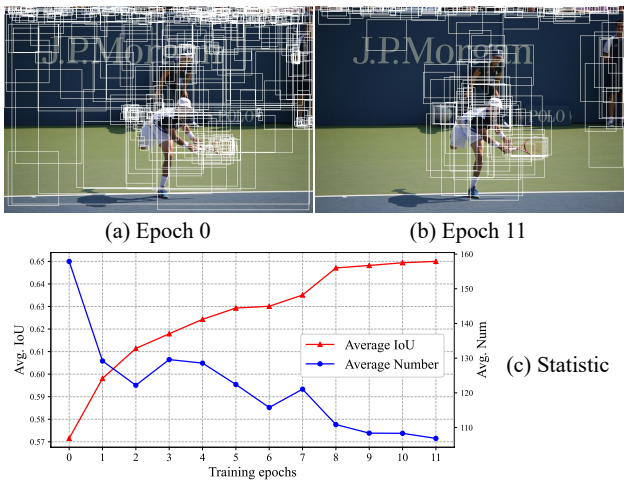

Figure 5: Visualizing dynamic anchors during the training process. The quality and number of anchors change along the training process making the model hard to optimize.

And we avoid assigning ones to negative weights of the only matched anchors. The weights only apply to losses not to matching costs.

Different from label weighting methods [18, 7, 17] in dense detectors, which align the separate regression and classification heads by giving larger weights to more suitable anchors within the candidate bag of each ground truth, Query Weighting is to tolerate dynamic anchors during the training process. Thus our Query Weighting is global-wise while label weighting is instance-wise. We show that six-decoder-layer detectors do not benefit from Query Weighting due to regressing from fixed queries in Section 5.5.

5. Experiments

5.1. Settings

Dataset. We conduct ASAG experiments on the widely used detection dataset COCO2017 [22]. We train our model on `train2017` (∼118k images) and report evaluation metrics on `val2017` containing 5k images. We find each parallel decoder performs similarly (∼0.2 AP).

Configurations. To show the generalizability and make a fair comparison, we conduct experiments with well-known decoders, such as AdaMixer [9], Sparse RCNN [31], Deformable DETR [43], and our corresponding models are ASAG-A, ASAG-S, and ASAG-D, respectively. Featurized Query RCNN [41] and Efficient DETR [36] are dense-initialized one-decoder-layer detectors of Sparse RCNN and Deformable DETR, respectively. We compare with these methods using a similar average number of anchors fairly. For the 100 queries setting, we set the range of the number of anchors for each image as [5, 200], η_f as 0.1, the number of predicted anchors for each patch on the fixed

Detector	FL	#An	#L	AP	AP_{50}	AP_{75}	AP_s	AP_m	AP_l	FPS
AdaMixer	-	100	6	42.7	61.5	45.9	24.7	45.4	59.2	23.8
	-	300	6	44.1	63.4	47.4	27.0	46.9	59.5	22.2
ASAG-A	$P_{3\text{-}6}$	107	1	42.6	60.5	45.8	25.9	45.8	56.9	28.9
	$P_{4\text{-}6}$	97	1	42.1	59.9	45.7	24.8	46.0	56.9	30.3
	$P_{5\text{-}6}$	87	1	40.3	57.5	43.3	21.2	44.4	57.1	31.3
	$P_{3\text{-}6}$	329	1	43.6	62.5	47.0	26.9	46.2	57.6	27.9
	$P_{4\text{-}6}$	313	1	43.4	62.1	46.8	26.4	46.4	57.7	29.0
	$P_{5\text{-}6}$	280	1	41.9	60.5	44.9	23.5	45.5	57.9	30.5

Table 2: Comparison with AdaMixer [9] using the standard $1\times$ schedule and R50. FL means feature map levels used in Anchor Generator. #An and #L denote the average number of anchors and the number of decoder layers, respectively. Models colored in yellow use efficient inference, and in blue use more than one decoder layer. Using only $P_{5\text{-}6}$ means do not use Adaptive Probing totally.

part and the adaptive part as 50 and 20. The corresponding configurations for the 300 queries setting are [50, 500], 0.05, 150, and 50.

Other implementation details. We keep the initial learning rate for the backbone and other parts as 2×10^{-5} and 2×10^{-4}. The batch size is 16. In the standard $1\times$ schedule, the learning rate drops at epoch 8 and 11 by a factor of 0.1. In the $3\times$ schedule, multi-scale training is utilized and the shorter side of images ranges from 640 to 800. The learning rate drops at epoch 24 and 33. No query patterns [34] is used. Following other DETR-like models, we use L1 loss, GIoU loss [28], and classification losses with coefficients 5, 2, 2. The Query Weighting applies to both regression and classification losses similar to [17]. AdamW [24] with weight decay 0.0001 is used as the optimizer.

