# Super Sparse DETR: YOLO-Competitive Convergence and Acceleration

## Supplementary Material

## A. Complexity Analysis

In this appendix we analyze the computational complexity of Super Sparse DETR and relate the theoretical savings to the design in Sec. 3. In particular, we show how encoder token sparsity, query sparsity and FFN width sparsity jointly reduce the FLOPs and memory footprint of the DEIM–based baseline while preserving its backbone and matching pipeline.

**Notation.** We follow the notation in Sec. 3. Let

- $L$ be the encoder sequence length, i.e., the total number of multi-scale tokens after flattening all feature levels;
- $Q$ be the number of decoder queries in the DEIM baseline;
- $C$ be the hidden width of the FFN;
- $\rho_{\text{enc}} \in (0, 1]$ be the encoder token keep ratio in Eq. (1), so that the gathered sequence length is $L' = \rho_{\text{enc}} L$;
- $K \leq Q$ be the number of queries preserved by our query sparsification in Eq. (1);
- $\alpha \in (0, 1]$ be the FFN width ratio in Eq. (1), so that the effective FFN width becomes $C' = \alpha C$;
- $d$ be the model dimension, $H$ the number of heads, and $P$ the number of sampling points per head in deformable attention.

We denote the number of encoder and decoder layers by $N_e$ and $N_d$, respectively.

### A.1. Baseline transformer complexity

The DEIM baseline adopts a Deformable DETR–style encoder–decoder. For a single encoder layer with deformable self-attention and one FFN, the dominant FLOPs can be written as

$$\mathcal{F}_{\text{enc}}^{\text{base}} \approx c_{\text{attn}}\, L\, HPd \;+\; c_{\text{ffn}}\, LCd, \tag{9}$$

where $c_{\text{attn}}, c_{\text{ffn}}$ are architecture-dependent constants. The first term comes from sampling $P$ key/value locations per token and head, and the second from the two linear layers in the FFN.

Similarly, for a single decoder layer with deformable cross-attention over the encoder output and its own FFN, the leading cost is

$$\mathcal{F}_{\text{dec}}^{\text{base}} \approx \tilde{c}_{\text{attn}}\, Q\, HPd \;+\; \tilde{c}_{\text{ffn}}\, QCd. \tag{10}$$

The total transformer cost of the baseline is then

$$\mathcal{F}_{\text{trans}}^{\text{base}} \approx N_e \mathcal{F}_{\text{enc}}^{\text{base}} + N_d \mathcal{F}_{\text{dec}}^{\text{base}}, \tag{11}$$

with backbone, classification and regression heads contributing an additional but smaller constant term. Memory usage scales linearly with the sequence and query lengths, $\mathcal{O}(Ld)$ for encoder activations and $\mathcal{O}(Qd)$ for decoder activations.

### A.2. Effect of encoder token sparsity

Secs. 3.2 and 3.3 show that we first learn token scores via DAM and then apply skip-update during training,

$$x \leftarrow x + M \odot \Delta_{\text{attn}}(x), \tag{12}$$

where $M \in \{0, 1\}^{B \times L \times 1}$ is a binary keep mask, and later convert the learned sparsity pattern into a shorter sequence by gather:

$$L' = \rho_{\text{enc}} L. \tag{13}$$

After export, all encoder Q/K/V projections and attention modules operate on length $L'$ instead of $L$. The encoder-layer FLOPs become

$$\begin{aligned} \mathcal{F}_{\text{enc}}^{\text{sparse}} &\approx c_{\text{attn}}\, L'\, HPd + c_{\text{ffn}}\, L'\, \alpha C\, d \\ &= \rho_{\text{enc}} \Big( c_{\text{attn}}\, L\, HPd + \alpha\, c_{\text{ffn}}\, LCd \Big) \end{aligned} \tag{14}$$

Thus the encoder FLOPs are reduced by approximately

$$\frac{\mathcal{F}_{\text{enc}}^{\text{sparse}}}{\mathcal{F}_{\text{enc}}^{\text{base}}} \approx \rho_{\text{enc}} \cdot \frac{c_{\text{attn}} + \alpha c_{\text{ffn}}}{c_{\text{attn}} + c_{\text{ffn}}} \lesssim \rho_{\text{enc}}, \tag{15}$$

where the last inequality uses the fact that FFN cost dominates attention in typical DETR settings. The encoder memory footprint also scales from $\mathcal{O}(Ld)$ down to $\mathcal{O}(\rho_{\text{enc}} Ld)$.

