Accelerate 3D Object Detection Models via Zero-Shot Attention Key Pruning

Supplementary Material

A. Query-based 3D Detectors

A.1. Overall Architecture

DETR-based methods have become the mainstream approach for 3D object detection. As shown in Fig. 8 (a), these methods typically take multi-view images as input and use an image backbone to extract image features F. Subsequently, F is fed into a transformer decoder along with predefined queries Q for interaction. Represented by PETR, **dense methods** adopt global attention, allowing Q to interact with F globally. In contrast, **sparse methods** such as DETR3D and Far3D employ deformable attention, selecting only a subset of F for interaction with Q. The output of the Transformer Decoder maintains the same shape as the predefined queries Q. It is passed to classification branches and regression branches to obtain classification scores and bounding boxes B. The bounding boxes B not only contain object location information but also include object size, orientation, velocity, and additional attributes (e.g., whether a pedestrian is standing, walking, or sitting). During training, DETR-based methods use a bipartite matching strategy to associate predictions with ground truth and compute classification and localization losses. Since ground truth is unavailable during inference, the model selects the final predictions based on classification scores.

The Transformer decoder used in 3D detectors has a structure of self-attention followed by cross-attention. The self-attention module of the first layer takes pre-defined query Q as input, and its output, together with image features, will be fed into the following cross-attention module. The output of the cross-attention module will be taken as input by the next layer's self-attention module. Our method uses the attention map generated by the cross-attention modules.

A.2. Advantages of Dense Methods over Sparse Methods

Since the introduction of DETR, DETR-based methods have gradually gained prominence in various vision tasks, including image classification, 2D object detection, 3D object detection, semantic segmentation, and object tracking. However, DETR-based methods also have several drawbacks, such as high computational cost and slow inference speed. One major issue is their slow convergence rate. DETR requires training for several hundred epochs on the COCO dataset before convergence. To address this, DeformableDETR was proposed. By utilizing reference points for local feature sampling, DeformableDETR significantly speeds up convergence, requiring only 50 epochs on the COCO dataset to achieve the same performance as DETR trained for 500 epochs. In the 3D object detection domain, sparse methods have been introduced to accelerate model convergence while also reducing the computational burden of the decoder to some extent.

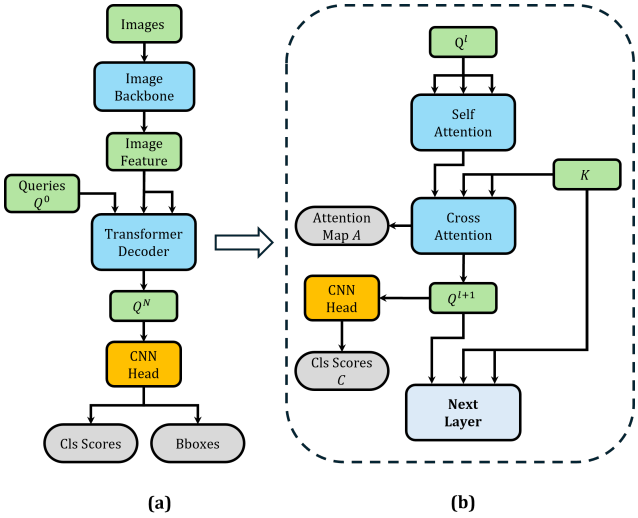


Figure 8. **(a)** Overall architecture of query-based 3D detectors. All dense models share the same overall structure: an image backbone followed by a transformer decoder. The transformer decoder consists of multiple stacked transformer layers. **(b)** The transformer layer used in the transformer decoder in (a). Each layer has a self-attention module and a cross-attention module. A CNN head can be used following the cross-attention module, which inputs updated queries and outputs the classification scores. We use attention map $A \in \mathbb{R}^{N_q \times N_k}$ and classification scores $C \in \mathbb{R}^{N_q}$ to calculate each key's importance score.

However, our comparison reveals that the overall model does not achieve a significant increase in inference speed, nor does it reduce memory consumption. In some cases, sacrificing performance necessitates additional compensatory mechanisms, introducing extra parameters that increase memory usage.

We selected state-of-the-art dense and sparse methods, OPEN and SparseBEV, respectively, for comparison. From the comparison in Tab. 6, it can be observed that when using the same backbone and image resolution, SparseBEV performs worse than OPEN on the nuScenes validation set. In terms of memory usage, during inference, SparseBEV consumes more memory than OPEN when using the same backbone and image resolution. Regarding inference speed, SparseBEV does not show a significant advantage over OPEN.