As for other default hyper-parameters, the patch size in Anchor Generator is 15, thus the interpolate size for P_5 is 30. γ_1 and γ_2 in Query Weighting are 0.4 and 0.6. For fast inference, we stop Adaptive Probing with an early stop if the number of selected anchors is less than 3. And we limit the number of patches for each level within 15. FPS are tested on a single NVIDIA 3090 GPU with batch size 1.

5.2. Main Results

Comparison under the standard $1\times$ schedule. We first fairly compare ASAG-A with AdaMixer [9] with a few epochs. As shown in Table 2, although dynamic anchors are changing and imprecise in the early time, Query Weighting stabilizes the training and ASAG-A still converges in 12 epochs and achieves comparable results with six-decoder-layer AdaMixer with $1.25\times$ speed-up. Since Adaptive Probing runs in an iterative and correct-and-replace way, we can stop at a specific level manually for efficient inference without re-training. Note that efficient inference will not hurt the performance of large objects.

Comparison with one-decoder-layer detectors. Since

Detector		#An	#L	AP	AP_{50}	AP_{75}	AP_s	AP_m	AP_l	F	FPS
F-QRCNN [41]		100	1	41.3	59.4	44.9	26.7	44.2	52.4	140	38.2
F-QRCNN* [41]		300	1	41.7	60.2	45.4	27.7	44.3	52.0	143	37.1
ASAG-S (Ours)	$P_{3\text{-}6}$	100	1	43.9	62.2	47.9	27.8	46.5	57.8	130	30.1
	$P_{4\text{-}6}$	89	1	43.6	61.7	47.5	26.7	46.7	57.8	130	31.2
	$P_{5\text{-}6}$	76	1	41.5	58.9	45.2	23.2	45.4	57.8	130	33.0
ASAG-S (Ours)	$P_{3\text{-}6}$	312	1	45.0	64.1	49.1	29.5	47.4	57.8	136	29.2
	$P_{4\text{-}6}$	292	1	44.8	63.9	48.8	28.9	47.5	57.8	136	30.5
	$P_{5\text{-}6}$	256	1	43.2	61.9	47.1	25.8	46.7	57.8	136	31.7
Effi-DETR [36]		300	1	45.1	63.1	49.1	28.3	48.4	59.0	210	-
ASAG-D (Ours)		253	1	45.8	64.1	49.4	27.3	49.6	61.0	182	19.7
ASAG-A (Ours)		102	1	45.3	63.3	48.9	27.3	48.5	59.7	131	28.9
ASAG-A (Ours)		312	1	46.3	65.1	50.3	29.9	49.2	59.6	139	27.9

Table 3: Comparison with other one-decoder-layer detectors with 36 epochs and R50. *: Reimplement by us using official codes. F denotes GFLOPs.

Detector	#An	#L	AP	AP_{50}	AP_{75}	AP_s	AP_m	AP_l	F	FPS
Sparse RCNN [31]	100	6	42.8	61.2	45.7	26.7	44.6	57.6	134	26.0
Sparse RCNN [31]	300	6	45.0	63.4	48.2	26.9	47.2	59.5	152	25.4
CF-QRCNN [41]	100	2	43.0	61.3	46.8	28.3	45.7	55.5	142	34.1
CF-QRCNN [41]	300	2	44.6	63.1	48.9	29.5	47.4	57.5	148	33.6
ASAG-S (Ours)	100	1	43.9	62.2	47.9	27.8	46.5	57.8	130	30.1
ASAG-S (Ours)	312	1	45.0	64.1	49.1	29.5	47.4	57.8	136	29.2
Deform DETR‡: [43]	300	6	44.5	63.6	48.7	27.1	47.6	59.6	173	19.0
Deform DETR+‡: [43]	300	6	46.2	64.7	49.0	28.3	49.2	61.5	173	19.0
ASAG-D (Ours)	253	1	45.8	64.1	49.4	27.3	49.6	61.0	182	19.7
AdaMixer* [9]	100	6	45.6	64.8	49.3	28.8	48.5	60.9	103	23.8
AdaMixer [9]	300	6	47.0	66.0	51.1	30.1	50.2	61.8	125	22.2
AdaMixer† [9]	300	6	48.0	67.0	52.4	30.0	51.2	63.7	201	17.6
ASAG-A (Ours)	102	1	45.3	63.3	48.9	27.3	48.5	59.7	131	28.9
ASAG-A (Ours)	312	1	46.3	65.1	50.3	29.9	49.2	59.6	139	27.9
ASAG-A (Ours)†	296	1	47.5	66.1	51.2	30.4	50.6	62.6	206	21.3

Table 4: Comparison with six-decoder-layer detectors using 36 epochs and R50. *: Reimplement by us using official codes. †: using R101. ‡: training with 50 epochs.

