Super Sparse DETR: YOLO-Competitive Convergence and Acceleration

Hebao Zhu
Chongqing University
20231008@stu.cqu.edu.cn

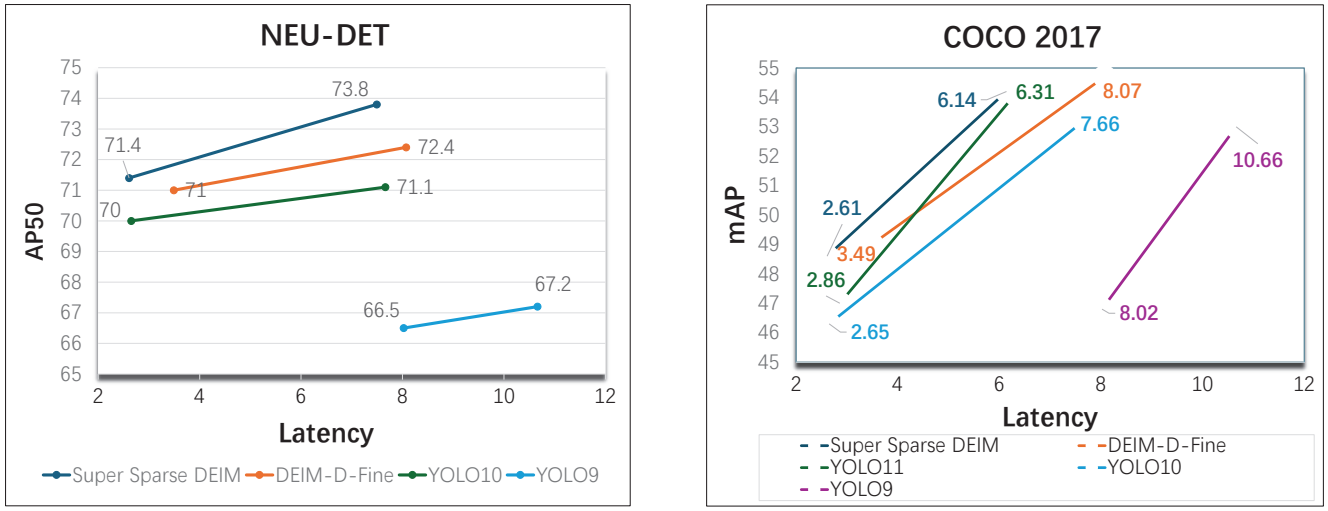


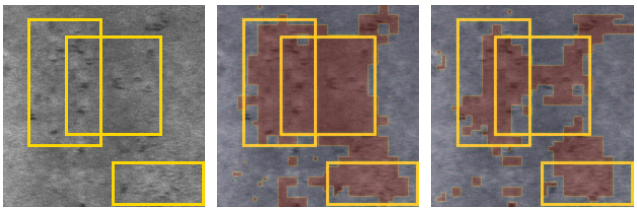
Figure 1. Comparison with state-of-the-art real-time object detectors on COCO [16] and NEU-DET [20]. The proposed Super Sparse model achieves low latency and high average precision when compared to state-of-the-art real-time object detectors

Abstract

We propose Super Sparse DETR, a deployment-oriented structured sparsity training framework for DETR that brings its efficiency closer to YOLO-style detectors. Specifically, our method leverages a decoder-derived attention aggregation map (DAM) and channel sensitivity (CS) to perform stage-wise structural sparsity on key components. Combined with skip-update of non-critical parameters during training and a gather operation at export time, these learned selections translate into deterministic reductions in sequence length and real inference speedups. To mitigate instability and information loss introduced by sparsity, we further apply consistency distillation from an EMA teacher to the pruned model. Extensive experiments on COCO and industrial defect datasets demonstrate that Super Sparse DETR achieves substantial acceleration while maintaining accuracy, laying a foundation for DETR to catch up with YOLO in industrial scenarios.

1. Introduction

In high-demand scenarios such as industrial defect detection, systems are often constrained by engineering metrics of **low latency** [4, 13, 26]. One-stage methods represented by the YOLO series [1, 8, 9, 27–29] excel in real-time performance but typically rely on post-processing Non-Maximum Suppression (NMS) [2, 17] and dense candidates, making the performance sensitive to thresholds and often leading to slower convergence. In contrast, **DETR (Detection Transformer)** [3] simplifies the detection pipeline and achieves end-to-end learning by Hungarian algorithm [14] which establishes a unique correspondence between predicted boxes and the ground-truth objects during training [25]. However, DETR suffers from slow



(a) Image (b) No sparse (c) Sparse

Figure 2. **Visualization of tokens without and with sparsity**. Sparse tokens focus on key regions, leading to faster inference.

convergence due to the inefficiency of Transformer in capturing local spatial information, and its global self-attention mechanism imposes high inference overhead. These factors collectively form key bottlenecks for DETR's practical industrial deployment.