Model	Backbone	ImageSize	mAP ↑	NDS ↑	mATE ↓	mASE ↓	mAOE ↓	mAVE ↓	mAAE ↓	Mem. (MiB) ↓	Inf. Time (ms) ↓
SparseBEV	ResNet50	704x256	45.45%	0.5559	0.5984	0.2706	0.4124	0.2435	0.1865	5640	77.72
	ResNet101	1408x512	50.12%	0.5920	0.5621	0.2648	0.3211	0.2427	0.1947	15386	193.25
OPEN	ResNet50	704x256	47.02%	0.5657	0.5676	0.2702	0.4221	0.2321	0.2019	4062	77.25
	ResNet101	1408x512	51.80%	0.6043	0.5314	0.2679	0.3457	0.2095	0.1922	10696	196.55

Table 6. Comparison of SparseBEV and OPEN.

In summary, while current dense methods converge more slowly than sparse methods, they offer advantages in performance and memory usage, without showing a clear disadvantage in inference speed. Due to their strong overall advantages, new dense methods continue to emerge. This further highlights the importance of our work.

B. Analysis of Computation Cost of Importance Scores

The FLOPs of Matrix Multiplication. Assume there are two matrices $A \in \mathbb{R}^{N\times C}$, $B \in \mathbb{R}^{M\times C}$. When computing $AB^T \in \mathbb{R}^{N\times M}$, each row $A_i \in \mathbb{R}^C$ of A is multiplied by each column $B_j \in \mathbb{R}^C$ of B^T, which requires C multiplications and $C-1$ additions. Since A has N rows and B^T has M columns, the total computational complexity is:

$$\begin{aligned} F_{\text{mat\_mul}} &= N \times M \times (C + (C-1)) \\ &= N \times M \times (2C-1) \end{aligned} \tag{11}$$

The FLOPs of a Multi-head Attention. Given the input $Q \in \mathbb{R}^{N_q\times E}$, $K \in \mathbb{R}^{N_k\times E}$, and $V \in \mathbb{R}^{N_k\times E}$, a multi-head attention module with H heads includes the operations shown in Eq. (1)-Eq. (4).

In Eq. (1), the operations involve matrix multiplications: one between $Q \in \mathbb{R}^{N_q\times E}$ and $\mathbb{R}^{E\times E}$, and two between $K, V \in \mathbb{R}^{N_k\times E}$ and $\mathbb{R}^{E\times E}$, with a total FLOPs:

$$N_k \cdot (4E^2 - 2E) + 2N_qE^2 - N_qE \tag{12}$$

In Eq. (3), where $q^h \in \mathbb{R}^{N_q\times E_h}, k^h \in \mathbb{R}^{N_k\times E_h}$:

1) It first calculates a matrix multiplication $q^h \times (k^h)^T$ with FLOPs $N_qN_k(2E_h - 1)$. For total H heads, FLOPs are:

$$N_k \cdot (2N_qE - N_qH) \tag{13}$$

2) Next, a square root is performed to get $\sqrt{E}$ with FLOPs 1.

3) Each element of $\left(q^h \times (k^h)^T\right)$ divides $\sqrt{E}$, resulting in FLOPs of N_qN_k. For total H heads, FLOPs are:

$$N_k \cdot N_qH \tag{14}$$

4) For the Softmax operation with a vector of shape $\mathbb{R}^N$ as input, it performs N exponentiations, $N-1$ additions and N divisions. Hence, Softmax $(\cdot)$ in Eq. (3) has $N_q(3N_k-1)$ FLOPs. For total H heads, FLOPs are:

$$N_k \cdot 3N_qH - N_qH \tag{15}$$

5) Next, $O^h = A^h \times v^h$ has $N_qE_h(2N_k - 1)$ FLOPs. For total H heads, FLOPs are:

$$N_k \cdot 2N_qE - N_qE \tag{16}$$

Hence, Eq. (3) and $O^h = A^h \times v^h$ has FLOPs:

$$N_k \cdot (4N_qE + 3N_qH) - N_qH - N_qE + 1 \tag{17}$$

Finally, Eq. (4) has FLOPs:

$$2N_qE^2 - N_qE \tag{18}$$

In summary, according to Eq. (12), Eq. (17) and Eq. (18), a single multi-head attention module with H heads contains FLOPs as follows:

$$\begin{aligned} F_{\text{mha}}(N_k) &= \lambda N_k + b \\ \lambda &= 4E^2 - 2E + 4N_qE + 3N_qH \\ b &= 4N_qE^2 - 3N_qE - N_qH + 1 \end{aligned} \tag{19}$$