### A.3. Effect of query sparsity

On the decoder side, we aggregate query usage $u_j^{\text{qry}}$ from DAM (Eq. (3)) and retain only the top-$K$ effective queries as described in Sec. 3.2. After gather, the decoder is instantiated with $K$ queries instead of $Q$. The per-layer cost becomes

$$\begin{aligned} \mathcal{F}_{\text{dec}}^{\text{sparse}} &\approx \tilde{c}_{\text{attn}}\, K\, HPd + \tilde{c}_{\text{ffn}}\, K\, \alpha C\, d \\ &= \frac{K}{Q} \Big( \tilde{c}_{\text{attn}}\, Q\, HPd + \alpha\, \tilde{c}_{\text{ffn}}\, QCd \Big) \end{aligned} \tag{16}$$

Therefore the decoder FLOPs scale approximately linearly with the retained query ratio $K/Q$, and the decoder memory scales from $\mathcal{O}(Qd)$ to $\mathcal{O}(Kd)$.

### A.4. Effect of FFN width sparsity

Channel Sensitivity (CS) in Sec. 3.2 provides a data-driven importance score $s_k^{\text{ffn}}$ for each FFN hidden channel, based on EMA-smoothed parameter movement and Fisher-style

gradient statistics. Pruning the least important channels yields effective width $C' = \alpha C$.

Given that FFN layers are parameter- and FLOP-dominant in both encoder and decoder, reducing the width linearly decreases their cost:

$$\mathcal{F}^{\text{sparse}}_{\text{ffn, enc}} \approx \alpha\, \mathcal{F}^{\text{base}}_{\text{ffn, enc}}, \qquad \mathcal{F}^{\text{sparse}}_{\text{ffn, dec}} \approx \alpha\, \mathcal{F}^{\text{base}}_{\text{ffn, dec}}, \tag{17}$$

and similarly reduces FFN parameters from $\mathcal{O}(C^2)$ to $\mathcal{O}(\alpha^2 C^2)$.

### A.5. Overall complexity and overhead of scoring

Combining the three forms of structured sparsity, the total transformer FLOPs after gather can be approximated as

$$\begin{aligned}\mathcal{F}^{\text{sparse}}_{\text{trans}} \approx\; & N_e\, \rho_{\text{enc}} \Big( c_{\text{attn}}\, L\, HPd + \alpha\, c_{\text{ffn}}\, LCd \Big) \\ & + N_d\, \frac{K}{Q} \Big( \tilde{c}_{\text{attn}}\, Q\, HPd + \alpha\, \tilde{c}_{\text{ffn}}\, QCd \Big)\end{aligned} \tag{18}$$

which is strictly smaller than $\mathcal{F}^{\text{base}}_{\text{trans}}$ whenever $\rho_{\text{enc}} < 1$, $K < Q$, or $\alpha < 1$. Because the encoder dominates both FLOPs and latency in Deformable-DETR style architectures, the factor $\rho_{\text{enc}}$ provides the largest practical gain, consistent with the empirical latency breakdown in Fig. 4.

Finally, the additional cost introduced by our scoring machinery is negligible compared to the savings:

- DAM aggregation reuses decoder cross-attention weights and only performs lightweight reductions over heads and layers.
- CS uses optimizer statistics $(\hat{m}_k, \hat{v}_k)$ and gradients that are already computed, adding only a few element-wise EMA and normalization operations per channel.

In practice, these operations contribute less than a small fraction of a percent of total FLOPs, while the combination of token, query and FFN sparsity yields substantial end-to-end latency reductions on both COCO and industrial defect benchmarks.

## B. Experimental Settings

In this section we provide the training schedule and sparsity-related hyper-parameters used in all experiments. Unless otherwise stated, we follow the default original settings (optimizer, weight decay, data augmentation, etc.), and only modify the sparsity-specific components described in Sec. 3 of the main paper. For the configurations of the YOLO series on different datasets, we refer to the original configurations in the corresponding papers to achieve a comparative effect and ensure the rigor of the experiments. The resulting schedule is summarized in Tab. 7.

### B.1 Sparsification schedule

During the first part of training, we only *collect* multi-factor scores $\{u^{\text{tok}}, u^{\text{qry}}, s^{\text{ffn}}\}$ without changing the network structure. Structural sparsity is then activated at fixed milestones expressed as fractions of the total number of epochs $T$:

- Query sparsity is applied *around* epoch $0.2T$.
- FFN width sparsity is applied *around* epoch $0.3T$.
- Encoder token sparsity is applied *around* epoch $0.6T$.

We recommend this sparsification schedule based on extensive experiments, which show that these milestones provide a stable trade-off between accuracy and latency. In practice, users can adjust the exact epochs proportionally according to their own tasks and datasets.