To alleviate DETR's convergence and efficiency challenges, numerous studies have improved convergence and accuracy from training mechanisms: hybrid supervision [30, 33], O2M auxiliary assignment [5, 12, 32] and denseO2O [11] introduce denser positive sample during training to accelerate convergence but *without directly reducing inference complexity*. Relevant research [6] indicates that **structured sparsity** can bring significant practical acceleration on general hardware, making it the most direct path for engineering deployment. Consequently, works [23, 34] focus on token sparse, which accelerates convergence by sparse encoder token but fails to consider other components and cannot achieve deterministic acceleration after deployment. Especially on industrial defect datasets, due to more monotonous backgrounds, sparsity has greater potential for efficiency improvement , as shown in Fig. 2.

To address the convergence and efficiency issues with a more complete sparse system, we propose a novel Multi-factor scoring mechanism based on decoder-derived attention aggregation map (DAM) [23] and channel sensitivity (CS). It comprehensively considers efficiency and risk, realizing sparsity over tokens, queries, and Feed-Forward Networks (FFN) during training. To solve the problem of no acceleration after deployment, we stabilize the selections into the inference graph through skip-update during training and gather during export, achieving end-to-end deterministic acceleration. Although the above improvements partially address risk issues, the information drift risk caused by structural sparsity remains non-negligible. To resolve the instability and potential loss of useful information brought by sparsity, we introduce Consistency Distillation to smooth the turbulence and transfer potentially useful information. Specifically, we use the EMA (Exponential Moving Average) Teacher before pruning as the teacher model to impose consistency constraints on the pruned student model. We also incorporate a "concave-rebound" learning rate protection window and momentum preservation for smooth recovery.

2. Related Work

Dense Supervision DETR. To alleviate the insufficient supervision of Hungarian matching [14], DN-DETR [15] explicitly provides learnable positive and negative samples through denoising queries, significantly accelerating convergence and improving stability; Group-DETR [5] incorporates O2M into intra-group parallel learning to achieve denser positive sample supervision; Hybrid Matching [12] combines O2O and O2M assignments during training, balancing end-to-end inference and faster optimization dynamics; RT-DETR [19] introduces hierarchical dense positive samples and efficient backbones under real-time constraints; DEIM [11] further optimizes training signals from the perspectives of matching and loss, shortening the convergence trajectory. Orthogonally, our framework staticizes the "better selections" during training into the inference graph through "skip-update to gather," achieving deterministic deployment acceleration.

Sparse Structure DETR. Sparsity is another main direction for reducing the computational overhead of DETR. Sparse-DETR [23] prunes redundant attention under the end-to-end paradigm through learnable sparsity, improving training and inference efficiency; Focus-DETR [34] reduces invalid computations through focused attention and point sampling, reallocating computing power to key regions. These methods mainly achieve "looking less" in token selection, but the structural shape between training and inference is not strictly staticized, making it difficult to guarantee deterministic speed gains after deployment. Our approach dynamizes the sparse training phase and structurally staticizes them during export.

Structured Sparsity. A review work [6] provides a systematic collation and empirical suggestions from the perspectives of "where to prune, when to prune, and how to prune." In terms of metrics, structured pruning is more likely to be converted into real inference acceleration; in terms of timing, pruning during training (PDT) [21] enables simultaneous learning and pruning, reducing additional fine-tuning costs. [18] proves that pruning is a valid strategy for improving model performance. [7] further demonstrates that dynamic sparsity training has the potential to make model better and faster. To offset the representation drift caused by structural changes during training, we adopt Consistency Distillation with an EMA-teacher to perform "execute-then-compensate" after each structural adjustment.