A transformer layer contains a self-attention module, a cross-attention module, two layer normalizations and a feed-forward network. Among these, only cross-module is related to *key* with $Q \in \mathbb{R}^{N_q\times E}, K, V \in \mathbb{R}^{N_k\times E}$ as inputs. Hence, its FLOPs are:

$$F_{\text{CA}}(N_k) = F_{\text{mha}}(N_k) \tag{20}$$

Cost of Computing Importance Scores. To calculate importance scores, we first need to calculate averaged attention maps A, which involves $N_qN_k(H-1)$ additions and N_qN_k divisions, with total FLOPs N_qN_kH. Next, we calculate $S_0 = A \odot \tilde{C}$, which contains $N_q \times N_k$ multiplications. After selection, we compute the sum along the column of S_1(Eq. (9)), which needs $N_k(k-1)$ additions. In total, the FLOPs of calculating importance scores are:

$$F_{\text{S}} = N_qN_kH + N_qN_k + N_k(k-1) \tag{21}$$

Considering a transformer module with L layers applying tgGBC with r, n, k.

The FLOPs before pruning:

$$\text{F}_{\text{before}} = L \cdot F_{\text{CA}}(N_k) \tag{22}$$

Model	Backbone	ImageSize	N_k	r	GFLOPs	Reduced FLOPs
StreamPETR	VovNet	1600x640	24000	-	174.91	-
				21000	61.44	-64.88%
OPEN	ResNet101	1408x512	16896	-	123.55	-
				12000	58.84	-52.37%

Table 7. FLOPs reduced of the transformer module after pruning. We show the results of StreamPETR and OPEN with $n = 2$, $k = 175$ for both models. 1 GFLOPs = 10^9 FLOPs.

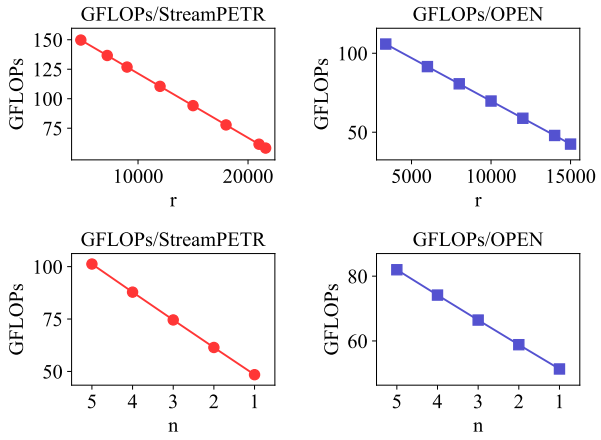


Figure 9. FLOPs decrease with the increase r and the decrease of n. StreamPETR (OPEN) shown in the figure uses VovNet (ResNet101) and an image size of 1600×640 (1408×512). For the first line, both models use n=2 and $k = 175$. For the second line, StreamPETR (OPEN) uses $k = 175$ and r=21000 (12000).

The FLOPs after pruning:

$$\begin{aligned} \mathrm{F}_{\text{after}} = & \sum_{i=0}^{n} F_{\text{CA}}\left(N_k - i \cdot \frac{r}{n}\right) \\ & + \max(0, L - (n+1)) \cdot F_{\text{CA}}(N_k - r) \\ & + \sum_{i=0}^{n-1} F_{\text{S}}\left(N_k - i \cdot \frac{r}{n}\right) \end{aligned} \tag{23}$$

For different models, we calculate FLOPs before and after pruning as exhibited in Tab. 7.

According to Eq. (23), the FLOPs decrease linearly with the increase of r and the decrease of n, as shown in Fig. 9.

As described in Sec. 4.5, the impact of k on inference time is negligible. This is because when k varies within the range $[1, N_q]$, its effect on FLOPs is minimal, as shown in Fig. 10. No matter how k changes, the computational cost of the Cross-Attention module varies by no more than 0.1 GFLOPs. Compared to the effects of r and n, the influence of k on inference time can be ignored.