This ordering (query $\rightarrow$ FFN $\rightarrow$ token) matches the multi-factor design: we first stabilize the decoder queries based on DAM usage, then shrink FFN capacity based on Channel Sensitivity, and finally shorten the encoder sequence via token sparsity, which has the strongest impact on encoder FLOPs (see Appendix A).

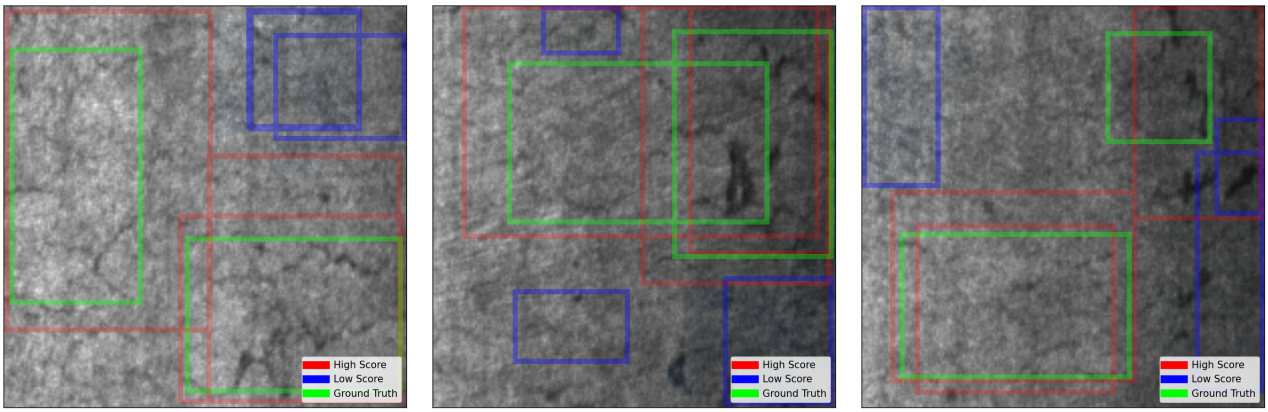
After each structural update we enable a short *compensation phase* with: (i) a concave-rebound learning-rate protection window and (ii) EMA-teacher distillation. Each compensation epoch is applied *after* each structural update:

- For large-scale datasets, we recommend using a compensation length of about 3 epochs. In this case, the sparsity ratios are relatively mild and each epoch already covers a large number of samples, so a small number of compensation epochs is sufficient to recover the accuracy loss after structural changes.
- For small-scale datasets, we recommend about 5 compensation epochs. In this case, the sparsity is typically stronger and each epoch sees fewer samples, so the model needs more iterations to adapt and fully recover its performance.
- For any dataset, the number of compensation epochs should not be set too large. Each compensation epoch requires an additional forward pass of the teacher model, which makes it almost as expensive as running an extra full training epoch. Since compensation is applied online during training and only targets the parts that have been pruned as "less meaningful" by our scoring, using too many compensation epochs would waste training time and may also cause the student to overfit to the teacher's behavior.

### B.2 Sparsification scale

We now detail the meaning and role of the three sparsity budgets ($\rho_{\text{enc}}$, $K$, $\alpha$) that appear in Eq. (1) of the main paper and in the complexity analysis in Appendix A. Specifically, the impacts of the parameters are shown in Tab 8. We keep all other hyper-parameters fixed to the DEIM-D-FINE configuration, so that the effect of each sparsity budget can be isolated. It can be seen that queries have a greater impact on the COCO dataset, while tokens have a more significant influence on industrial datasets.


(a) COCO




(b) NEU-DET

Figure 5. **Sparsification scores of queries.** Here are query scores of different dataset by DAM. As the figures show, high-score queries are closer to the ground-truth objects. Removing low-score (low-matched) queries therefore leads to more accurate and efficient detection.

Table 7. **Hyperparameter schedule across model sizes.**

| Setting | X | L | M | S |
|---|---|---|---|---|
| epochs | 50 | 50 | 90 | 120 |
| FFN_epoch | [17,20] | [17,20] | [25,29] | [30,35] |
| Query_epoch | [10,13] | [10,13] | [15,19] | [20,25] |
| Token_epoch | [32,35] | [32,35] | [52,60] | [70,80] |
| Comp_epoch | 3 | 3 | 4 | 5 |
| LR_Reduction | 0.8 | | | |
| $\tau$ of EMA | 0.9 | | | |
| $\rho_{\text{enc}}$ | 0.8 | | | |
| $K$ | 250 | | | |
| $\alpha$ | 0.75 | | | |
| $\lambda_{\text{cls}}$ / $\lambda_{\text{box}}$ | 0.02 | | | |