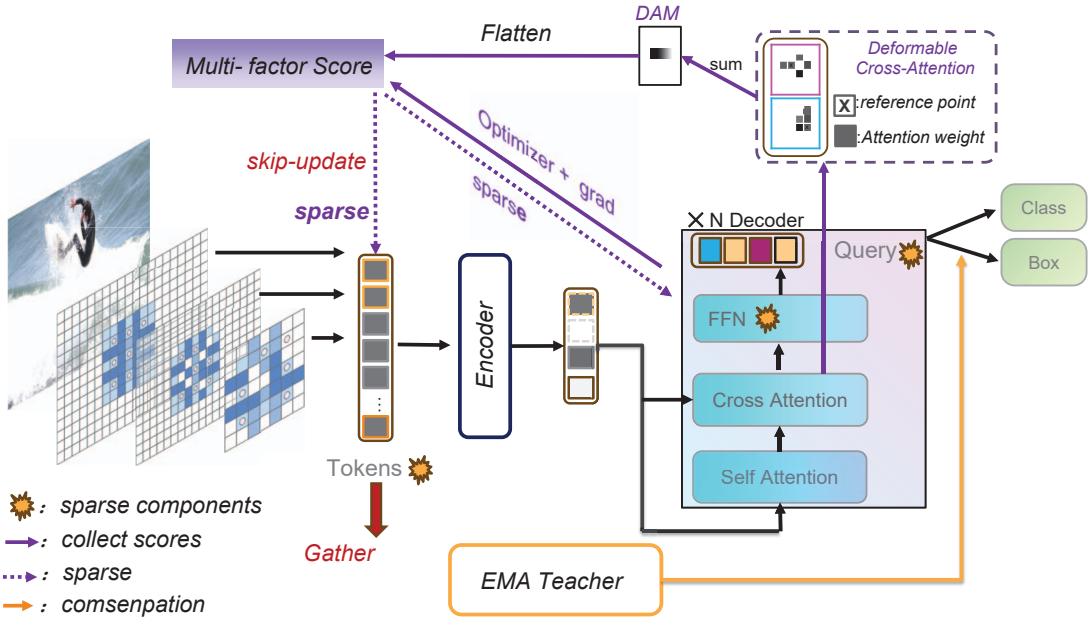


Figure 3. **Overall flow of our pipeline**: *collect* → *sparse* → *compensation*. Training leverages skip-update; export gathers indices to structurize computation.

3. Method

3.1. Preliminary

We view structured sparsity as a single pipeline from learnable dynamic compute at training to deterministic structural shrinkage at inference. Given image x and annotations y, we jointly learn detector parameters θ and three structural subsets:

$$\min_{\theta,\,\mathcal{S}_{\text{tok}},\,\mathcal{S}_{\text{qry}},\,\mathcal{S}_{\text{ffn}}} \mathbb{E}_{(x,y)}\Big[\mathcal{L}_{\text{det}}\big(\theta;\mathcal{S}_{\text{tok}},\mathcal{S}_{\text{qry}},\mathcal{S}_{\text{ffn}}\big)\Big] \quad \text{s.t.} \quad |\mathcal{S}_{\text{tok}}| = \rho_{\text{enc}} L,\;\; |\mathcal{S}_{\text{qry}}| = K,\;\; |\mathcal{S}_{\text{ffn}}| = \alpha C \tag{1}$$

where L is the encoder sequence length and C is the FFN hidden width; $(\rho_{\text{enc}}, K, \alpha)$ are budgets or retained ratio. For different datasets, we can adjust the budgets to control the sparsity of the model to obtain balance between acceleration and accuracy. The overall flow (Fig. 3) is concluded as follows: *Collect* → *Sparse* → *Compensation*, with training-time *skip-update* and export-time *gather*.

Explanation. Figure 3 summarizes the stage-wise pipeline. In the *collect* phase, we maintain EMA-smoothed multi-factor scores for tokens (u^{tok}), queries (u^{qry}), and FFN channels ($s^{\textbf{ffn}}$). In the *sparse* phase, structures are updated at milestones: tokens by per-level coverage-ratio, queries by Top-K with coverage, and FFN width by ratio α with per-layer minimum-keep. During training, skip-update gates incremental branches to avoid early shape shocks in encoder; at export, we gather indices of selected tokens, turning learned selections into deterministic reductions in FLOPs, memory and latency.

3.2. Collect and Sparse: Multi-factor Scoring

DAM. Sparse-DETR [23] and Focus-DETR [34] primarily apply DAM to sparsify tokens. DAM aggregates decoder cross-attention across layers into a usage map for each encoder feature level ℓ. Let $\mathbf{U}_\ell \in \mathbb{R}^{H_\ell \times W_\ell}$ denote this usage map. For Deformable DETR [35], DAM accumulates the attention weight of a query onto the bilinear neighbors of its sampled point:

$$\mathbf{U}_\ell(u,v) \propto \sum_{h,q,p} w_{\ell,h,q,p}\,\delta\big((u,v),\Pi(\mathbf{s}_{\ell,h,q,p})\big), \qquad u_i^{\textbf{tok}} = \mathbf{U}_{\ell(i)}(u_i,v_i) \tag{2}$$

where h is the attention head index, p is the sampling point index, and $w_{\ell,h,q,p}$ is the sampling point weight of deformable attention; $\mathbf{s}_{\ell,h,q,p}$ denotes the spatial coordinates of the sampling point, and $\Pi(\cdot)$ is a coordinate mapping function used to convert the sampling point coordinates to the corresponding position on the feature map; $\delta(\cdot)$ is an indicator function that outputs 1 if (u,v) matches $\Pi(\mathbf{s}_{\ell,h,q,p})$, and 0 otherwise. $u_i^{\textbf{tok}}$ is directly taken as the value at its spatial index (u_i,v_i) on the usage map of the corresponding feature level $\ell(i)$.