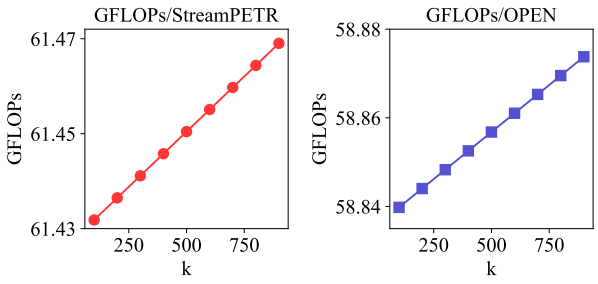


Figure 10. When k changes, there is almost no change in FLOPs.

C. More Results

C.1. Optimal Results

As described in Sec. 4.2, to facilitate clearer comparison and save space, we uniformly report the results with $n = 2$ and $k = 175$. However, this is not the optimal configuration for every model. Here, the optimal configuration refers to the maximum value of r and the minimum value of n that can be applied when the mAP and NDS decrease by no more than 1%. For the vast majority of models, we can directly use $n = 1$ for pruning, as shown in Tab. 8. For models that have already achieved optimal results, as shown in Tab. 1 (e.g., PETR-vov, FocalPETR, MV2D, StreamPETR-vov), we do not repeat the results here.

C.2. More Comparison to ToMe

As described in Sec. 4.3, our method is the first to apply zero-shot pruning to 3D object detection models. Therefore, there are no similar methods available for a fair comparison. However, we can attempt to transfer zero-shot pruning methods from the Vision Transformer (ViT) domain to 3D object detection models. Among these methods, ATS relies on a classification token specifically designed for classification tasks, and Zero-TPrune depends on a square-shaped attention map. Neither of these features exists in 3D object detection methods, making it impossible to transfer ATS and Zero-TPrune to 3D object detection. In contrast, ToMe does not have a strong dependency on the shape of the attention map or the classification token. Therefore, we selected ToMe as the comparison method.

As shown in the Tab. 9, for PETR and MV2D, ToMe performs extremely poorly—when pruning only 50% of the keys, the model's performance completely collapses, with mAP dropping by more than 10%. In contrast, tgGBC can maintain model performance. Even when $n = 1$ and some results are suboptimal, with mAP dropping by more than 1%, the model does not degrade into an unusable state.

Model	Backbone	ImageSize	N_k	r	n	k	mAP ↑	NDS ↑	mATE ↓	mASE ↓	mAOE ↓	mAVE ↓	mAAE ↓	Inf. Time(ms) ↓
PETR	ResNet50	1408x512	16896	-	-	-	31.74%	0.3669	0.8392	0.2797	0.6145	0.9521	0.2322	131.94
				10000	1	175	30.78%	0.3579	0.8540	0.2832	0.6168	0.9703	0.2354	110.35(-16.36%)
PETRv2	VovNet	800x320	12000	-	-	-	41.05%	0.5024	0.7232	0.2692	0.4529	0.3896	0.1932	157.52
				8000	2	175	40.29%	0.4918	0.7333	0.2750	0.4526	0.4428	0.1930	139.33(-11.55%)
StreamPETR	ResNet50	704x256	4224	-	-	-	38.01%	0.4822	0.6781	0.2763	0.6401	0.2831	0.2007	60.52
				2000	1	900	37.96%	0.4822	0.6828	0.2757	0.6373	0.2794	0.2009	51.59(-14.76%)
3DPPE	VovNet	800x320	6000	-	-	-	39.81%	0.4460	0.7040	0.2699	0.4951	0.8438	0.2177	125.13
				3000	1	175	39.57%	0.4430	0.7089	0.2729	0.4974	0.8496	0.2201	94.09(-24.81%)
M-BEV	VovNet	800x320	12000	-	-	-	35.14%	0.4640	0.7300	0.2717	0.4980	0.4324	0.1845	178.87
				6000	1	175	34.91%	0.4600	0.7379	0.2727	0.5049	0.4428	0.1873	150.93(-15.63%)
OPEN	ResNet50	704x256	4224	-	-	-	47.02%	0.5657	0.5676	0.2702	0.4221	0.2321	0.2019	77.25
				2000	1	900	46.88%	0.5636	0.5687	0.2721	0.2332	0.2315	0.2026	63.31(-18.05%)
	VovNet	800x320	6000	-	-	-	52.07%	0.6128	0.5250	0.2566	0.2811	0.2148	0.1982	118.55
				3000	1	175	52.12%	0.6130	0.5250	0.2569	0.2810	0.2148	0.1985	110.21(-7.04%)
	ResNet101	1408x512	16896	-	-	-	51.80%	0.6043	0.5314	0.2679	0.3457	0.2095	0.1922	196.55
				12000	1	175	51.47%	0.6018	0.5356	0.2691	0.3446	0.2131	0.1931	174.70(-11.16%)
ToC3D	ToC3DViT	1600x800	30000	-	-	-	54.20%	0.6187	0.5589	0.2571	0.2716	0.2353	0.2007	863.47
				27000	1	900	53.31%	0.6121	0.5736	0.2591	0.2713	0.2381	0.2029	809.48(-6.25%)

