SPARSE DETR: EFFICIENT END-TO-END OBJECT DETECTION WITH LEARNABLE SPARSITY

Anonymous authors
Paper under double-blind review

ABSTRACT

DETR is the first end-to-end object detector using a transformer encoder-decoder architecture and demonstrates competitive performance but low computational efficiency. The subsequent work, Deformable DETR, enhances the efficiency of DETR by replacing dense attention with deformable attention, which achieves $10\times$ faster convergence and improved performance. Using the multiscale feature to ameliorate performance, however, the number of encoder queries increases by $20\times$ compared to DETR, and the computation cost of the encoder attention remains a bottleneck. We observe that the encoder queries referenced by the decoder account for only 45% of the total, and find out the detection accuracy does not deteriorate significantly even if only the referenced queries are polished in the encoder block. Inspired by this observation, we propose `Sparse DETR` that selectively updates only the queries expected to be referenced by the decoder, thus help the model effectively detect objects. In addition, we show that applying an auxiliary detection loss on the selected queries in the encoder improves the performance while minimizing computational overhead. We validate that `Sparse DETR` achieves better performance than Deformable DETR even with only 10% encoder queries on the COCO dataset. Albeit only the encoder queries are sparsified, the total computation cost decreases by 38% and the frames per second (FPS) increases by 42% compared to Deformable DETR. Code will be released.

1 INTRODUCTION

In recent years, we have witnessed the dramatic advancement and the success of object detection in deep learning. Diverse object detection methods have been proposed, but the existing algorithms that perform positive matching with the ground truth as a heuristic way require non-maximum suppression (NMS) post-processing of near-duplicate predictions. Recently, Carion et al. (2020) has introduced a fully end-to-end detector DETR by eliminating the need for NMS post-processing through a set-based objective. The training objective is designed by employing the Hungarian algorithm that considers both classification and regression costs, and achieves highly competitive performance. However, DETR is unable to use multi-scale features such as feature pyramid networks (Lin et al., 2017), which are commonly used in object detection to improve the detection of small objects. The main reason is increased memory usage and computation by adding Transformer (Vaswani et al., 2017) architecture. As a result, its ability to detect small objects is relatively poor.

To address this problem, Zhu et al. (2021) has proposed a deformable-attention inspired by the deformable convolution (Dai et al., 2017) and reduced the quadratic complexity to linear complexity through key sparsification in the attention module. By using deformable attention, deformable DETR addresses the slow convergence and high complexity issue of DETR, which enables the encoder to use multi-scale features as an input and significantly improves performance on detecting small objects. However, using the multi-scale features as an encoder input increases the number of tokens to be processed by about 20 times. Eventually, despite efficient computation for the same token length, the overall complexity increases back again, making the model inference slower even than vanilla DETR.

In general, natural images often contain large background regions irrelevant to the objects of interest, and accordingly, in end-to-end detectors, the tokens corresponding to the background also occupy a significant portion. In addition, the importance of each regional feature is not identical, which has

been proven by the two-stage detectors successfully do their job by focusing only on the foreground. It suggests that there exists considerable regional redundancy that can be reduced in the detection tasks and seeking to devise an efficient detector focusing on the salient regions is a necessary and natural direction. In this work, we observe that the encoder queries referenced by the decoder, using the trained Deformable DETR model, accounts for only about 45% of the total, and find out polishing only the part of the encoder queries referenced by the decoder's object queries does not significantly degrade the detection performance (0.1 AP degradation on COCO dataset). Inspired by this observation, we propose a learnable decoder cross-attention map predictor to sparsify encoder queries. In the existing methods (Carion et al., 2020; Zhu et al., 2021), the encoder takes all the queries, i.e. the backbone features combined with corresponding positional embeddings, as input without discrimination. Meanwhile, our approach distinguishes encoder queries to be referenced later in the decoder and considers only those queries in self-attention. Therefore, this can significantly reduce the number of encoder queries involved in the computation and reduce the total computational cost for detection. We further propose the encoder auxiliary loss only for selected encoder queries to improve detection performance while minimizing computational overhead. The proposed auxiliary loss not only improves performance, but also allows training a larger number of encoder layers.

Extensive experiments on the COCO 2017 benchmark (Lin et al., 2014) demonstrate that Sparse DETR effectively reduces computational cost while achieving better detection performance. Without bells and whistles, Sparse DETR using Swin-T (Liu et al., 2021) backbone achieves 48.2 AP with 38% reduction of the entire computational cost compared to the 48.0 AP baseline and 49.2 AP with 23% reduction. In the case of the experiment that achieves 48.2 AP using only 10% of encoder queries, the computational cost of the transformer encoder block is reduced by approximately 82%.

We summarize our contributions as follows:

- We propose *encoder query sparsification* method for an efficient end-to-end object detector, by which we lighten the attention complexity in the encoder. This efficiency enables stacking more encoder layers than Deformable DETR, leading to performance improvement within the same computational budget.
- We propose two novel sparsification criteria to sample the informative subset from the entire query set: *Objectness Score (OS)* and *Decoder cross-Attention Map (DAM)*. Based on the decoder cross-attention map criterion, the sparsified model preserves detection performance even when using only 10% of the whole queries.
- We adopt an *encoder auxiliary loss* only for the selected queries. This additional loss not only stabilizes the learning process, but also greatly improves performance, with only marginally increased training time.

2 Related Work

Efficient computation in vision transformers. It is a well-known problem that the attention computation in Transformers incurs the high time and memory complexity. The vision transformers need to digest even bigger token sets as input so that a large body of works (Parmar et al., 2018; Child et al., 2019; Ho et al., 2019; Wang et al., 2020; Katharopoulos et al., 2020; Choromanski et al., 2021; Kitaev et al., 2020) has been proposed lightweight attention mechanisms for them. Most of those works shed light on the complexity that resides only in a single-scale attention module, which hinders direct extension to the multi-scale features generally required in object detection.

One of the promising approaches for the lighter transformer attention is input-dependent *token* sparsification. DynamicViT (Rao et al., 2021) and IA-RED^2 (Pan et al., 2021), similar to our work, both propose jointly-learned token selectors generating the sparsity patterns to be overlaid on the input tokens. Those approaches mainly focus on sparsifying a backbone network evaluated on the classification tasks, while our interest lies in a sparse encoder of the end-to-end object detectors.

On the other hand, there has been a line of works sharing the spirit with ours in that they aim at sparse transformers in the DETR-based framework. Deformable DETR (Zhu et al., 2021) conducts sparse attention computation by sampling only a fraction of the entire key set with learnable 2-d offsets, which enables to use multi-scale feature maps with a reasonable computational cost. It can be viewed as a *key* sparsification method but with dense queries, while our approach further reduces the

query set pursuing even more sparsity. PnP-DETR (Wang et al., 2021) shortens the token length of the transformer encoder by introducing the Polling and Pull (PnP) module to sample the foreground tokens and condense the background tokens into a smaller set. However, their method cannot naively be integrated with Deformable DETR, since their sparsification breaks the 2d spatial structure of the key set assumed in the deformable attention. On the contrary, Sparse DETR preserves the 2d sample space of the set and can be seamlessly combined with the deformable attention, which facilitates handling the multi-scale features. Thus, our approach gets benefits from *both* the deformable *key* sampling and the proposed *query* sparsification. Most of all, we propose the explicit objectives for the token selection network, whereas the aforementioned works have no explicit objective implying their beliefs in a *good* selection strategy, merely relying on the final detection objective.

Auxiliary Loss. Auxiliary loss (Szegedy et al., 2015) is widely adopted to deliver gradients to the early layers of deep networks. DETR variants employ auxiliary Hungarian matching objectives at the end of every decoder layer with extra FFN heads so that each decoder layer directly learns to detect the correct number of objects out of the decoder's outputs. Unlike the decoder's queries whose number is relatively small(e.g. 300), the number of encoder's queries has much larger scales when using multi-scale features. Thus, extending the layerwise auxiliary loss to the multi-scale encoder increases the training time cost by feeding too many tokens to the attached FFN heads. In Sparse DETR, thanks to the sparsity already induced in the encoder, we can instantly economize that cost while enjoying the auxiliary gradients in a wider range of intermediate layers.

3 Approach

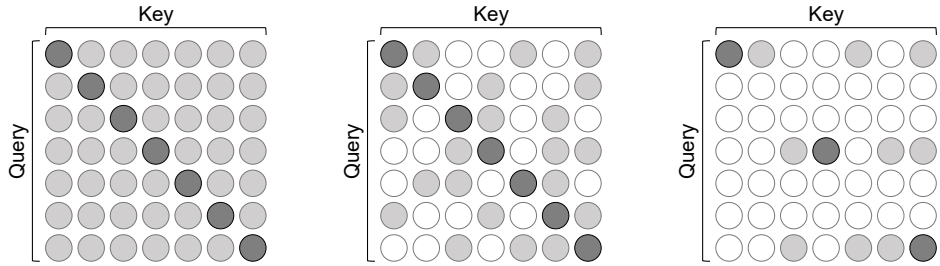
In this section, we present our main contributions: (a) formulating a generalized saliency-based query sparsification scheme for the encoder, (b) proposing the effective saliency criteria with which that scheme can practically work, and (c) employing the top-k decoder query selection and encoder auxiliary losses to improve the performance. Before describing the details, we revisit the key components of DETR (Carion et al., 2020) and Deformable DETR (Zhu et al., 2021).