Anchor Generator alleviates the feature discrepancy caused by predicting from grids, ASAG-D outperforms Efficient DETR [36] by 0.7 AP and ASAG-S outperforms Featurized Query RCNN [41] by 2.6 AP, as shown in Table 3, showing the effectiveness of ASAG. Besides, different from computing densely on large feature maps, ASAG sparsely selects patches on different feature maps, saving much computation. Further, since the patch on P_6 is the whole image, ASAG enjoys global receptive fields, bringing about much higher AP_l compared with dense(grid)-initialized ones.

Comparison with six-decoder-layer detectors with the same decoder type. As shown in Table 4, while existing one-decoder-layer detectors with dense initialization fall behind their six-decoder-layer counterparts by a large margin, our model greatly narrows the performance gap. In particular, our ASAG-S even outperforms Sparse RCNN [31] in the 100 queries setting with faster speed and fewer FLOPs. More comparisons with other well-known detectors can be found in supplementary materials.

5.3. Ablation Studies

We conduct the following ablation studies with ASAG-A with R50, 100 queries, and $1\times$ training schedule due to its fast convergence speed.

Dynamic #Query	Query Weighting	Auxiliary Head	Replace Anchor	AP	AP_{50}	AP_{75}	AP_s	AP_m	AP_l	#An
			✓	36.9	54.6	39.6	21.3	39.7	49.5	100
✓			✓	38.9	56.7	42.1	22.7	41.7	51.5	102
✓	✓		✓	41.6	59.2	44.9	24.5	44.3	55.6	103
✓	✓	✓	✓	42.6	60.5	45.8	25.9	45.8	56.9	107
✓	✓	✓		42.4	60.2	46.0	25.2	45.5	56.9	121

(a) Ablation studies on each component. Replace Anchor denotes replace the selected anchors with newly generated ones in Adaptive Probing.

η_h	η_l	AP	AP_{50}	AP_{75}	AP_s	AP_m	AP_l	#An	N_{P4}	N_{P3}
0.7	0.3	41.4	58.9	44.7	22.9	45.0	57.0	99	1.3	0.6
	0.2	41.9	59.7	45.3	24.8	45.2	56.9	104	2.2	1.6
	0.1	42.6	60.5	45.8	25.9	45.8	56.9	107	4.9	4.5
	0.05	42.6	60.6	46.1	25.9	45.8	56.8	124	9.9	10.9
0.8	0.1	42.5	60.4	45.8	25.9	45.7	56.9	107	4.9	4.5
0.7		42.6	60.5	45.8	25.9	45.8	56.9	107	4.9	4.5
0.6		42.5	60.4	45.8	25.9	45.7	56.9	107	4.9	4.5
0.4		42.5	60.3	45.7	25.2	45.8	56.9	108	4.9	4.5

(b) Ablation studies on **Confidence Threshold** for Adaptive Probing.

Size	AP	AP_{50}	AP_{75}	AP_s	AP_m	AP_l	#An	N_{P4}	N_{P3}
10	42.1	59.8	45.6	24.8	44.7	56.6	106	4.4	3.6
12	42.3	60.1	45.9	25.3	45.2	56.8	108	4.9	4.3
15	42.6	60.5	45.8	25.9	45.8	56.9	107	4.9	4.5
18	42.2	60.1	45.7	24.6	45.3	56.5	105	4.8	4.6

(c) Ablation studies on **Patch Size** for Adaptive Probing.

η_{iou}	AP	AP_{50}	AP_{75}	AP_s	AP_m	AP_l	#An	N_{P4}	N_{P3}
0.4	42.5	60.9	45.5	25.3	45.4	57.2	117	7.4	7.3
0.25	42.6	60.5	45.8	25.9	45.8	56.9	107	4.9	4.5
0.2	42.4	60.3	45.6	25.3	45.7	56.8	104	4.3	3.9
0.1	42.2	59.9	45.7	24.7	45.8	56.9	98	3.3	2.9

(d) Ablation studies on **NMS Threshold** for Adaptive Probing.