Table 8. **Comparison of different scales of sparsity.**

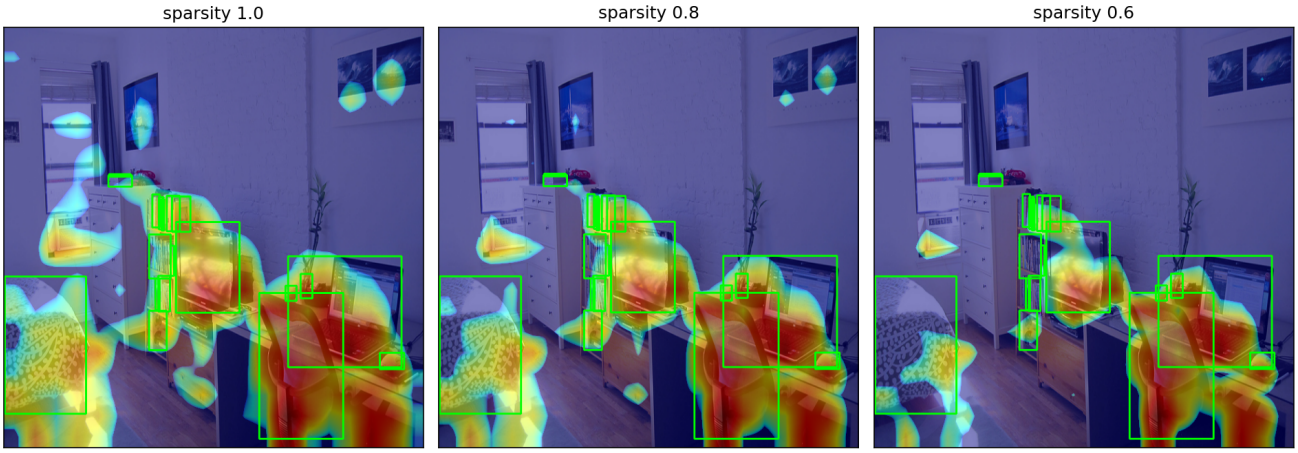
| Setting | latency (ms) | AP of COCO | AP of NEU-DET |
|---|---|---|---|
| $\rho_{\text{enc}} = 1$ | 3.49 | 49.0 | 39.0 |
| $\rho_{\text{enc}} = 0.8$ | 2.97 | 48.8 | 39.9 |
| $\rho_{\text{enc}} = 0.6$ | 2.74 | 48.7 | 38.5 |
| $K = 300$ | 3.49 | 49.0 | 39.0 |
| $K = 250$ | 2.95 | 47.3 | 40.5 |
| $K = 200$ | 2.74 | 45.9 | 40.2 |
| $\alpha = 1.0$ | 3.49 | 49.0 | 39.0 |
| $\alpha = 0.75$ | 3.30 | 48.4 | 40.9 |
| $\alpha = 0.5$ | 3.11 | 48.0 | 40.4 |

**Encoder keep ratio $\rho_{\text{enc}}$** The encoder keep ratio $\rho_{\text{enc}} \in (0, 1]$ specifies the fraction of encoder tokens that remain after the gather operation:

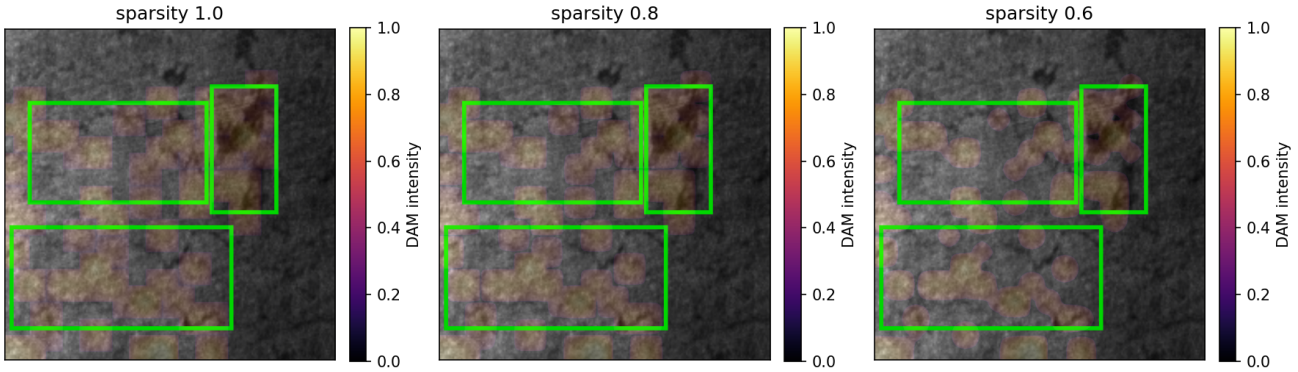
$$L' = \rho_{\text{enc}} L,$$

where $L$ is the original multi-scale encoder sequence length and $L'$ is the shortened length after sparsification. As derived in Appendix A, encoder FLOPs and activation memory both scale approximately linearly with $\rho_{\text{enc}}$:

$$F_{\text{enc}}^{\text{sparse}} \propto \rho_{\text{enc}} L, \qquad \text{Mem}_{\text{enc}}^{\text{sparse}} \propto \rho_{\text{enc}} L.$$




(a) COCO




(b) NEU-DET

Figure 6. **Sparsification scale of tokens by $\rho_{\text{enc}}$.** Here are grouped token heatmaps of different dataset. From left to right, $\rho_{\text{enc}}$ = 1, 0.8, 0.6. AS figures show that small sparsification scale of tokens has a significant impact on latency by ignoring some tokens which are important or unimportant.

Thus $\rho_{\text{enc}}$ directly controls how much encoder computation is removed.

**Number of kept queries $K$.** The parameter $K \leq Q$ denotes the number of decoder queries that are kept after query sparsification. It controls the size of the set prediction space at inference time:

$$Q' = K, \qquad F_{\text{dec}}^{\text{sparse}} \propto K.$$

On large-scale datasets with many classes such as COCO 2017, too small a $K$ severely harms recall, as each query is responsible for covering a subset of objects. This effect is evident in the ablations on COCO and NEU-DET (Tabs. 3 and 4): enabling only query sparsity leads to the largest AP drop on COCO, while token and FFN sparsity are much less harmful. Therefore, on COCO we keep $K$ close to the baseline number of queries and use a moderate reduction ($K = 250$ for a baseline of $Q = 300$).

For industrial defect datasets such as NEU-DET, the number of categories is small and images typically contain only a few objects. In this regime, a large number of queries mainly increases the number of negative samples. As shown in Tab. 3, enabling only query sparsity slightly improves mAP on NEU-DET while reducing latency. This suggests that for small, low-diversity datasets one can choose a more aggressive query budget (e.g. $K/Q$ in the range 0.6–0.8), whereas for COCO-like datasets it is safer to keep $K/Q \approx 0.8$ or higher.

**FFN width ratio $\alpha$.** The FFN width ratio $\alpha \in (0, 1]$ specifies the fraction of FFN hidden channels that are kept after Channel Sensitivity-based pruning. If the original FFN

width is $C$, the effective width after pruning is $C' = \alpha C$. Since FFN layers dominate both the parameter count and FLOPs of transformer blocks, $\alpha$ has a strong influence on both model size and latency:

$$\text{Params}_{\text{FFN}} \propto \alpha^2 C^2, \qquad F_{\text{ffn}}^{\text{sparse}} \propto \alpha C.$$

In our main experiments we set $\alpha = 0.75$, which removes 25% of FFN channels while preserving sufficient capacity in both encoder and decoder. As shown in the ablations in Tab Tabs. 3 and 4, FFN sparsity alone causes only a small AP drop on COCO and even leads to an AP gain on NEU-DET, indicating that many FFN channels are indeed redundant.

**Dataset-dependent choice.** Putting the above together, the three budgets play different roles on different types of datasets:

- **Large, diverse datasets** Encoder token sparsity is the most reliable source of speedup, while query sparsity is the most sensitive to AP. We therefore recommend moderate encoder sparsity and FFN pruning (e.g. $\rho_{\text{enc}} \approx 0.8, \alpha \approx 0.7 \sim 0.8$) and a conservative reduction in queries (e.g. $K/Q \geq 0.8$). The configuration $(\rho_{\text{enc}}, K, \alpha) = (0.8, 250, 0.75)$ used in our experiments follows this rule and leads to less than 0.5% AP drop with about 25% latency reduction (see Tab. 1).
- **Small, low-diversity industrial datasets (NEU-DET).** The number of classes and objects per image is small of industrial datasets, so over-parameterized decoders mostly introduce redundant negatives. As shown in Tab. 3, query and FFN sparsity are the main contributors to AP improvement, while token sparsity mainly controls latency. For such datasets, we recommend using similar or slightly larger $\rho_{\text{enc}}$ to avoid over-sparsifying the already compact encoder, but allowing more aggressive query and FFN pruning (smaller $K/Q$ and $\alpha$) within the range where validation AP is stable.