But prior work do not consider that industrial defect datasets admit further sparsification, which can yield addi-

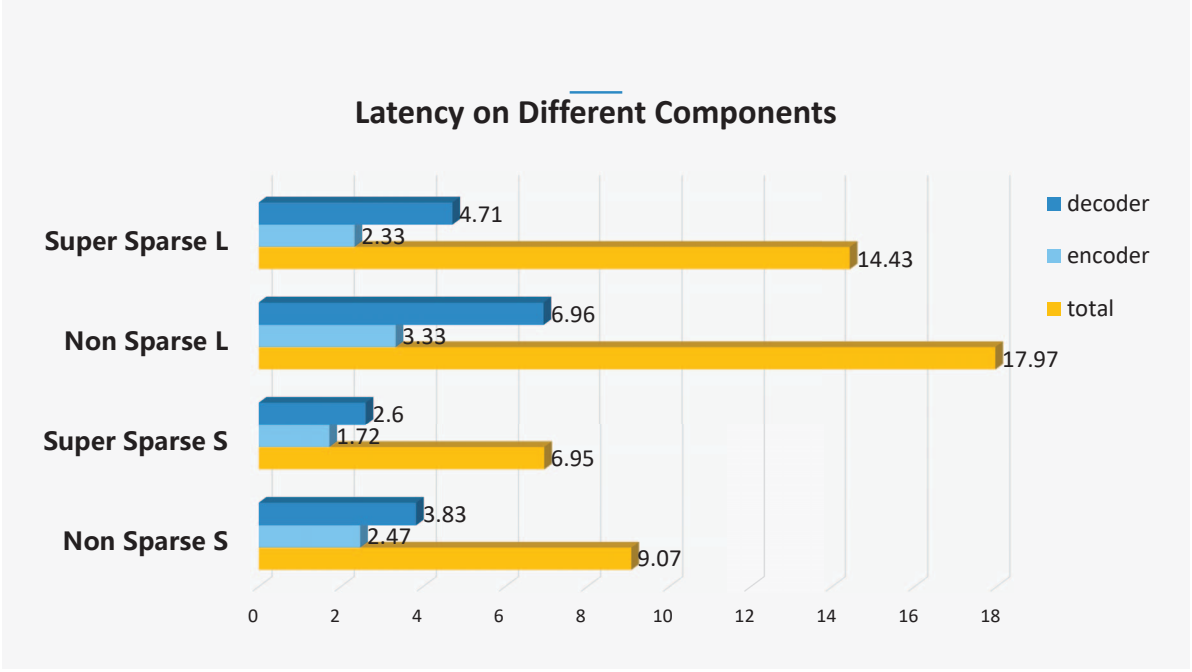


Figure 4. **Latency Analysis for Components.**

tional speedups. GPQ DETR [31] also shows that in sparsifying query can improve performance. Given that DAM is a direct estimator of which queries the model truly uses, query usage is a local pooling around the query's reference point:

$$u_j^{\mathbf{qry}} = \mathrm{AvgPool}_{3\times 3}\big(\mathbf{U}_{\ell(j)} \text{ around } ref(q_j)\big). \tag{3}$$

where 3×3 is the pooling window size, and $ref(q_j)$ is the reference point of query q_j projected to the coordinates of the corresponding feature level $\ell(j)$.

The core rationale behind this design lies in the fact that the decoder's supervision signal focuses on regions and queries that truly contribute to the set prediction task. As a direct estimator of this effective information, DAM can accurately preserve the key structures supporting model performance through the selection of u^{tok} and u^{qry}, while removing redundant non-contributors. This not only accelerates convergence but also reduces computational overhead, making it particularly suitable for industrial defect data characterized by concentrating effective regions and redundanting backgrounds.

Channel Sensitivity. Channel Sensitivity(CS) is used to measure the importance of FFN hidden channels [10, 24], providing a basis for FFN width sparsification. We adopt a gradient-based score that integrates parameter displacement and a Fisher-style proxy term, balancing the parameter stability during optimization and loss sensitivity. The specific calculation is as follows:

$$\begin{aligned} \Delta\theta_k &= -\frac{\hat{m}_k}{\sqrt{\hat{v}_k}+\epsilon}, \\ \mathbf{M}_k &= \mathrm{EMA}_\tau(|\Delta\theta_k|), \\ \mathbf{F}_k &= \mathrm{EMA}_\tau\big((\nabla_\theta \mathcal{L})^2\big), \end{aligned} \tag{4}$$

where θ_k represents the parameters corresponding to the k-th FFN hidden channel, $\nabla_\theta \mathcal{L}$ represents the gradient of the loss function with respect to the parameter θ_k, $\hat{m}_k$ and $\hat{v}_k$ are the first-order and second-order momentum estimates of gradients, respectively, ϵ is used to avoid division by zero; M_k accumulates the magnitude of parameter displacements, and F_k accumulates the squared loss gradients. For each FFN hidden channel k, its final importance score is obtained by weighted fusion of the normalized M and F, as shown in the formula below:

$$s_k^{\mathbf{ffn}} = \mathrm{EMA}_\tau\Big(\mathrm{Norm}(\mathbf{M}_k) \cdot \mathrm{Norm}(\mathbf{F}_k)\Big) \tag{5}$$

where, $\mathrm{Norm}(\cdot)$ refers to L2 normalization.

Multi-factor Scoring. Multi-factor Scoring maintains EMA-smoothed structure scores $\{u^{\text{tok}}, u^{\text{qry}}\}$ and width scores $\{s^{\mathbf{ffn}}\}$. Respectively, token is linked with encoder latency as well as query and FFN channel are linked with decoder. Unlike previous methods that only consider token sparsification to achieve a reduction in encoding latency, our method additionally takes into account the latency of

Algorithm 1 pipeline (*collect-sparse-compensation*).