Method	AR^{1000}_{100}	AR^{1000}_{300}
RPN [27]	-	61.93
CF-QRCNN [41]	57.31	63.42
Dynamic Anchors	45.22	50.21
Dynamic Proposals	59.28	64.90

(e) Comparison on **AR** with the 3× recipe.

Table 5: Experiment results of ASAG-A with the 1× training recipe and 100 anchors on COCO `val` except for AR. The settings in our default model are colored in gray. #An denotes the average number of dynamic anchors. N_{P4} and N_{P3} denote the number of patches used in Adaptive Probing on corresponding feature maps.

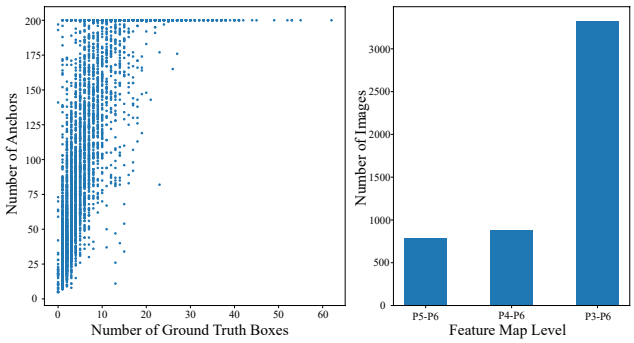

Figure 6: Statistic for Anchor Generator. **Left**: Correlation between the number of ground truth boxes and the generated anchors. **Right**: Histogram of used feature map level in Anchor Generator and the number of images.

Main components. In Table 5a, we ablate the main components newly introduced in our model. First, since anchors predicted from different feature maps using different predictors can not be treated equally, using dynamic number of anchors rather than fixed topk anchors based on scores is more appropriate, adding 2.0 AP. From the last row, we also find that preserving the selected anchors in Adaptive Probing brings no help but with more anchors, showing that the generated ones on larger feature maps are better than the selected ones and anchors cannot be sorted by scores equally. Second, Query Weighting stabilizes training and brings 2.7 AP gains. Further, providing adequate supervision signals is crucial for sparse detectors and using extra auxiliary parallel one-layer decoders increases 1.0 AP.

Patch size for Adaptive Probing. Since patches cropped on different feature maps predict anchors independently, patch sizes for each level can be different. For simplicity, we use the same size for each level in Adaptive Probing and share the MLP predictor. In Table 5c, large and small patch sizes are both inappropriate for small objects. For small patch sizes, fewer anchors are selected in Adaptive Probing due to length constraints, resulting in fewer patches. And relatively larger context is needed to detect small and middle objects, similar to the findings in [6]. As for large patch sizes, the predictor overlooks some small objects since we predict sparsely and without using spatial priors.

Confidence threshold for Adaptive Probing. In the upper part of Table 5b, relatively high η_l ignores some possible small objects thus resulting in fewer anchors and patches and poor performance on small objects. Threshold η_l lower than 0.1 is not necessary since low confidence anchors tend to be noisy anchors. The lower part shows Adaptive Probing is robust to η_h. Note that Adaptive Probing will not affect AP_l.

NMS threshold for Adaptive Probing. To reduce redundancy, we use NMS to reduce overlapped patches and the threshold affects the number of patches a lot. In Table 5d, a low threshold reduces some useful patches mistakenly.

Comparison on AR. To validate the quality of our dynamic anchors and dynamic proposals, in Table 5e, we compare AR^{1000} with well-known RPN [27] and QGN used in [41]. AR^{1000}_{100} means AR^{1000} under the 100 queries setting. Although our model lacks dense priors, like anchor boxes or anchor points, dynamic proposals still outperform RPN and QGN in terms of AR^{1000}. And our Anchor Generator is

η_h	η_l	AP(↑)	mMR(↓)	R(↑)	η_{iou}	AP(↑)	mMR(↓)	R(↑)
0.7	0.3	90.8	44.0	96.4	0.4	90.6	43.7	96.0
	0.2	91.2	43.7	96.7	0.25	91.3	43.5	96.9
	0.1	91.3	43.5	96.9	0.2	91.2	43.8	96.8
	0.05	91.4	43.5	96.9	0.1	90.2	44.4	95.6
0.8	0.1	91.3	43.5	96.9	Sparse* RCNN	89.2	48.3	95.9
0.7		91.3	43.5	96.9				
0.6		91.3	43.5	96.9	Deformable* DETR	86.7	54.0	92.5
0.5		91.2	43.6	96.7				

Table 6: CrowdHuman results on different **Confidence Threshold**s and **NMS Threshold**s for Adaptive Probing using ASAG-S. Rows in gray denote the default settings on COCO. *: Results are taken from Sparse RCNN[31].

lightweight using only 0.06 GFLOPs.