3.1 Preliminary

DETR. DETR takes the flattened spatial feature map $\mathbf{x}_{\text{bb}} \in \mathbb{R}^{N \times D}$ from a backbone network into the transformer encoder, where N denotes the number of queries (i.e. features) and D denotes query dimension. The encoder iteratively polishes $\mathbf{x}_{\text{bb}}$ by several vanilla self-attention modules. Then, the transformer decoder takes both the refined queries and M learnable object queries $\{q_i\}_{i=1 \cdots M}$ as inputs and predicts a tuple of a class score $\mathbf{c} \in [0,1]^C$ and a bounding box $\mathbf{b} \in [0,1]^4$ for each object query q_i, denoted as $\{\hat{\mathbf{y}}_i\} = \{(\mathbf{c}_i, \mathbf{b}_i)\}$, where C denotes the number of classes. All components including the backbone network are jointly trained by performing the bipartite matching between the ground truth $\{\mathbf{y}_i\}$ and predictions $\{\hat{\mathbf{y}}_i\}$.

Deformable DETR. Deformable DETR replaces the vanilla dense attention, which is the main computational bottleneck in DETR, with a deformable attention module. This significantly reduces the computational cost and improves the convergence. Suppose that we have the same size of a set of queries (denoted as Ω_q) and a set of keys (denoted as Ω_k), which means $|\Omega_q| = |\Omega_k| = N$. The conventional dense attention computes the attention weight A_{qk} for every pair $\{(q,k) : q \in \Omega_q, k \in \Omega_k\}$, resulting in quadratic complexity with respect to N. Deformable attention reduces this quadratic complexity into the linear one by only considering relevant keys for each query. Specifically, deformable attention computes attention weight A_{qk} for all queries and a small set of keys: $\{(q,k) : q \in \Omega_q, k \in \Omega_{qk}\}$, where $\Omega_{qk} \subset \Omega_k$ and $|\Omega_{qk}| = K \ll N$.

Due to this key sparsification, Deformable DETR is able to use the multi-scale features of the backbone network, improving the detection performance of small objects significantly. Paradoxically, using the multi-scale feature increases the number of queries in the transformer encoder by about $20\times$ compared to DETR, making that the encoder becomes the computational bottleneck of deformable DETR. This motivates us to develop a sparsification method to reduce the number of queries in the encoder aggressively, which is described in the next sections.



(a) DETR (b) Deformable DETR (c) Sparse DETR

Figure 1: **Attention complexity**. The circles in the square matrix represent the attention between keys and queries. The gray/white circles correspond to preserved/removed connection respectively, and darker gray on the diagonal positions means where the token attends to itself. (a) Dense attention in DETR takes quadratic complexity. (b) Deformable DETR uses key sparsification, thus takes linear complexity. (c) Sparse DETR further uses query sparsification. Attention in Sparse DETR also takes linear complexity, but is much lighter than Deformable DETR's.

3.2 Encoder Query Sparsification

In this section, we introduce our query sparsification scheme that the encoder module selectively refines a small number of encoder queries. This encoder query subset is obtained from the backbone feature map $\mathbf{x}_{\text{bb}}$ with a certain criterion, which is described in the subsequent section. For features that are not updated in this process, the values of $\mathbf{x}_{\text{bb}}$ are passed through the encoder layers without being changed.

Formally, suppose that we have a scoring network $g : \mathbb{R}^d \to \mathbb{R}$ that measures saliency of each query in $\mathbf{x}_{\text{bb}}$. We then define ρ-salient regions Ω_s^ρ as the top-ρ% queries with the highest scores, for a given keeping ratio ρ, i.e. $S = |\Omega_s^\rho| = \rho \cdot |\Omega_q| \ll |\Omega_q| = N$. Then, the i-th encoder layer polishes the features $\mathbf{x}_{i-1}$ by:

$$\mathbf{x}_{i,j} = \begin{cases} \mathbf{x}_{i-1,j} & j \notin \Omega_s^\rho \\ \text{LN}(\text{FFN}(\mathbf{z}_{i,j}) + \mathbf{z}_{i,j}) & j \in \Omega_s^\rho, \text{ where } \mathbf{z}_{i,j} = \text{LN}(\text{DFAttn}(\mathbf{x}_{i-1,j}, \mathbf{x}_{i-1}) + \mathbf{x}_{i-1,j}), \end{cases} \tag{1}$$

where DFAttn refers to deformable attention, LN to layer normalization (Ba et al., 2016), and FFN to a feed-forward network. Even in the case of unselected queries, the values are still passed through the encoder layer, so they can be referenced when updating the selected queries. This means that the unselected queries can hand over information to the selected queries without losing the value of themselves while minimizing the computational cost. Here, we use deformable attention for refining queries in Ω_s^ρ, but the proposed encoder query sparsification is applicable regardless of which attention method the encoder uses.

Complexity of Attention Modules in Encoder. Deformable DETR reduces the attention complexity through key sparsification, and we further reduce the attention complexity through query sparsification, as shown in Fig. 1. Conventional dense attention in DETR requires quadratic complexity $O(N^2)$, where N is the query length. Deformable attention requires linear complexity $O(NK)$, where $K \ll N$ is the number of keys for each query. Sparse attention requires only $O(SK)$, where $S \ll N$ is the number of salient encoder queries.

3.3 Finding Salient Encoder Queries

In this section, we introduce how to find a salient query set Ω_s^ρ from a backbone feature $\mathbf{x}_{\text{bb}}$. We propose a method for determining saliency using a cross attention map from the transformer decoder. Before presenting our approach, we first discuss a simple yet effective method based on the objectness scores obtained from a separate detection head. The limitation of this simple approach motivates us to develop an advanced one, which is described in the following paragraph.

Objectness Score. Measuring objectness per each query is very natural to determine which ones from a backbone feature should be further updated in the transformer encoder. It is widely known

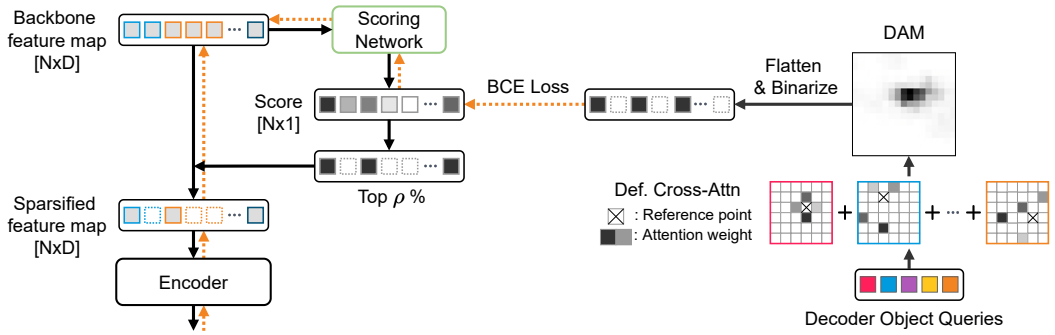


Figure 2: Illustration on how to learn a scoring network by predicting binarized Decoder cross-Attention Map (DAM), where a dashed orange arrow means a backpropagation path. The left part shows the forward/backward passes in the encoder, and the right one presents how to construct DAM for learning the scoring network. See **Appendix A.1** for implementation details of the scoring network.

that feature map from a pretrained backbone network is able to find the saliency of objects, which is why the region proposal network (RPN) has been successfully adopted in many object detectors (Ren et al., 2015; Dai et al., 2016; He et al., 2017). Inspired by this observation, we introduce an additional detection head and Hungarian loss to the backbone feature map, where the structure of the newly added head is the same as the one of the final detection head. Then, we can select the top-$\rho\%$ encoder queries with the highest class scores as a salient query set Ω_s^ρ. This approach is effective to sparsify encoder queries, but we believe that it is sub-optimal to the transformer decoder, because the selected queries from the separate detection head are not explicitly considered for the decoder.

Decoder Cross-Attention Map. We consider another approach to select a subset of encoder queries that are highly relevant to the decoder in a more explicit manner. We observe that the cross-attention maps from the transformer decoder could be used for measuring the saliency, because the decoder gradually attends to a subset of encoder queries that are favorable to detect objects as training continues. Motivated by this, we introduce a scoring network that predicts a *pseudo ground-truth* of the saliency defined by decoder cross-attention maps, and use it to determine which queries should be further refined on the fly. Fig. 2 summarizes how to train a scoring network, and details are presented below.

To determine the saliency of each query of $\mathbf{x}_{\text{bb}}$, we have to aggregate the decoder cross-attentions between all object queries and the encoder query. This procedure produces a single map of the same size as the feature map from the backbone, which is defined as Decoder cross-Attention Map (DAM). In the case of the dense attention, DAM can be easily obtained by summing up attention maps from every decoder layer. In case of deformable attention, for each encoder query, the corresponding value of DAM can be obtained by accumulating the attention weights of decoder queries whose attention offsets are directed toward the encoder query. Refer to the **Appendix A.2** for the details in the DAM creation.