1. $o_s \leftarrow \text{Model}(x;\theta)$; idx $\leftarrow \text{Hungarian}(o_s, y)$.
2. $\{\tilde{U}_\ell\} \leftarrow \text{AggregateDAM}(o_s)$ (stop-grad); $s^{ffn} \leftarrow \text{ChannelSensitivity}()$.
3. EMA-update for structure ($u^{\text{tok}}, u^{\text{qry}}$) and width ($s^{ffn}$).
4. If milestone and changed(): select tokens (per-level top-k with coverage) or queries (top-k) or FFN (prune ratio α); enable *skip-update*; set LR window; set flag_comp = True.
5. $L \leftarrow \mathcal{L}_{\text{det}}(o_s, y)$.
6. If flag_comp : $o_t \leftarrow \text{Model}(x;\theta^{\text{ema}})$; $L \leftarrow L + \lambda_{\text{cls}}\mathcal{L}^{\text{KD}}_{\text{cls}} + \lambda_{\text{box}}\mathcal{L}^{\text{KD}}_{\text{box}}$; set flag_comp = False.
7. Update θ; $\theta^{\text{ema}} \leftarrow \mu\,\theta^{\text{ema}} + (1-\mu)\theta$.

the decoder. Fig. 4 demonstrates that *with Super Sparse, the latency of both the encoder and the decoder are significantly reduced.* Furthermore, it can be observed from the figures that when the model scale is large, the latency of the decoder accounts for a substantial proportion, meaning sparsifying only the encoder yields negligible effects

3.3. Sparse: Learnable Sparse Computation and Structuralization

Skip-update (training). To avoid early-stage shape shocks, we gate only incremental branches without changing tensor shapes. For an encoder layer with input x and a binary token mask $\mathbf{M} \in \{0,1\}^{B\times L\times 1}$ (1=keep):

$$x \leftarrow x + \mathbf{M} \odot \Delta_{\text{attn}}(x) \tag{6}$$

leaving normalization paths intact.

Gather (export). After training converges, we freeze stable masks to index sets. We then gather encoder sequences so Q/K/V and attention operate on the short sequence $L' = \rho_{\text{enc}} L$; This converts learned selection into deterministic reductions of FLOPs. Complexity derivations are provided in the **Appendix**;

3.4. Compensation: EMA teacher

Distillation. Structure updates induce distribution shift. We distill only when a change occurs, using an EMA teacher with non-sparse structure. Given efficiency and memory considerations, we distill only the classification and box terms from the outputs of the final layer of decoder:

$$\begin{aligned}\mathcal{L}^{\text{KD}}_{\text{cls}} &= T^2\,\text{KL}\Big(\text{softmax}(\tfrac{z_t}{T}) \,\Big\|\, \log\text{softmax}(\tfrac{z_s}{T})\Big)\\ \mathcal{L}^{\text{KD}}_{\text{box}} &= \text{SmoothL1}\big(b_s;\, b_t\big).\end{aligned} \tag{7}$$

They are added directly to the detection loss with two weights:

$$\mathcal{L} = \mathcal{L}_{\text{det}} + \lambda_{\text{cls}}\mathcal{L}^{\text{KD}}_{\text{cls}} + \lambda_{\text{box}}\mathcal{L}^{\text{KD}}_{\text{box}}. \tag{8}$$

where z_t and z_s denote the teacher and student logits, b_t and b_s denote the corresponding bounding boxes, T is the distillation temperature and λ_{cls} and λ_{box} are scalar weights for the classification and box distillation losses, respectively.

LR Protection window. We adopt a milestone-driven loop with mutual exclusion across token or query or FFN. After each structural update we use a short *LR protection window* to smooth training.

4. Experiments

4.1. Experimental Setup

Dataset. We conduct experiments on the challenging COCO 2017 [16] detection dataset. To show the effectiveness of our methods, we also conduct experiments on the classic industrial defect dataset NEU-DET [20]. We follow the common practice and report the standard average precision on the COCO validation dataset and additional focus on efficiency on NEU-DET.

Implementation Details We implement our method on DEIM [11], leveraging its high-quality matching and fast convergence. We adopt Hgnetv2 as the backbone and set trained size and evaluated size at 640×640. To ensure the representativeness of the experimental results, we mainly present the **S**-scale model and the **L**-scale model. The former is primarily applied to scenarios with limited device conditions where speed is prioritized, while the latter is mainly used in complex scenarios where accuracy is prioritized. More detailed implementation details are provided in the **Appendix**.

4.2. Main Results

COCO. We report the main comparative results across real-time YOLO-based and DETR-based detectors. To demonstrate the effectiveness of our approach, we compare against recent state-of-the-art models which strike a strong balance between AP and efficiency. We also include representative YOLO-series detectors to highlight the efficiency of our method. Table 1 show our approach delivers markedly better efficiency than latest DETR variants and YOLO variants with nearly 25% drop in latency and no more than 0.5% drop in AP. This indicates that our method remains highly accurate while being efficient.