Distribution of the number of dynamic anchors. As shown in the left part of Figure 6, there is a clear positive correlation between the number of ground truth boxes and the generated anchors, showing that the Anchor Generator is adaptive to different images by generating more queries for difficult images and vice versa.

Distribution of the number of used feature map levels. Our Adaptive Probing method enjoys the early-stop mechanism. As shown in the right part of Figure 6, roughly 40% of the images in the validation set do not use all the feature maps, saving some computation.

Robustness of hyper-parameters in Adaptive Probing. In Adaptive Probing, we only select anchors whose confidences fall in $[\eta_l, \eta_h]$ to crop patches in the larger feature maps and the patches are filtered by NMS with IoU threshold η_{iou} for reducing redundancy. To show the robustness of these hyper-parameters, we follow Sparse RCNN [31] to conduct experiments on CrowdHuman [30] dataset, which is significantly different from COCO. Following Sparse RCNN, we run ASAG-S with 50 epochs and the average number of anchors within 500. As shown in Table 6, the default hyper-parameters of Adaptive Probing on COCO still work on CrowdHuman. And ASAG-S outperforms Sparse RCNN and Deformable DETR by a large margin.

5.4. Visualization

Comparison on feature maps. In this work, we provide a sparse way to initialize object queries by using patches as the prediction units thus alleviating the feature map discrepancy caused by predicting on grids. As shown in Figure 7, feature maps from our method are more similar to six-decoder-layer sparse detectors, which activate objects in an adaptive way rather than uniformly.

Visualization for Adaptive Probing. We visualize some results to understand how Anchor Generator works. More pictures will be displayed in the supplementary materials. In Figure 8, we draw dynamic anchors in white and patches in red. As shown in the first column, the anchors for different images are different and have covered most foreground objects, showing them truly adaptive and precise. Anchor Generator tends to generate more anchors for small objects through Adaptive Probing, which greatly enhances the ability to detect small objects. Although the red boxes, *i.e.* patches, sparsely locate on the images, they precisely cover small objects, such as the things on the table and the spectators in the stands, greatly increasing the recall rate.

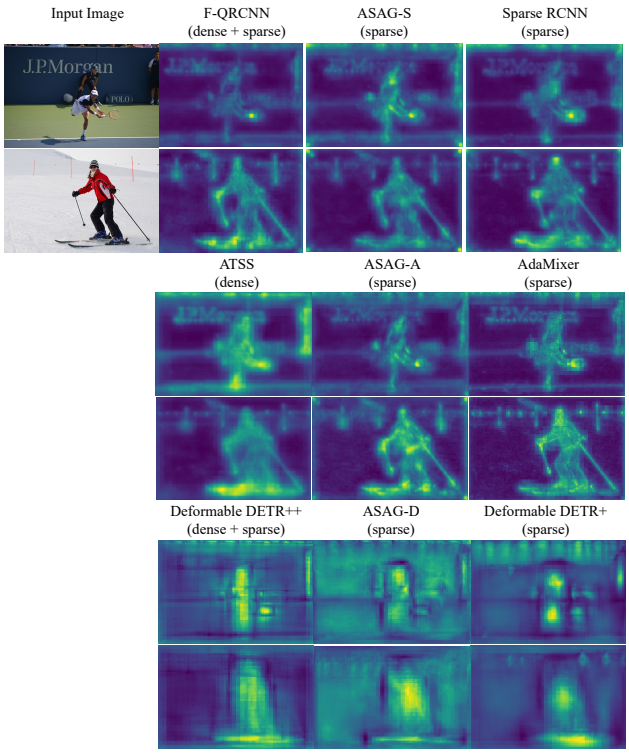

Figure 7: Visualization of feature maps of different methods. Our models with sparse initialization are more consistent with sparse decoders.

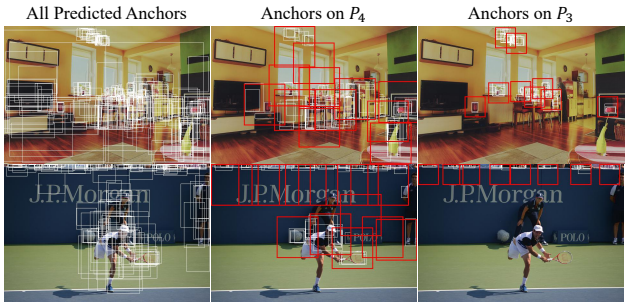

Figure 8: Visualization of dynamic anchors and Adaptive Probing. The white and red boxes represent anchors and patches, respectively.