To train the scoring network, we binarize DAM so that the top-$\rho\%$ (by attention weights) of queries is only retained. This is because our goal is to find a small subset of queries that the decoder references the most, rather than precisely predicting how much each encoder query will be referenced by the decoder. This binarized DAM implies the one-hot target that indicates whether each query is included in the top-$\rho\%$ most referenced queries. Then, we consider a 4-layer scoring network g to predict how likely a given query is included in the top-ρ % most referenced queries, and the network is trained by minimizing the binary cross entropy (BCE) loss between the binarized DAM and prediction:

$$\mathcal{L}_{dam} = -\frac{1}{N}\sum_{i=1}^{N} \text{BCE}(g(\mathbf{x}_{\text{bb}})_i, \text{DAM}_i^{\text{bin}}), \tag{2}$$

where $\text{DAM}_i^{\text{bin}}$ means the binarized DAM value of the ith encoder query.

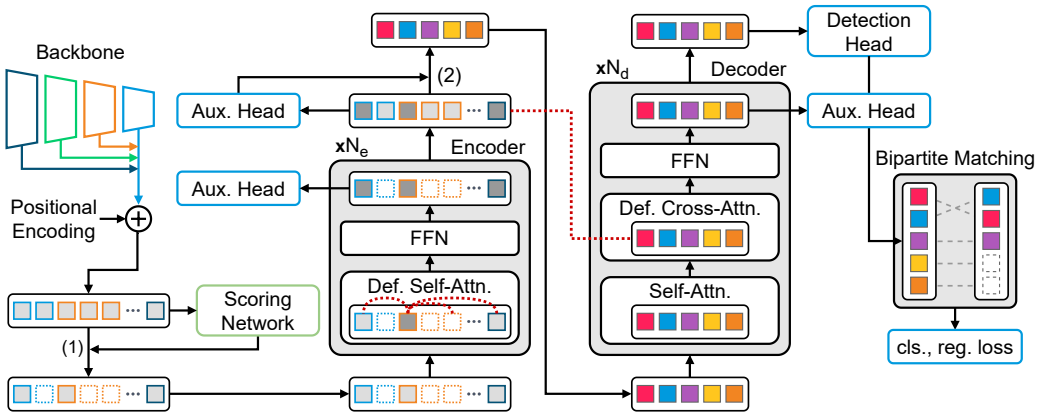


Figure 3: **Sparse DETR architecture**. Sparse DETR introduces three additional components: (a) the scoring network, (b) auxiliary heads in the encoder, and (c) the auxiliary head to select the top-k queries for the decoder. Sparse DETR measures the saliency of encoder queries by using the scoring network, and selects the top-$\rho\%$ queries, which is referred to as (1) in the diagram. After refining only the selected queries in the encoder blocks, the auxiliary head selects the top-k elements from the encoder output, which is served as the decoder queries. This process is referred to as (2) in the diagram. In addition, we remark that additional auxiliary heads in each encoder block play a key role in achieving improved performance. Only sparsified encoder queries are passed to the encoder auxiliary heads for efficiency. All auxiliary heads in the encoder and decoder are trained with a Hungarian loss.

One may say that since DAM in the early phase of training is not accurate, pruning out the encoder queries based on the result in the decoder degrades the final performance or hurts the convergence. However, we empirically observe that the optimization is very stable even in the early phase of training, and achieves better performance compared to the method based on objectness score. We describe detailed comparisons in the experiments section.

3.4 Additional Components

In this section, we introduce two additional components: (a) top-k decoder queries selection and (b) applying auxiliary losses on the encoder queries. We empirically observe that these greatly help improve the final performance and stabilize the optimization. The overall architecture of Sparse DETR including these components is depicted in Fig. 3.

Top-k Decoder Queries. In DETR and Deformable DETR, decoder queries are given by only learnable queries or with predicted reference points via another head after the encoder. In Efficient DETR (Yao et al., 2021), decoder takes a part of encoder output as input, similar to RoI Pooling (Ren et al., 2015). Here, an auxiliary detection head is attached to the encoder output $\mathbf{x}_{\text{enc}}$ and the head calculates the objectness (class) score of each encoder output. Based on the score, the top-k encoder outputs are passed as decoder queries, similar to objectness score-based encoder query sparsification. Since this outperforms the methods based on learnable queries or the two-stage scheme, we include this top-k decoder query selection in our final architecture.

Encoder Auxiliary Loss. In DETR, auxiliary detection heads are attached to decoder layers, but not to encoder layers. Due to a significantly larger number of encoder queries (about 18k queries) compared to decoder queries (about 300), encoder auxiliary heads will heavily increase the computational cost. In Sparse DETR, however, only part of encoder queries are refined by the encoder, and adding auxiliary heads only for sparsified encoder queries is not a big burden.

We empirically observe that applying an auxiliary detection head along with Hungarian loss on the selected queries improves the final detection performance. Following the analysis of Sun et al. (2021), we conjecture that this performance gain is obtained by the Hungarian loss, which helps distinguish the confusing features in the encoder.

4 EXPERIMENTS

We compare Sparse DETR with the conventional object detectors, including the recently proposed ones in the DETR family. In addition, we conduct an ablation study to support our claims in section 3, presenting the performance comparison between query selection criteria (OS vs. DAM), the effectiveness of the encoder auxiliary loss, and the dynamic sparsification during inference.

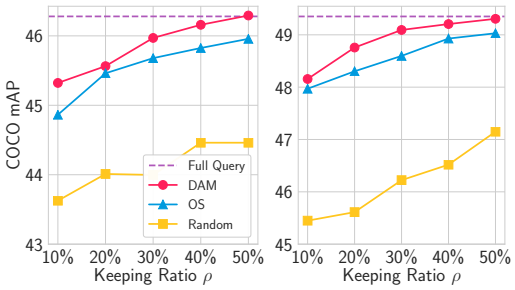
Table 1: **Detection results of Sparse DETR on COCO 2017 val set.** Top-k & BBR denotes that we sample the top-k object queries instead of using the learned object queries (Yao et al., 2021), and perform bounding box refinement in the decoder block (Zhu et al., 2021), respectively. FLOPs and FPS are measured in the same way as used in Zhu et al. (2021). The results marked by †, ‡ are the reported ones from Zhu et al. (2021) and Wang et al. (2021), respectively. The results marked by § mean that the backbone network is initialized by SCRL (Roh et al., 2021) instead of the ImageNet (Deng et al., 2009) pre-trained one, because SCRL aims to learn dense representations in a self-supervised way.

Method	Epochs	Keeping ratio (ρ)	Top-k & BBR	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L	params	FLOPs	FPS
***ResNet50** backbone:*												
F-RCNN-FPN†	109	N/A		42.0	62.1	45.5	26.6	45.4	53.4	42M	180G	26
DETR†	500	100%		42.0	62.4	44.2	20.5	45.8	61.1	41M	86G	28
DETR-DC5†	500	100%		43.3	63.1	45.9	22.5	47.3	61.1	41M	187G	12
PnP-DETR‡	500	33%		41.1	61.5	43.7	20.8	44.6	60.0	-	-	-
	500	50%		41.8	62.1	44.4	21.2	45.3	60.8	-	-	-
PnP-DETR-DC5‡	500	33%		42.7	62.8	45.1	22.4	46.2	60	-	-	-
	500	50%		43.1	63.4	45.3	22.7	46.5	61.1	-	-	-
Deformable-DETR	50	100%		43.9	62.8	47.8	26.1	47.4	58.0	40M	173G	19.1
	50	100%	✓	46.0	65.2	49.8	28.2	49.1	61.0	41M	177G	18.2
Sparse-DETR	50	10%	✓	45.3	65.8	49.3	28.4	48.3	60.1	41M	105G	25.3
	50	20%	✓	45.6	65.8	49.6	28.5	48.6	60.4	41M	113G	24.8
	50	30%	✓	46.0	65.9	49.7	29.1	49.1	60.6	41M	121G	23.2
	50	40%	✓	46.2	66.0	50.3	28.7	49.0	61.4	42M	128G	21.8
	50	50%	✓	46.3	66.0	50.1	29.0	49.5	60.8	42M	136G	20.5
Sparse-DETR§	50	10%	✓	46.9	67.2	51.0	30.2	49.7	62.3	41M	105G	25.3
	50	20%	✓	47.3	67.1	51.4	29.7	50.3	62.7	41M	113G	24.8
	50	30%	✓	47.4	67.3	51.4	30.1	50.5	62.4	41M	121G	23.2
	50	40%	✓	47.7	67.4	51.6	30.0	50.8	62.9	42M	128G	21.8
	50	50%	✓	47.9	67.5	52.1	30.5	51.2	63.2	42M	136G	20.5
***Swin-T** backbone:*												
DETR	500	100%		45.4	66.2	48.1	22.9	49.5	65.9	45M	92G	26.8
Deformable-DETR	50	100%		45.7	65.3	49.9	26.9	49.4	61.2	40M	180G	15.9
	50	100%	✓	48.0	68.0	52.0	30.3	51.4	63.7	41M	185G	15.4
Sparse-DETR	50	10%	✓	48.2	69.2	52.3	29.8	51.2	64.5	41M	113G	21.2
	50	20%	✓	48.8	69.4	53.0	30.4	51.9	64.8	41M	121G	20.0
	50	30%	✓	49.1	69.5	53.5	31.4	52.5	65.1	41M	129G	18.9
	50	40%	✓	49.2	69.5	53.5	31.4	52.9	64.8	41M	136G	18.0
	50	50%	✓	49.3	69.5	53.3	32.0	52.7	64.9	41M	144G	17.2

Implementation Details. We use ResNet50 (He et al., 2016) and Swin Transformer (Liu et al., 2021) as pre-trained backbone networks, where Swin Transformer is one of the state-of-the-art architecture in the ViT family. We stack 6 encoder and 6 decoder layers, each with an auxiliary head at the end. We train the model on a $4\times$V100 GPU machine with a total batch size of 16, for 50 epochs, where the initial learning rate is 0.0002 and decayed by 1/10 at the 40 epoch. Unless otherwise specified, we use the same hyperparameters as in Deformable DETR.