NEU-DET. Our primary motivation is to unlock the capabilities of DETR-style models for industrial inspection. Accordingly, we conduct detailed efficiency experiments on NEU-DET. Table 2 shows that our approach achieves significantly lower latency and improved accuracy. As discussed

Table 1. **Comparison with real-time object detectors on COCO 2017 [16].** ⋆ indicates that the NMS is tuned with a confidence threshold of 0.01. We measure end-to-end latency using TensorRT FP16 on an NVIDIA T4 GPU

Model	#Epochs	#Params	GFLOPs	Latency (ms)	mAP^{val}	AP^{val}_{50}	AP^{val}_{75}	AP^{val}_{S}	AP^{val}_{M}	AP^{val}_{L}
YOLO-based Real-time Object Detectors										
YOLOv8-S [8]	500	11	29	6.96	44.9	61.8	48.6	25.7	49.9	61.0
YOLOv8-L [8]	500	43	165	12.31	52.9	69.8	57.5	35.3	58.3	69.8
YOLOv9-S [28]	500	7	26	8.02	46.8	61.8	48.6	25.7	49.9	61.0
YOLOv9-C [28]	500	25	102	10.66	53.0	70.2	57.8	36.2	58.5	69.3
YOLOv10-S⋆ [27]	500	7	22	2.65	46.3	63.0	50.4	26.8	51.0	63.8
YOLOv10-L⋆ [27]	500	24	120	7.66	53.2	70.1	58.1	35.8	58.5	69.4
YOLOv11-S⋆ [9]	500	9	22	2.86	47.0	63.9	50.7	29.0	51.7	64.4
YOLOv11-L⋆ [9]	500	57	195	6.31	54.1	70.8	58.9	37.0	59.2	69.7
DETR-based Real-time Object Detectors										
D-FINE-S [22]	120	10	25	3.49	48.5	65.6	52.6	29.1	52.2	65.4
D-FINE-L [22]	72	31	91	8.07	54.0	71.6	58.4	36.5	58.0	71.9
Focus-DEIM-S [34]	120	10	23	3.09	47.1	65.8	53.5	28.3	51.0	62.0
Focus-DEIM-L [34]	50	31	86	7.49	53.5	71.5	57.1	35.2	58.0	71.1
DEIM-D-FINE-S [11]	120	10	25	3.49	49.0	65.9	53.1	30.4	52.6	65.7
DEIM-D-FINE-L [11]	50	31	91	8.07	54.7	72.4	59.4	36.9	59.6	71.8
Super Sparse DEIM-S	120	9	19	**2.61**	48.6	66.0	52.2	29.5	52.2	65.4
Super Sparse DEIM-L	50	30	75	**6.14**	54.2	72.3	57.6	35.3	58.6	72.0

Table 2. **Comparison with real-time object detectors on NEU-DET [20].** ⋆ indicates that the NMS is tuned with a confidence threshold of 0.01.

Model	#Epochs	#Params	GFLOPs	Latency (ms)	mAP^{val}	AP^{val}_{50}
YOLOv9-S [28]	300	7	26	8.02	32.0	66.5
YOLOv9-C [28]	300	25	102	10.66	33.9	67.2
YOLOv10-S⋆ [27]	300	7	22	2.65	36.0	70.0
YOLOv10-L⋆ [27]	300	24	120	7.66	40.5	71.1
Focus-DEIM-S [34]	120	10	23	3.09	39.4	69.7
Focus-DEIM-L [34]	50	31	86	7.49	41.0	72.5
DEIM-D-FINE-S [11]	120	10	25	3.49	39.0	71.0
DEIM-D-FINE-L [11]	50	31	91	8.07	40.7	72.4
Super Sparse DEIM-S	120	9	19	**2.61**	**40.2**	**71.4**
Super Sparse DEIM-L	50	30	75	**6.14**	**42.2**	**73.8**

earlier, industrial defect datasets differ markedly from conventional benchmarks: backgrounds are more uniform and objects are fewer, making sparse particularly valuable. We also find that L variants show limited improvement on such small datasets due to their tendency to overfit which indicates more suitable for sparse.

4.3. Ablation Studies

Effectiveness of Multi-factor score. We conduct experiments by adopting sparse by enabling sparsity in only one component to show each contribution. Table 3 shows that in NEU-DET, token sparsity has a negative impact on AP but great improve on latency. Table 4 shows that in COCO 2017, query sparsity has a serious impact on AP , while Token sparsity have positive effect on AP.

Table 3. **Ablation on Multi-factor score in NEU-DET [20].** ✓indicates the component is enabled.