Detector	Query Weighting	AP	AP_{50}	AP_{75}	AP_s	AP_m	AP_l
AdaMixer [9]		39.2	47.7	42.1	21.5	42.2	55.3
	✓	42.7	61.5	45.9	24.7	45.4	59.2

Table 7: Query Weighting for six-decoder-layer detectors.

Detector	#L	AP	AP_{50}	AP_{75}	AP_s	AP_m	AP_l
ASAG-A	1	42.6	60.5	45.8	25.9	45.8	56.9
ASAG-A	2	43.2	61.6	46.9	25.6	46.4	59.0
AdaMixer	6	42.7	61.5	45.9	24.7	45.4	59.2

Table 8: Effect of different number of decoder layers.

5.5. Discussion

Is Query Weighting beneficial to six-decoder-layer detectors? Most six-decoder-layer detectors and dense ones detect objects from image-agnostic and fixed queries (or anchors). The quality of initial queries can not be improved during the training process. Thus each decoder layer should be trained under difference IoU level [1] and Query Weighting can not benefit them, as shown in Table 7.

Is it equal to increase the same number of anchors for one- and six-decoder-layer detectors? Since six-decoder-layer detectors work well with image-agnostic and fixed queries, query initialization brings minor help to them. But more queries for them narrow the search space for each query. However, query initialization is vital to one-decoder-layer detectors and directly affects their performance [36]. As shown in Table 5e, increasing queries from 100 to 300 only brings 5.6 AR since the top100 proposals already lie in the top300 ones. Thus the benefit of increasing queries for one-decoder-layer detectors is less than for six-decoder-layer ones, shown in Table 2 and Table 4. How to scale up one-decoder-layer detectors needs future work.

More analysis on AP_l. A core advantage of DETR-like detectors is that they reason globally so that the AP_l is significantly higher than traditional detectors. However, although our ASAGs are comparable on AP with corresponding six-decoder-layer counterparts, the AP_l still lags behind them, as shown in Table 4. And it seems that the AP_l of ASAGs cannot increase when using more queries just as the counterparts. Here we provide some analysis.

The AP_l is affected by two factors: the number of decoder layers and query initialization. Regarding the number of decoder layers, we run ASAG-A with two decoder layers in Table 8 and get 43.2AP and 59.0AP_l (vs one layer 42.6AP and 56.9AP_l), in which AP_l is already comparable with AdaMixer. As for query initialization, ASAG views the whole image in P_6 as a patch and predicts anchors from it, enjoying global receptive fields during initialization, and thus shows superiority to dense-initialized one-decoder-layer detectors on AP_l in Table 3.

Besides, the phenomenon that AP_l does not improve with more queries also occurred in another one-decoder-layer detector (see F-QRCNN in Table 3), showing it is not specific to ASAG. Since one-decoder-layer detectors use image-specific queries and large objects are relatively easy to detect, the initial queries are accurate enough, and increasing queries brings a little recall for large objects. In contrast, queries of six-decoder-layer detectors are randomly initialized. Thus, the gains on AP_l for one-decoder-layer detectors are smaller than six-decoder-layer counterparts when using more queries.

6. Conclusion

In this work, we find that dense initialization is not optimal for one-decoder-layer sparse detectors since predicting on grids leads to feature conflict, which hampers their performance. To tackle this problem, we propose ASAG and predict dynamic anchors based on patches in a sparse way, thus alleviating feature conflict. We further design Adaptive Probing to generate patches on different levels, which is adaptive to the number of used feature maps, the number of patches, and the locations of patches. Finally, simple but effective Query Weighting stabilizes the training. With the novel design, our dynamic anchors and proposals are better than dense ones without using predefined spatial priors. Experiments show that we greatly increase the performance of one-decoder-layer detectors and narrow the performance gap while retaining the fast speed.

Acknowledgments. This work was supported partially by the NSFC (U21A20471, U1911401, 62072482), Guangdong NSF Project (No. 2023B1515040025, 2020B1515120085).