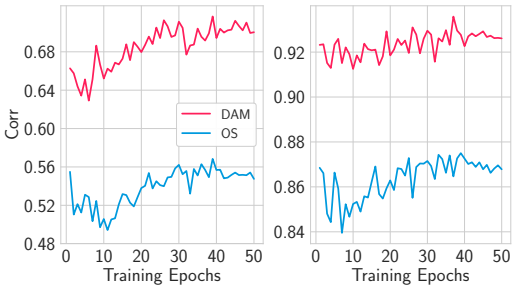
4.1 COMPARISON WITH OBJECT DETECTION BASELINES

Baselines. We compare Sparse DETR with Faster-RCNN with FPN (Lin et al., 2017), DETR (Carion et al., 2020), Deformable DETR (Zhu et al., 2021), and PnP DETR (Wang et al., 2021). We also compare with DETR and Deformable DETR that uses Swin-Tiny (Liu et al., 2021) as a backbone.



(a) ResNet50 (b) Swin-T

Figure 4: **Selection criteria.**



(a) Keeping ratio 10% (b) Keeping ratio 40%

Figure 5: **Correlation graph**.

Here, for brevity, we denote Deformable DETR with the top-k object query selection and bounding box refinement, as Deformable DETR+. In Sparse DETR, encoder queries are sparsified with keeping ratios of 10%, 20%, 30%, 40%, and 50%, using DAM criterion. We demonstrate detection performance and inference costs on COCO `val2017` dataset.

Result. Table 1 shows the results of Sparse DETR and the other baselines on COCO `val2017` set. Remarkably, on the ResNet50 backbone, Sparse DETR with a keeping ratio over 30% outperforms all the baselines while minimizing the computational cost. Even with the keeping ratio reduced down to 10%, Sparse DETR still performs better than most baselines except for Deformable DETR+. More surprisingly, on the Swin-T backbone, Sparse DETR only with the keeping ratio 10% outperforms all the baselines with no exception, while improving FPS by 38% compared to Deformable DETR+.

Remark that, compared to the most competitive baseline, Deformable DETR+, the improvement in AP_L is relatively noticeable on the Swin-T backbone even under the extreme sparsity of 10%, while the performance gap on the ResNet50 backbone comes evenly from different sizes of objects. We conjecture it is because a single token in Swin-T can hold a wider region of information than the one in ResNet50, so even if we aggressively sparsify the encoder query, the network seems to have enough information to detect objects.

4.2 Comparison Between Query Selection Criteria

Baselines. To verify the benefits of the proposed saliency criteria, we compare three query sparsification criteria: random, Objectness Score (OS), and Decoder cross-Attention Map (DAM). The random baseline samples a fixed ratio of arbitrary queries. Note that the proposed encoder auxiliary loss is applied for all the methods.

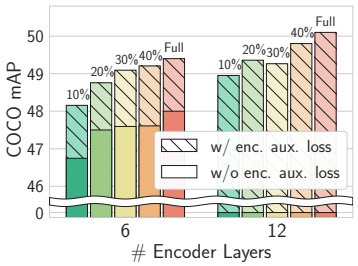
Result. As illustrated in Fig. 4, the random strategy suffers noticeable performance degradation. On the other hand, the DAM-based model outperforms the OS-based model at every ratio and almost catches up with the non-sparse baseline when using 50% of queries. See the **Appendix A.4** for detailed results of this experiment.

To analyze the reason that DAM-based model outperforms its counterpart, we measure the overlap between the queries referred by the decoder and the queries refined by the encoder. As a metric, we compute a scalar correlation $Corr$ as:

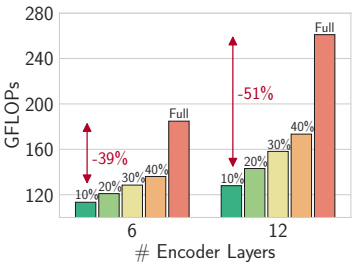
$$Corr := \frac{\sum_{x \in \Omega_D \cap \Omega_s^\rho} \text{DAM}_x}{\sum_{x \in \Omega_D} \text{DAM}_x},$$

where Ω_D is the query set referred by the decoder and DAM_x is the DAM-value corresponding to query x. This $Corr$ metric indicates the ratio of queries polished by the encoder among the queries referred by the decoder.

Fig. 5 demonstrates that $Corr$ of DAM-based model rises higher than that of OS-based model. This result implies that DAM-based model is a more suitable sparsification method for the decoder,



(a) mAP



(b) GFLOPs

Figure 6: **Ablation of # encoder layers**.

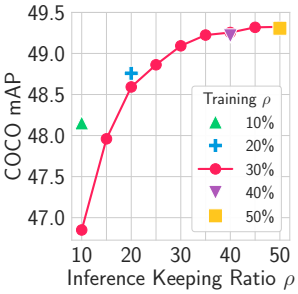


Figure 7: **Dynamic sparsification**.

because the queries referenced by the decoder are explicitly refined in the encoder, which achieves better detection performance. See the **Appendix A.4** for detailed results of this experiment.

4.3 Effectiveness of the encoder auxiliary loss.

Owing to the sparsified query set in our model, we can apply the auxiliary loss to the encoder layers without sacrificing too much computational cost. Apart from improving efficiency and performance, we find another benefit of the encoder auxiliary loss that allows us to safely stack more encoder layers without failing to converge.

As shown in Fig. 6, the encoder auxiliary loss not only enhances detection performance, but also consistently increases detection performance as the encoder layers doubled to 12. However, we observe that the training without its assistance utterly fails when using 12 encoder layers. We conjecture that gradient propagated through decoder cross-attention may vanish as we stack more encoder layers, thus intermediate gradients from the auxiliary loss are required. See the **Appendix A.5** for detailed results of this experiment.

4.4 Dynamic Sparsification for Inference Stage

To deploy the models in various hardware conditions of real-world applications, one often should retrain them at different scales according to the performance-computation trade-off required. We evaluate if our model trained with a fixed sparsity can adapt well to dynamic sparsity at inference time to check out Sparse DETR can avoid that hassle. Figure 7 shows the performance under the varied keeping ratio (ρ) during inference when the model trained using the Swin-T backbone and 30% of encoder queries with the DAM-based method. When the inference keeping ratio is small, the performance of dynamic sparsification is slightly degraded, but the overall performance is satisfactory at various keeping ratios given that only a single model is used.

PnP DETR introduces *dynamic ratio training* to achieve similar performance to the fixed keeping ratio counterpart. However, without the additional trick, it suffers significant performance degradation, for instance, 5.0 AP drop when training/inference keeping ratio is 0.33/0.5, despite the increased number of queries. On the contrary, Sparse DETR achieves 0.2 AP improvement in a similar condition where the training/inference keeping ratio is 0.3/0.5. To conclude, our method shows better robustness compared to PnP DETR without further treatment, showing a greater potential of dynamic adaptation to different hardware environments. Note that any technique such as dynamic ratio training is orthogonal to our method and introducing it may bring even more robustness.

5 Conclusion

In this paper, we have presented encoder query sparsification algorithm that lowers the computational cost of the encoder, which is a computational bottleneck in the DETR and Deformable DETR. By doing so, the proposed Sparse DETR architecture outperforms the Deformable DETR even when using only 10 % of the encoder query, and decreases the overall computation by 38%, and increases the FPS by 42% compared to the Deformable DETR. We hope that our proposed method will provide insights to effectively detect objects in the transformer structure in the future.

REFERENCES

Lei Jimmy Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

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

Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. *CoRR*, abs/1904.10509, 2019.

Krzysztof Marcin Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamás Sarlós, Peter Hawkins, Jared Quincy Davis, Afroz Mohiuddin, Lukasz Kaiser, David Benjamin Belanger, Lucy J. Colwell, and Adrian Weller. Rethinking attention with performers. In *ICLR*. OpenReview.net, 2021.

Jifeng Dai, Yi Li, Kaiming He, and Jian Sun. R-FCN: object detection via region-based fully convolutional networks. In *NIPS*, pp. 379–387, 2016.

Jifeng Dai, Haozhi Qi, Yuwen Xiong, Yi Li, Guodong Zhang, Han Hu, and Yichen Wei. Deformable convolutional networks. In *ICCV*, 2017.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross B. Girshick. Mask R-CNN. In *ICCV*, 2017.

Dan Hendrycks and Kevin Gimpel. Bridging nonlinearities and stochastic regularizers with gaussian error linear units. *CoRR*, abs/1606.08415, 2016.

Jonathan Ho, Nal Kalchbrenner, Dirk Weissenborn, and Tim Salimans. Axial attention in multidimensional transformers. *CoRR*, abs/1912.12180, 2019.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. In *ICML*, volume 119 of *Proceedings of Machine Learning Research*, pp. 5156–5165. PMLR, 2020.

Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. In *ICLR*. OpenReview.net, 2020.

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.

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

Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *ICCV*, 2021.

Bowen Pan, Yifan Jiang, Rameswar Panda, Zhangyang Wang, Rogério Feris, and Aude Oliva. IA-RED^2: Interpretability-aware redundancy reduction for vision transformers. *arXiv preprint arXiv:2106.12620*, 2021.

Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Lukasz Kaiser, Noam Shazeer, Alexander Ku, and Dustin Tran. Image transformer. In *ICML*, volume 80 of *Proceedings of Machine Learning Research*, pp. 4052–4061. PMLR, 2018.

Yongming Rao, Wenliang Zhao, Benlin Liu, Jiwen Lu, Jie Zhou, and Cho-Jui Hsieh. DynamicViT: efficient vision transformers with dynamic token sparsification. *arXiv preprint arXiv:2106.02034*, 2021.

Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster R-CNN: towards real-time object detection with region proposal networks. In *NIPS*, pp. 91–99, 2015.

Byungseok Roh, Wuhyun Shin, Ildoo Kim, and Sungwoong Kim. Spatilly consistent representation learning. In *CVPR*. IEEE, 2021.

Peize Sun, Yi Jiang, Enze Xie, Wenqi Shao, Zehuan Yuan, Changhu Wang, and Ping Luo. What makes for end-to-end object detection? In *ICML*, volume 139, pp. 9934–9944, 2021.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *CVPR*, pp. 1–9. IEEE Computer Society, 2015.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*, 2017.

Huiyu Wang, Yukun Zhu, Bradley Green, Hartwig Adam, Alan L. Yuille, and Liang-Chieh Chen. Axial-deeplab: Stand-alone axial-attention for panoptic segmentation. In *ECCV (4)*, volume 12349 of *Lecture Notes in Computer Science*, pp. 108–126. Springer, 2020.

Tao Wang, Li Yuan, Yunpeng Chen, Jiashi Feng, and Shuicheng Yan. PnP-DETR: towards efficient visual analysis with transformers. In *ICCV*, 2021.

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.

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.

A APPENDIX

A.1 IMPLEMENTATION DETAILS OF THE SCORING NETWORK.

The scoring network is consists of 4 linear layers where Layer Normalization (Ba et al., 2016) comes before the first layer and every layer except for the last one is followed by GELU (Hendrycks & Gimpel, 2016) activation. The output dimension of the 1st layer is 256 and halved to 128 and 64 at the 2nd and 3rd layers. The last layer outputs 1-d logit for the BCE loss. Since the network locally processes the input tokens in a token-wise manner, the final decisions may overlook global statistics without additional treatment. To remedy this issue, we set aside half of the output dimension of the first layer as a global feature, and average them across the whole token set, then concatenate it with each of the remained local features to maintain the original dimension. We also exclude the tokens that correspond to the zero-padded area when selecting top-$\rho\%$ scores, thereby we can prevent those tokens from participating in Hungarian matching process and getting meaningless gradients from the detection objective.

A.2 DAM CREATION IN DEFORMABLE ATTENTION

As attention offset calculated in deformable attention is a fractional position, deformable attention uses bilinear interpolation to get values. Thus, we also use bilinear interpolation to obtain DAM.

Assume that, one of the attention offsets, weights and the reference point of query q is calculated as p, A and r, respectively. Then, deformable attention takes values as:

$$\sum_{x} A \cdot G(x, r+p) \cdot v(x)$$

, where x enumerates all integral spatial locations in the feature map, $G(\cdot,\cdot)$ is the bilinear interpolation kernel defined as $G(a,b) = \max(0, 1-|a_x - b_x|) \cdot \max(0, 1-|a_y - b_y|)$ and v is the values. Similarly, we accumulate DAM-value for location x as:

$$\sum_{(p,A,r)} A \cdot G(x, r+p)$$

. Then, we accumulate DAM over every object query.

A.3 ALTERNATIVE OBJECTIVES FOR DAM-BASED MODEL.

As a training objective of the scoring network using DAM, we can consider other alternatives as long as they can encourage the predicted scores to represent the relative saliency of the queries. One of the naive alternatives is the *regression loss* by which the scoring network directly predicts the values in DAM. The *ranking loss* can be another choice with which the network focuses more on learning the relativeness rather than estimating the set of salient queries.

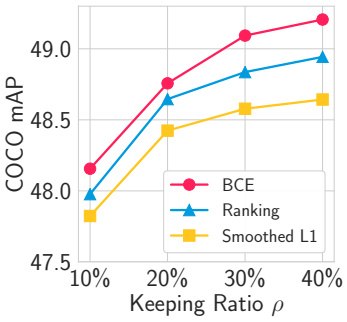


Figure 8: **DAM loss ablation**

Figure 8 shows the default BCE loss outperforms the alternatives. First, it is well-known that the regression problem is much harder than classification. Furthermore, since the value of DAM changes during training, the regression loss to predict the accurate value is more difficult. In case of the pairwise ranking loss, ranking the DAM elements may also be unstable as DAM gradually evolves. Meanwhile, the BCE loss may reduce those element-level noises down to the set-level in that its binary (keep or drop) targets retain more consistency compared to the exact values or ranks. Refer to Table 2 to see the exact values of the points represented in Figure 8.

Table 2: **Comparision between the alternative objectives for DAM-based scoring network**.

Loss	Keeping ratio (ρ)	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
smoothed L1	10%	47.8	68.9	51.7	29.8	50.9	64.0
	20%	48.4	69.0	52.5	31.1	51.4	64.7
	30%	48.6	69.0	52.6	31.1	51.9	64.7
	40%	48.6	69.3	52.9	33.4	51.8	64.5
ranking	10%	48.0	69.1	52.1	29.9	51.4	64.6
	20%	48.7	69.5	53.0	31.1	51.8	65.1
	30%	48.8	69.2	52.8	31.4	52.0	64.9
	40%	48.9	69.3	53.1	31.5	52.2	64.7
BCE	10%	48.2	69.2	52.3	29.8	51.2	64.5
	20%	48.8	69.4	53.0	30.4	51.9	64.8
	30%	49.1	69.5	53.5	31.4	52.5	65.1
	40%	49.2	69.5	53.5	31.4	52.9	64.8

Table 3: **Comparison between query selection criteria when using R50 backbone**.

Scoring method	Keeping ratio (ρ)	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L	params	FLOPs	FPS
N/A	100%	46.3	65.8	50.1	29.0	49.4	61.7	41M	177G	18.2
	0%	42.2	63.0	45.6	25.9	45.3	56.5	36M	91G	35.0
random	10%	43.6	64.3	47.2	26.7	46.9	58.4	41M	102G	27.7
	20%	44.0	64.8	47.8	27.3	47.0	58.4	41M	110G	25.6
	30%	44.0	64.9	47.5	27.4	47.4	58.2	41M	117G	24.1
	40%	44.5	65.1	48.0	27.3	47.8	59.8	41M	125G	22.5
	50%	44.4	64.8	48.0	27.8	47.4	59.2	41M	133G	21.1
OS	10%	44.9	65.2	48.7	27.9	47.8	60.4	41M	106G	26.6
	20%	45.5	65.5	49.3	28.7	48.3	60.5	41M	114G	24.7
	30%	45.7	65.8	49.5	29.7	48.5	60.8	41M	121G	23.2
	40%	45.8	65.5	49.8	29.1	48.8	60.5	41M	129G	21.8
	50%	46.0	65.9	49.8	28.8	48.9	60.6	41M	136G	20.6
DAM	10%	45.3	65.8	49.3	28.4	48.3	60.1	41M	105G	26.5
	20%	45.6	65.8	49.6	28.5	48.6	60.4	41M	113G	24.8
	30%	46.0	65.9	49.7	29.1	49.1	60.6	41M	121G	23.2
	40%	46.2	66.0	50.3	28.7	49.0	61.4	41M	128G	21.8
	50%	46.3	66.0	50.1	29.0	49.5	60.8	41M	136G	20.5

A.4 EXPERIMENTAL DETAILS FOR DIFFERENT QUERY SELECTION CRITERIA

Table 3 and 4 contains specific values used to plot (a) ResNet50 and (b) Swin-T backbone in Figure 4, respectively. Additionally, they also include a lower-bound baseline that has no scoring method with keeping ratio 0%, meaning that the entire encoder block is removed and the backbone features are directly passed to the decoder. Even with the lowest keeping ratio 10%, all the scoring methods including random criterion outperform this lower-bound baseline. Note that all experiments reported in Table 3 and 4 except for the keeping ratio 0% use the encoder auxiliary loss for training.

A.5 EXPERIMENTAL DETAILS FOR EFFECTIVENESS OF THE ENCODER AUXILIARY LOSS.

Table 5 presents the detailed values of Figure 5. As shown in the table, training of a deeper encoder of more than 9 layers fails without the auxiliary loss, but if it is adopted, the convergence becomes feasible as the intermediate gradients provided to the early encoder layers may augment the vanishing gradients propagated from the decoder layers.

Table 4: **Comparison between query selection criteria when using Swin-T backbone**.