Super Sparse DEIM-S					
Token	Query	FFN	mAP^{val}	AP^{val}_{50}	Latency (ms)
✓			39.9	69.7	2.97
	✓		40.5	71.7	2.95
		✓	40.9	72.6	3.30
✓	✓	✓	40.2	71.4	2.61

Distillation Compensation. To validate the effectiveness of compensation and the learning-rate (LR) protection window, we conduct comparisons on COCO and NEU-DET. Table 5 shows that compensation has a positive effect on

Table 4. **Ablation on Multi-factor score in COCO 2017 [16].** ✓indicates the component is enabled.

Super Sparse DEIM-S				
Token	Query	FFN	mAP^{val}	Latency (ms)
✓			48.8	2.97
	✓		47.3	2.95
		✓	48.4	3.30
✓	✓	✓	48.6	2.61

Table 5. **Ablation on Compensation in NEU-DET [20] and COCO 2017 [16].** ✓indicates the component is enabled.

Super Sparse DEIM-S			
EMA teacher	Dataset	mAP^{val}	Latency (ms)
✓	COCO 2017 [16]	48.6	2.70
	COCO 2017 [16]	47.0	2.70
✓	NEU-DET [20]	40.2	2.70
	NEU-DET [20]	39.3	2.70

Table 6. **Training time in GPU hours in NEU-DET [20].**

Super Sparse DEIM-S			
Model	epoch	mAP^{val}	GPU hours
YOLOv10-S* [27]	300	36.0	1.10
DEIM-D-FINE-S [11]	110	39.0	1.24
Super Sparse DEIM-S	110	40.2	1.08

AP especially in COCO 2017.

Training time in GPU hours. We conduct experiments by comparing training time in GPU hours on a single RTX 3090ti GPU with batch size = 24. Table 6 shows that our approach achieves significantly shorter training time compared to other methods. Our method saves 20% more time compared with the original baseline.

4.4. Analysis

Sparsity components. Table 3 and Table 4 show that for large and complex datasets like COCO 2017, token sparsification yields the greatest benefits, as it can filter out irrelevant backgrounds and greatly accelerate processing speed. In contrast, query sparsification has the most significant impact on AP, because large datasets often contain an extremely large number of categories, and an insufficient number of queries will fail to cover the objects present in the samples. For small datasets such as industrial defect datasets, however, query sparsification and FFN pruning yield the greatest benefits for AP. This is because small datasets have fewer categories, and an excessive number of queries will instead lead to an increase in negative samples, resulting in deviation in the learning direction.

Effectiveness of compensation. From the comparison of results before and after applying the compensation mechanism in Table 5, it is evident that the compensation mechanism is crucial for large datasets. This is because the model itself has a limited capacity to carry knowledge; when sparsification is implemented while a large amount of experience needs to be recorded, it is easy to lose relevant information. Additionally, our method only compensates for the output of the last layer of the decoder layer under the consideration of memory and training efficiency. The insufficient compensation intensity may lead to a decrease in AP on COCO

Sparsifying During Training. From above experiments and relative analysis [6], sparsifying after training costs more time both in training (model is heavy) and postprocessing; sparsifying before training easily loses potential information. Though we employ Multi-factor score to collect scores and EMA teacher distillation to compensate, model is becoming more increasingly lightweight during training, indicating sparsifying during training can alleviate optimization burden and transferring useful information to the sparsified model, providing a good trade-off between efficiency and performance.

4.5. Limitation and Future Directions

Despite the significant efficiency gains, our approach still relies on deformable attention, whose CUDA implementation is not supported on some edge devices. This in part explains why YOLO-series detectors remain more widely deployed than DETR. To address this limitation and broaden applicability, future work will explore replacing deformable attention with operators available on edge platforms, such as RoIAlign, enabling wider deployment of DETR-style models.

5. Conclusion

This work proposes Super Sparse DETR, a structured sparsity training framework tailored for DETR's industrial deployment. By integrating DAM and CS into a multi-factor scoring mechanism, combined with skip-update, export-time gather, and EMA teacher distillation, the framework achieves deterministic acceleration while preserving detection accuracy. This work addresses DETR's core bottlenecks in efficiency and industrial adaptability, laying a solid foundation for DETR-style models to compete with YOLO in real-world deployment scenarios.

References

[1] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4: Optimal speed and accuracy of object detection. *arXiv*, 2020. 1

[2] Navaneeth Bodla, Bharat Singh, Rama Chellappa, and Larry S. Davis. Soft-nms—improving object detection with one line of code. In *ICCV*, 2017. 1

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

[4] Anthony Ashwin Peter Chazhoor, Edmond S.L.Ho, Bin Gao, and Wai Lok Woo. A review and benchmark on state-of-the-art steel defects detection. *SN Computer Science*, 2023. 1

[5] Qiang Chen, Xiaokang Chen, Jian Wang, Shan Zhang, Kun Yao, Haocheng Feng, Junyu Han, Errui Ding, Gang Zeng, and Jingdong Wang. Group detr: Fast detr training with group-wise one-to-many assignment. In *ICCV*, 2023. 2

[6] Hongrong Cheng, Miao Zhang, and Javen Qinfeng Shi. A survey on deep neural network pruning—taxonomy, comparison, analysis, and recommendations. *TPAMI*, 2024. 2, 7