References

[1] Zhaowei Cai and Nuno Vasconcelos. Cascade r-cnn: Delving into high quality object detection. In *CVPR*, 2018. 9

[2] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *ECCV*, 2020. 1, 2

[3] Jiahao Chang, Shuo Wang, Guangkai Xu, Zehui Chen, Chenhongyi Yang, and Feng Zhao. Detrdistill: A universal knowledge distillation framework for detr-families. *arXiv preprint arXiv:2211.10156*, 2022. 2, 3

[4] Qiang Chen, Xiaokang Chen, Gang Zeng, and Jingdong Wang. Group detr: Fast training convergence with decoupled one-to-many label assignment. *arXiv preprint arXiv:2207.13085*, 2022. 2, 3

[5] Xiaokang Chen, Jiahui Chen, Yan Liu, and Gang Zeng. D ˜3 etr: Decoder distillation for detection transformer. *arXiv preprint arXiv:2211.09768*, 2022. 2

[6] Yukang Chen, Yanwei Li, Tao Kong, Lu Qi, Ruihang Chu, Lei Li, and Jiaya Jia. Scale-aware automatic augmentation for object detection. In *CVPR*, 2021. 7

[7] Chengjian Feng, Yujie Zhong, Yu Gao, Matthew R Scott, and Weilin Huang. Tood: Task-aligned one-stage object detection. In *ICCV*, 2021. 5

[8] Peng Gao, Minghang Zheng, Xiaogang Wang, Jifeng Dai, and Hongsheng Li. Fast convergence of detr with spatially modulated co-attention. In *ICCV*, 2021. 2

[9] Ziteng Gao, Limin Wang, Bing Han, and Sheng Guo. Adamixer: A fast-converging query-based object detector. In *CVPR*, 2022. 1, 2, 3, 5, 6, 9

[10] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *ICCV*, 2017. 1, 3

[11] Qinghang Hong, Fengming Liu, Dong Li, Ji Liu, Lu Tian, and Yi Shan. Dynamic sparse r-cnn. In *CVPR*, 2022. 2

[12] Linjiang Huang, Kaixin Lu, Guanglu Song, Liang Wang, Si Liu, Yu Liu, and Hongsheng Li. Teach-detr: Better training detr with teachers. *arXiv preprint arXiv:2211.11953*, 2022. 2

[13] Ding Jia, Yuhui Yuan, Haodi He, Xiaopei Wu, Haojun Yu, Weihong Lin, Lei Sun, Chao Zhang, and Han Hu. Detrs with hybrid matching. *arXiv preprint arXiv:2207.13080*, 2022. 2, 3

[14] Ding Jia, Yuhui Yuan, Haodi He, Xiaopei Wu, Haojun Yu, Weihong Lin, Lei Sun, Chao Zhang, and Han Hu. Detrs with hybrid matching. *arXiv preprint arXiv:2207.13080*, 2022. 2, 3

[15] Alexander Kirillov, Yuxin Wu, Kaiming He, and Ross Girshick. Pointrend: Image segmentation as rendering. In *CVPR*, 2020. 5

[16] Feng Li, Hao Zhang, Shilong Liu, Jian Guo, Lionel M Ni, and Lei Zhang. Dn-detr: Accelerate detr training by introducing query denoising. In *CVPR*, 2022. 2

[17] Shuai Li, Chenhang He, Ruihuang Li, and Lei Zhang. A dual weighting label assignment scheme for object detection. In *CVPR*, 2022. 5, 6

[18] Xiang Li, Wenhai Wang, Lijun Wu, Shuo Chen, Xiaolin Hu, Jun Li, Jinhui Tang, and Jian Yang. Generalized focal loss: Learning qualified and distributed bounding boxes for dense object detection. In *NeurIPS*, 2020. 4, 5

[19] Yunsheng Li, Yinpeng Chen, Xiyang Dai, Dongdong Chen, Mengchen Liu, Pei Yu, Ying Jin, Lu Yuan, Zicheng Liu, and Nuno Vasconcelos. Should all proposals be treated equally in object detection? In *ECCV*, 2022. 2

[20] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *CVPR*, 2017. 1

[21] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *ICCV*, 2017. 1

[22] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *ECCV*, 2014. 2, 5

[23] Shilong Liu, Feng Li, Hao Zhang, Xiao Yang, Xianbiao Qi, Hang Su, Jun Zhu, and Lei Zhang. Dab-detr: Dynamic anchor boxes are better queries for detr. In *ICLR*, 2022. 2

[24] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *ICLR*, 2019. 6

[25] Depu Meng, Xiaokang Chen, Zejia Fan, Gang Zeng, Houqiang Li, Yuhui Yuan, Lei Sun, and Jingdong Wang. Conditional detr for fast training convergence. In *ICCV*, 2021. 2