Scoring method	Keeping ratio (ρ)	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L	params	FLOPs	FPS
N/A	100%	49.4	69.4	53.5	31.9	52.6	65.1	41M	185G	15.4
	0%	43.7	65.8	46.9	27.0	46.7	60.0	37M	96G	26.5
random	10%	45.5	67.6	48.8	28.4	48.5	62.2	41M	110G	22.1
	20%	45.6	67.5	49.2	28.6	49.1	62.2	41M	118G	20.8
	30%	46.2	68.1	49.7	29.5	49.7	63.0	41M	125G	19.7
	40%	46.5	68.2	50.0	29.9	49.8	63.0	41M	133G	18.7
	50%	47.2	68.3	50.9	29.1	50.4	63.9	41M	141G	17.7
OS	10%	48.0	69.1	52.1	29.9	51.1	64.4	42M	114G	21.4
	20%	48.3	69.1	52.5	30.4	51.6	64.2	42M	122G	20.2
	30%	48.6	69.2	53.0	31.0	52.0	64.6	42M	129G	18.6
	40%	48.9	69.4	53.1	33.0	51.9	64.5	42M	137G	18.2
	50%	49.0	69.2	53.5	31.2	52.4	65.0	42M	145G	17.2
DAM	10%	48.2	69.2	52.3	29.8	51.2	64.5	41M	113G	21.2
	20%	48.8	69.4	53.0	30.4	51.9	64.8	41M	121G	20.0
	30%	49.1	69.5	53.5	31.4	52.5	65.1	41M	129G	18.9
	40%	49.2	69.5	53.5	31.4	52.9	64.8	41M	136G	18.0
	50%	49.3	69.5	53.3	32.0	52.7	64.9	41M	144G	17.2

Table 5: **Effectiveness of the encoder auxiliary loss**. When the number of encoder layers is more than 9, the model training fails, but if the encoder auxiliary loss is adopted, the model training is feasible regardless of the number of encoder layers, and accuracy is improved.

# of encoder	Keeping ratio (ρ)	Aux. loss	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L	params	FLOPs	FPS
6	100%		48.0	68.0	52	30.3	51.4	63.7	41M	185G	15.4
	100%	✓	49.4	69.4	53.5	31.9	52.6	65.1	41M	185G	15.4
	10%		46.8	68.0	50.6	29.7	49.7	63.3	41M	113G	21.2
	20%		47.5	68.3	51.4	31.4	50.4	64.4	41M	121G	20.0
	30%		47.6	67.9	51.4	29.9	51.1	63.9	41M	129G	18.9
	40%		47.6	68.2	51.5	30.3	50.8	64.0	41M	136G	18.0
	10%	✓	48.2	69.2	52.3	29.8	51.2	64.5	41M	113G	21.2
	20%	✓	48.8	69.4	53.0	30.4	51.9	64.8	41M	121G	20.0
	30%	✓	49.1	69.5	53.5	31.4	52.5	65.1	41M	129G	18.9
	40%	✓	49.2	69.5	53.5	31.4	52.9	64.8	41M	136G	18.0
9	100%	✓	49.7	69.4	54.1	32.4	52.9	65.4	44M	220G	12.8
12	100%	✓	50.1	69.6	54.6	32.2	53.4	65.8	46M	261G	11.0
	10%	✓	49.0	69.5	53.5	31.6	52.2	65.2	46M	128G	19.2
	20%	✓	49.4	69.6	53.5	31.9	52.8	65.4	46M	143G	17.5
	30%	✓	49.3	69.4	53.6	31.7	52.5	65.6	46M	158G	15.6
	40%	✓	49.8	69.8	54.3	33.1	53.4	65.4	46M	173G	14.6

A.6 USING A LARGER TRANSFORMER-BASED BACKBONE(SWIN-B)

We perform experiments on Sparse DETR with Swin-Base(Liu et al., 2021) backbone to see if our method shows similar efficiency and performance gain even when using a heavier transformer-based backbone. Table 6 illustrates a comparison of COCO detection performance between Swin-T and Swin-B backbone under the varied sparsity. Due to the increased capacity, using Swin-B backbone significantly boosts up the baseline AP up to 52.5 (+4.5) but with 2.4$\times$ parameters and 2.1$\times$ computational cost. With the keeping ratio of 40% and the encoder auxiliary loss, the performance gap remains at a similar level(+4.1). We can also observe consistent performance gains as the keeping ratio gets higher while the increasing gap converges more quickly than Swin-T. It may be because a single visual token of Swin-B can incorporate a wider range of information due to the deeper at-

tention hierarchies and a smaller number of tokens is required to fully represent all the objects in an image. Note that the efficiency of a backbone network is behind the scope of this paper. Our work is orthogonal to the backbone sparsification approaches, e.g. DynamicViT (Rao et al., 2021), and we leave the integration with those works as future work.

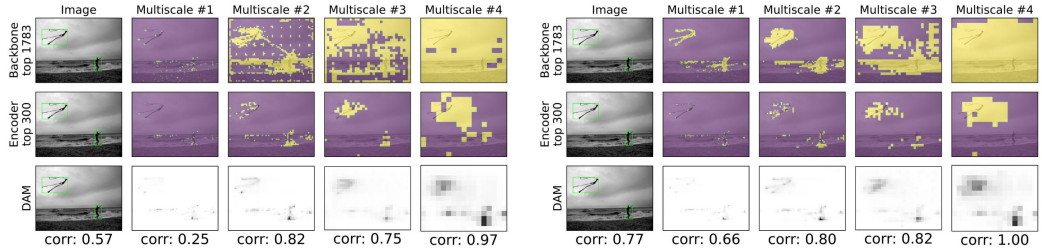
Table 6: **Performance of Sparse DETR with Swin-B backbone**. The same environment and hyperparameters as Experiments section are used, except for changing the backbone to a larger scale. Note that Aux. loss means only the ones applied to the encoder layers.

Backbone	Keeping ratio (ρ)	Aux. loss	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L	params	FLOPs	FPS
Swin-T	100%		48.0	68.0	52	30.3	51.4	63.7	41M	185G	15.4
	10%	✓	48.2	69.2	52.3	29.8	51.2	64.5	41M	113G	21.2
	20%	✓	48.8	69.4	53.0	30.4	51.9	64.8	41M	121G	20.0
	30%	✓	49.1	69.5	53.5	31.4	52.5	65.1	41M	129G	18.9
	40%	✓	49.2	69.5	53.5	31.4	52.9	64.8	41M	136G	18.0
Swin-B	100%		52.5	72.9	56.9	34.7	56.5	69.6	101M	400G	7.6
	10%	✓	52.2	73.5	57.0	34.0	56.3	70.3	101M	335G	8.8
	20%	✓	53.1	73.8	57.9	34.6	56.9	70.6	101M	343G	8.6
	30%	✓	53.2	73.7	57.7	35.3	56.8	70.8	101M	350G	8.4
	40%	✓	53.3	73.4	58.0	36.3	57.2	70.9	101M	358G	8.2

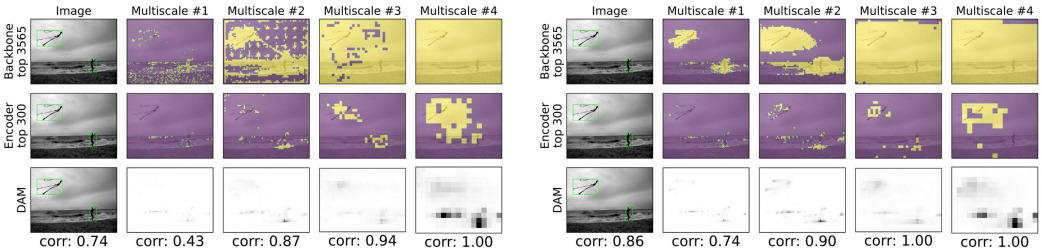
A.7 Visualizations of Selected Encoder Queries

We visualize selected encoder queries and top-k decoder queries for each criterion, OS, and DAM. In the first row, selected encoder queries from the backbone feature map are visualized as yellow regions, whereas unselected queries are visualized as purple regions. In the second row, selected top-k decoder queries from encoder output are visualized in the same manner. In the final row, DAM values and $Corr$ metrics are visualized. $Corr$ is measured as in section 4.2.

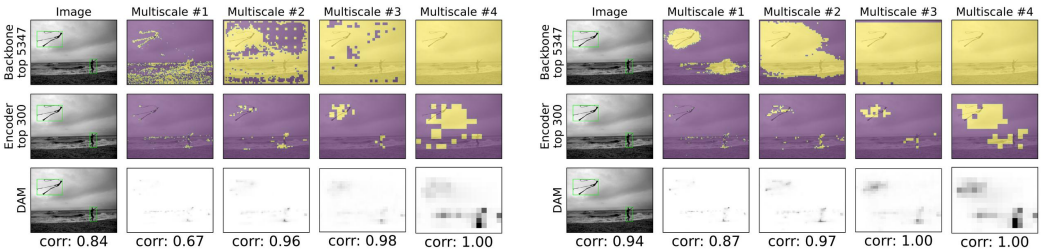
Interestingly, the DAM-based selection seems to better capture the objects than OS-based selection. The OS-based selection also captures objects well, but it typically focuses on the high-frequency edges that are not only in the foreground but also in the background. On the other hand, the DAM-based selection captures the boundary of the objects and also their inner areas and is less distracted from the background edges. We analyze that DAM focuses on the boundary of objects to lower the regression loss, and attends to the inside of objects to lower the classification loss. Finally, the scoring network predicts such a DAM well, and refining the encoder queries according to it finally helps achieve better detection performance.