[7] Utku Evci, Trevor Gale, Jacob Menick, Pablo Samuel Castro, and Erich Elsen. Rigging the lottery: Making all tickets winners. In *ICML*, 2020. 2

[8] Jocher Glenn. Yolov8. GitHub: Ultralytics/ultralytics, 2023. 1, 6

[9] Jocher Glenn. Yolov11. GitHub: Ultralytics/ultralytics, 2024. 1, 6

[10] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *ICCV*, 2017. 4

[11] Shihua Huang, Zhichao Lu, Xiaodong Cun, Yongjun Yu, Xiao Zhou, and Xi Shen. Deim: Detr with improved matching for fast convergence. In *CVPR*, 2025. 2, 5, 6, 7

[12] Ding Jia, Yuhui Yuan, Haodi He, Xiaopei Wu, Haojun Yu, Weihong Lin, Lei Sun, Chao Zhang, and Han Hu. Detrs with hybrid matching. In *CVPR*, 2023. 2

[13] Ramona Kühlechner. Object detection survey for industrial applications with focus on quality control. *Production Engineering*, 2025. 1

[14] Harold W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 1955. 1, 2

[15] 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

[16] 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. 1, 5, 6, 7

[17] Shanshan Liu, Di Huang, and Yunhong Wang. Adaptive nms: Refining pedestrian detection in a crowd. In *CVPRW*, 2019. 1

[18] Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the value of network pruning. In *ICLR*, 2019. 2

[19] Wenyu Lv, Yian Zhao, Qinyao Chang, Kui Huang, Guanzhong Wang, and Yi Liu. Rt-detrv2: Improved baseline with bag-of-freebies for real-time detection transformer. In *arXiv*, 2024. 2

[20] Xiang Lv, Fajie Duan, Jia jia Jiang, Xiao Fu, and Lin Gan. Deep metallic surface defect detection: The new benchmark and detection network. *Sensors*, 2020. 1, 5, 6, 7

[21] Decebal Constantin Mocanu, Elena Mocanu, Peter Stone, Phuong H Nguyen, Madeleine Gibescu, and Antonio Liotta. Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science. *Nature Communications*, 2018. 2

[22] Yansong Peng, Hebei Li, Peixi Wu, Yueyi Zhang, Xiaoyan Sun, and Feng Wu. D-fine: Redefine regression task in detrs as fine-grained distribution refinement. In *ICLR*, 2024. 6

[23] Byungseok Roh, Jaewoong Shin, Wuhyun Shin, and Saehoon Kim. Sparse detr: Efficient end-to-end object detection with learnable sparsity. In *ICLR*, 2022. 2, 3

[24] Victor Sanh, Thomas Wolf, and Alexander M.Rush. Movement pruning: Adaptive sparsity by fine-tuning. In *NeurIPS*, 2020. 4

[25] Peize Sun, Yi Jiang, Rufeng Zhang, Xin Yuan, Zehuan Yuan, Ping Luo, and Jiaya Jia. What makes for end-to-end object detection? In *ICML*, 2021. 1

[26] Sebastian Trinks. Real time quality assurance and defect detection in industry 4.0. In *LWDA*, 2021. 1

[27] Ao Wang, Hui Chen, Lihao Liu, Kai Chen, Zijia Lin, Jungong Han, and Guiguang Ding. Yolov10: Real-time end-to-end object detection. In *NeurIPS*, 2024. 1, 6, 7

[28] Chien-Yao Wang, I-Hau Yeh, and Hong-Yuan Mark Liao. Yolov9: Learning what you want to learn using programmable gradient information, 2024. 6

[29] Ling Wang, Xinbo Liu, Juntao Ma, Wenzhi Su, and Han Li. Real-time steel surface defect detection with improved multi-scale yolo-v5. *Processes*, 2023. 1

[30] Shuo Wang, Chunlong Xia, Feng Lv, and Yifeng Shi. Rt-detrv3: Real-time end-to-end object detection with hierarchical dense positive supervision, 2024. 2

[31] Lizhen Xu, Zehao Wu, Wenzhao Qiu, Shanmin Pang, Xiuxiu Bai, Kuizhi Mei, and Jianru Xue. Redundant queries in detr-based 3d detection methods: Unnecessary and prunable, 2024. 4

[32] Chang-Bin Zhang, Yujie Zhong, and Kai Han. Mr. detr: Instructive multi-route training for detection transformers. In *CVPR*, 2025. 2

[33] Chuyang Zhao, Yifan Sun, Wenhao Wang, Qiang Chen, Errui Ding, Yi Yang, and Jingdong Wang. Ms-detr: Efficient detr training with mixed supervision. In *CVPR*, 2024. 2

[34] Dongsheng Zheng, Weixuan Dong, Hailin Hu, Xinghao Chen, and Yunhe Wang. Less is more: Focus attention for efficient detr. In *ICCV*, 2023. 2, 3, 6

[35] 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. 3