[26] Jeffrey Ouyang-Zhang, Jang Hyun Cho, Xingyi Zhou, and Philipp Krähenbühl. Nms strikes back. *arXiv preprint arXiv:2212.06137*, 2022. 2, 3

[27] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *NeurIPS*, 2015. 1, 7

[28] Hamid Rezatofighi, Nathan Tsoi, JunYoung Gwak, Amir Sadeghian, Ian Reid, and Silvio Savarese. Generalized intersection over union: A metric and a loss for bounding box regression. In *CVPR*, 2019. 6

[29] Byungseok Roh, JaeWoong Shin, Wuhyun Shin, and Saehoon Kim. Sparse detr: Efficient end-to-end object detection with learnable sparsity. In *ICLR*, 2022. 2

[30] Shuai Shao, Zijian Zhao, Boxun Li, Tete Xiao, Gang Yu, Xiangyu Zhang, and Jian Sun. Crowdhuman: A benchmark for detecting human in a crowd. *arXiv preprint arXiv:1805.00123*, 2018. 8

[31] Peize Sun, Rufeng Zhang, Yi Jiang, Tao Kong, Chenfeng Xu, Wei Zhan, Masayoshi Tomizuka, Lei Li, Zehuan Yuan, Changhu Wang, et al. Sparse r-cnn: End-to-end object detection with learnable proposals. In *CVPR*, 2021. 1, 2, 3, 5, 6, 8

[32] Zhi Tian, Chunhua Shen, Hao Chen, and Tong He. Fcos: Fully convolutional one-stage object detection. In *ICCV*, 2019. 1

[33] Tao Wang, Li Yuan, Yunpeng Chen, Jiashi Feng, and Shuicheng Yan. Pnp-detr: towards efficient visual analysis with transformers. In *ICCV*, 2021. 2

[34] Yingming Wang, Xiangyu Zhang, Tong Yang, and Jian Sun. Anchor detr: Query design for transformer-based detector. In *AAAI*, 2022. 2, 6

[35] Chenhongyi Yang, Zehao Huang, and Naiyan Wang. Querydet: Cascaded sparse query for accelerating high-resolution small object detection. In *CVPR*, 2022. 4, 5

[36] Zhuyu Yao, Jiangbo Ai, Boxun Li, and Chi Zhang. Efficient detr: improving end-to-end object detector with dense prior. *arXiv preprint arXiv:2104.01318*, 2021. 1, 3, 5, 6, 9

[37] Gongjie Zhang, Zhipeng Luo, Yingchen Yu, Kaiwen Cui, and Shijian Lu. Accelerating detr convergence via semantic-aligned matching. In *CVPR*, 2022. 2

[38] Gongjie Zhang, Zhipeng Luo, Yingchen Yu, Jiaxing Huang, Kaiwen Cui, Shijian Lu, and Eric P Xing. Semantic-aligned matching for enhanced detr convergence and multi-scale feature fusion. *arXiv preprint arXiv:2207.14172*, 2022. 2

[39] Hao Zhang, Feng Li, Shilong Liu, Lei Zhang, Hang Su, Jun Zhu, Lionel M Ni, and Heung-Yeung Shum. Dino: Detr with improved denoising anchor boxes for end-to-end object detection. *arXiv preprint arXiv:2203.03605*, 2022. 2

[40] Shifeng Zhang, Cheng Chi, Yongqiang Yao, Zhen Lei, and Stan Z Li. Bridging the gap between anchor-based and anchor-free detection via adaptive training sample selection. In *CVPR*, 2020. 1

[41] Wenqiang Zhang, Tianheng Cheng, Xinggang Wang, Qian Zhang, and Wenyu Liu. Featurized query r-cnn. *arXiv preprint arXiv:2206.06258*, 2022. 1, 2, 3, 5, 6, 7

[42] Minghang Zheng, Peng Gao, Renrui Zhang, Kunchang Li, Xiaogang Wang, Hongsheng Li, and Hao Dong. End-to-end object detection with adaptive clustering transformer. In *BMVC*, 2021. 2

[43] Xizhou Zhu, Weijie Su, Lewei Lu, Bin Li, Xiaogang Wang, and Jifeng Dai. Deformable detr: Deformable transformers for end-to-end object detection. In *ICLR*, 2021. 1, 2, 3, 5, 6

[44] Zhuofan Zong, Guanglu Song, and Yu Liu. Detrs with collaborative hybrid assignments training. *arXiv preprint arXiv:2211.12860*, 2022. 2, 3