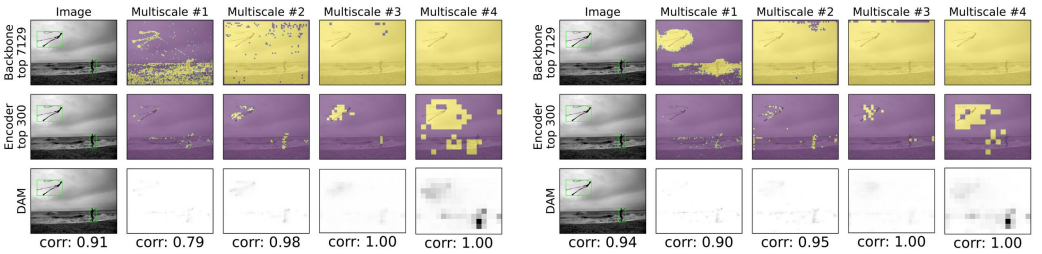
(a) OS-based model ($\rho = 0.1$) (b) DAM-based model ($\rho = 0.1$)



(c) OS-based model ($\rho = 0.2$) (d) DAM-based model ($\rho = 0.2$)

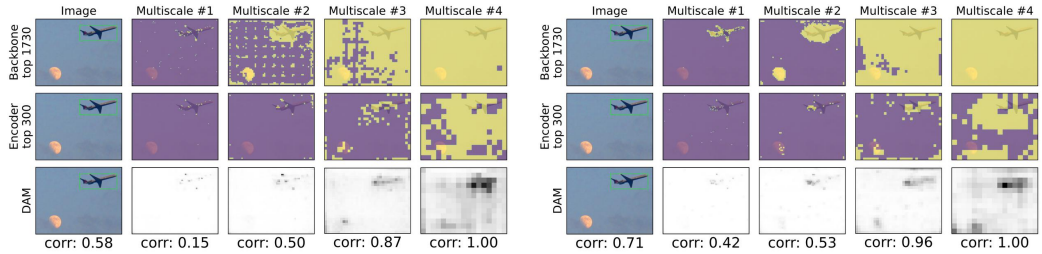


(e) OS-based model ($\rho = 0.3$) (f) DAM-based model ($\rho = 0.3$)



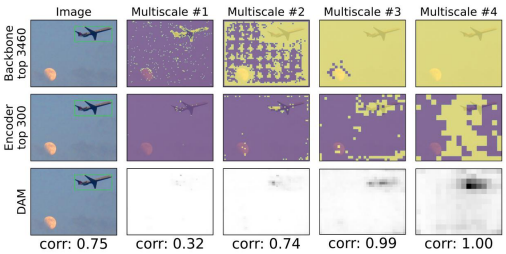
(g) OS-based model ($\rho = 0.4$) (h) DAM-based model ($\rho = 0.4$)

Figure 9: Visualization of selected queries and DAM for COCO validation image #289960

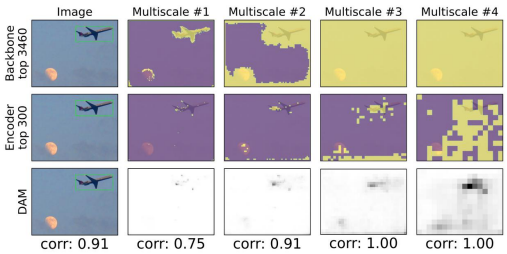


(a) OS-based model ($\rho = 0.1$)

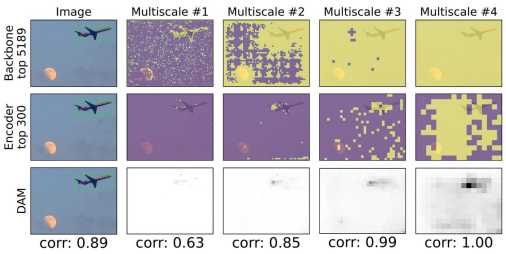
(b) DAM-based model ($\rho = 0.1$)



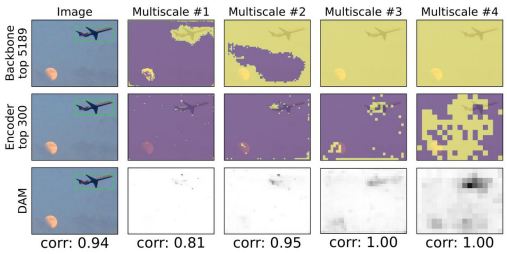
(c) OS-based model ($\rho = 0.2$)



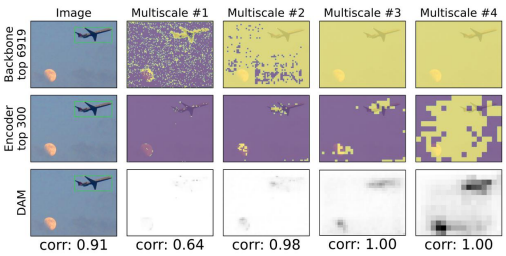
(d) DAM-based model ($\rho = 0.2$)



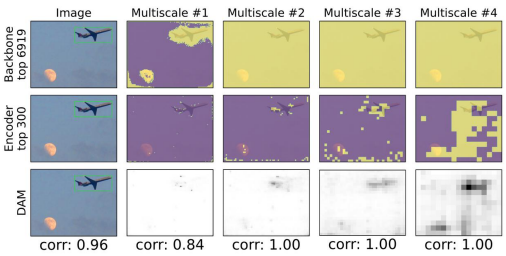
(e) OS-based model ($\rho = 0.3$)



(f) DAM-based model ($\rho = 0.3$)

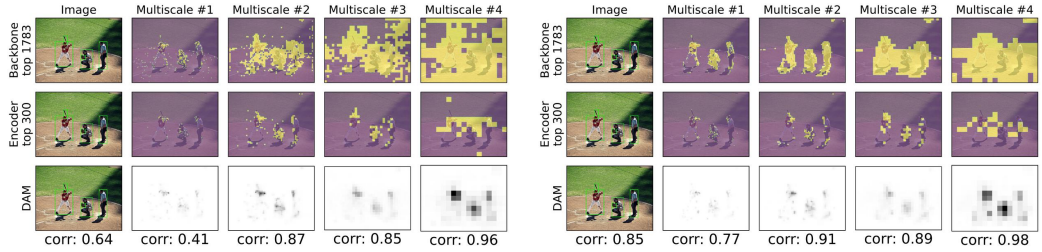


(g) OS-based model ($\rho = 0.4$)



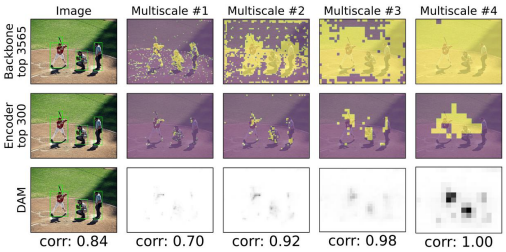
(h) DAM-based model ($\rho = 0.4$)

Figure 10: Visualization of selected queries and DAM for COCO validation image #22396

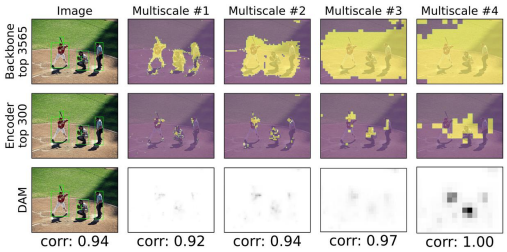


(a) OS-based model ($\rho = 0.1$)

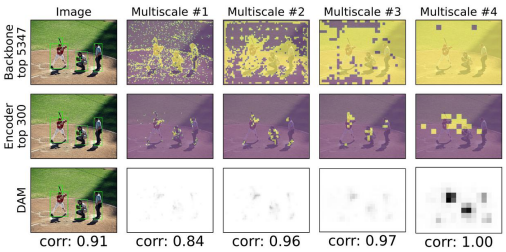
(b) DAM-based model ($\rho = 0.1$)



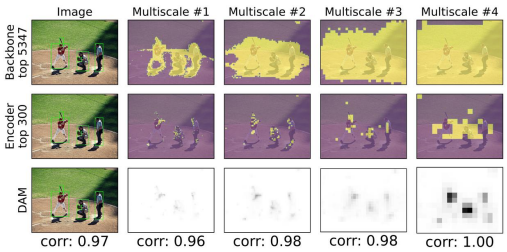
(c) OS-based model ($\rho = 0.2$)



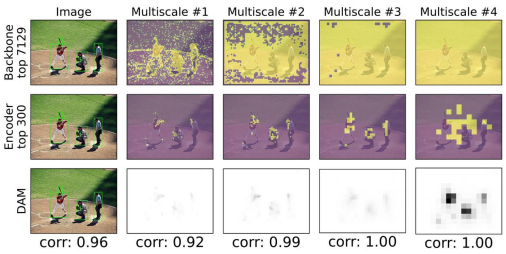
(d) DAM-based model ($\rho = 0.2$)



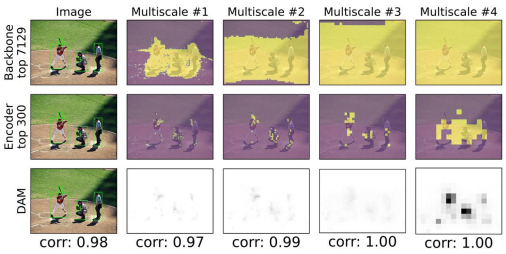
(e) OS-based model ($\rho = 0.3$)



(f) DAM-based model ($\rho = 0.3$)

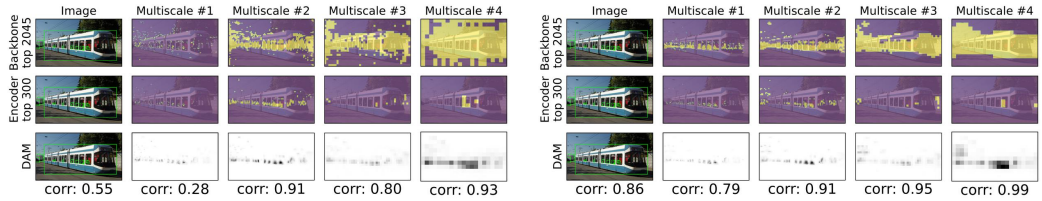


(g) OS-based model ($\rho = 0.4$)



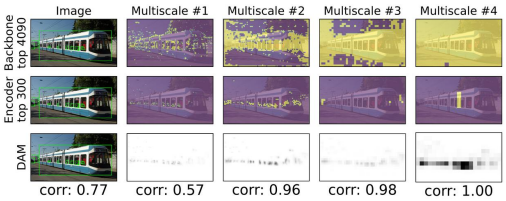
(h) DAM-based model ($\rho = 0.4$)

Figure 11: Visualization of selected queries and DAM for COCO validation image #46252

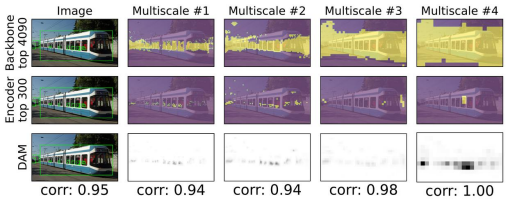


(a) OS-based model ($\rho = 0.1$)

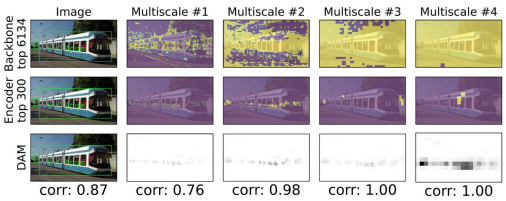
(b) DAM-based model ($\rho = 0.1$)



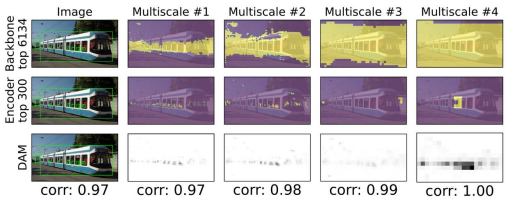
(c) OS-based model ($\rho = 0.2$)



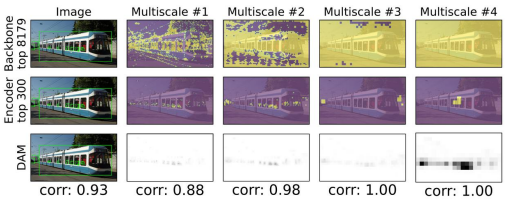
(d) DAM-based model ($\rho = 0.2$)



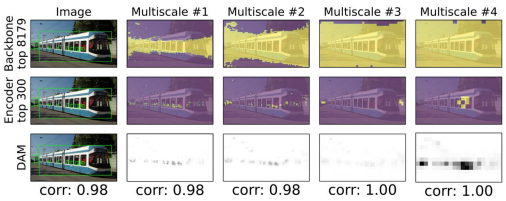
(e) OS-based model ($\rho = 0.3$)



(f) DAM-based model ($\rho = 0.3$)

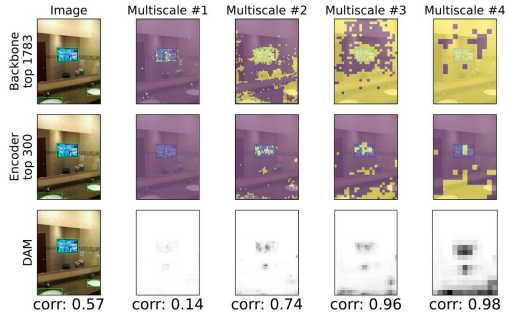


(g) OS-based model ($\rho = 0.4$)

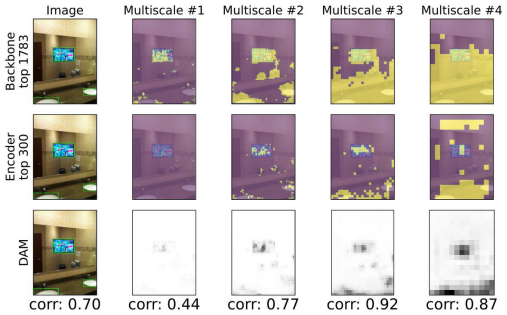


(h) DAM-based model ($\rho = 0.4$)

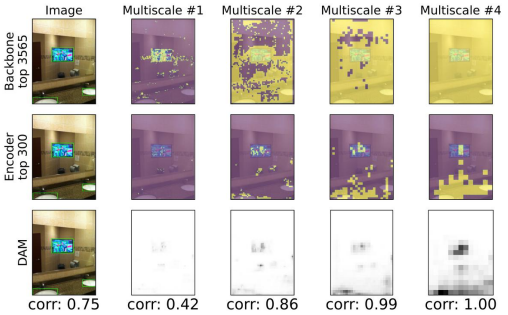
Figure 12: Visualization of selected queries and DAM for COCO validation image #6040



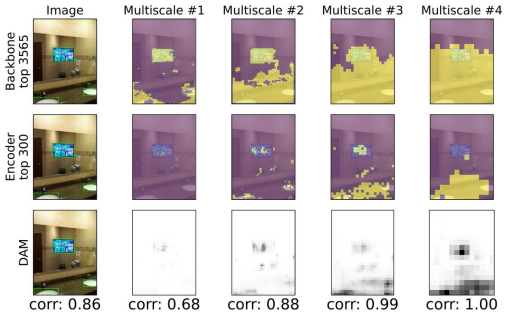
(a) OS-based model ($\rho = 0.1$)



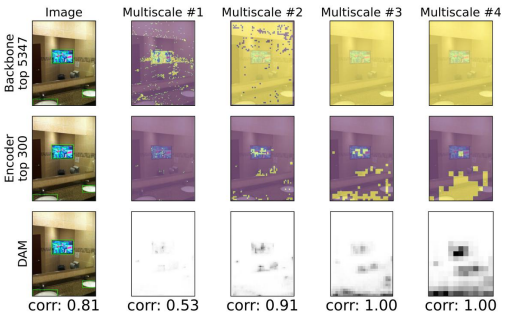
(b) DAM-based model ($\rho = 0.1$)



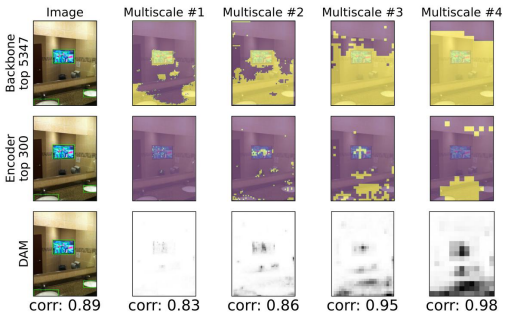
(c) OS-based model ($\rho = 0.2$)



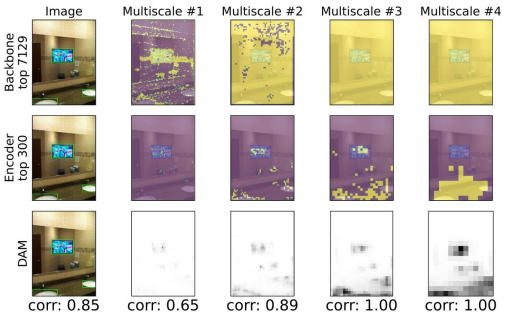
(d) DAM-based model ($\rho = 0.2$)



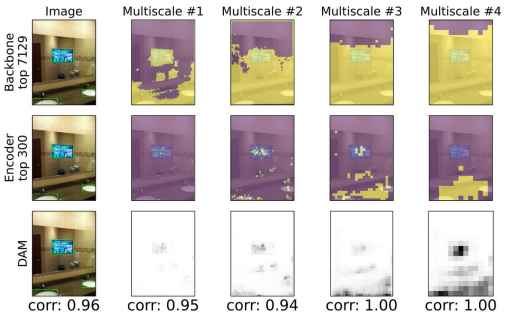
(e) OS-based model ($\rho = 0.3$)



(f) DAM-based model ($\rho = 0.3$)



(g) OS-based model ($\rho = 0.4$)



(h) DAM-based model ($\rho = 0.4$)

Figure 13: Visualization of selected queries and DAM for COCO validation image #17379