Table 8. Optimal results for each model.

Model	Backbone	ImageSize	Pruning	r	n	mAP ↑	NDS ↑	mATE ↓	mASE ↓	mAOE ↓	mAVE ↓	mAAE ↓
PETR	ResNet50	1408x512	-	-	-	31.74%	0.3669	0.8392	0.2797	0.6145	0.9521	0.2322
			ToMe	8000	2	30.48%	0.3543	0.8708	0.2816	0.6091	0.9852	0.2342
				8000	1	29.81%	0.3490	0.8714	0.2823	0.6173	0.9878	0.2418
				12000	2	29.63%	0.3461	0.8859	0.2836	0.6198	0.9895	0.2412
			Ours	8000	2	31.22%	0.3639	0.8436	0.2808	0.6133	0.9517	0.2329
				8000	1	31.58%	0.3651	0.8418	0.2801	0.6148	0.9576	0.2330
				12000	2	30.78%	0.3579	0.8540	0.2832	0.6168	0.9703	0.2354
	VovNet	1600x640	-	-	-	40.45%	0.4517	0.7287	0.2706	0.4485	0.8399	0.2178
			ToMe	12000	2	21.88%	0.2835	0.9517	0.4471	0.5913	1.0230	0.2688
				12000	1	27.17%	0.3350	0.8710	0.3560	0.5512	0.9959	0.2342
				18000	2	26.58%	0.3162	0.9143	0.4300	0.5663	1.0210	0.2564
			Ours	12000	2	40.42%	0.4502	0.7305	0.2702	0.4501	0.8512	0.2172
				12000	1	39.37%	0.4425	0.7429	0.2722	0.4592	0.8517	0.2174
				18000	2	39.53%	0.4432	0.7482	0.2720	0.4538	0.8539	0.2167
3DPPE	VovNet	800x320	-	-	-	39.81%	0.4460	0.7040	0.2699	0.4951	0.8438	0.2177
			ToMe	2000	2	39.56%	0.4432	0.7083	0.2715	0.4972	0.8490	0.2198
					1	36.99%	0.4197	0.7500	0.2719	0.5337	0.8735	0.2233
			Ours	2000	2	39.74%	0.4449	0.7057	0.2707	0.4956	0.8465	0.2202
					1	39.57%	0.4430	0.7089	0.2729	0.4974	0.8496	0.2201
MV2D	ResNet50	1408x512	-	-	-	44.92%	0.5399	0.6246	0.2657	0.3840	0.4009	0.1722
			ToMe	50%	2	13.62%	0.2754	0.9028	0.3472	0.7737	0.6777	0.2261
					1	13.45%	0.2721	0.9000	0.3477	0.7890	0.6864	0.2286
			Ours	50%	2	44.11%	0.5384	0.6248	0.2657	0.3844	0.4024	0.1717
					1	41.17%	0.5183	0.6289	0.2693	0.3887	0.4165	0.1717

Table 9. More results of comparison to ToMe [1]. Due to the use of bipartite matching, ToMe cannot prune more than 50% of keys in one layer; hence, some results of $r > N_k/2$ are lacking.

C.3. Comparison to More Baselines

We also add comparisons with DART[37] and random pruning in Tab. 11 to demonstrate the effectiveness of tgGBC.

C.4. More Ablation Experiments

We report additional results for different values of r, n, k in Tab. 10. As discussed in the main text Sec. 4.2, the optimal parameter selection varies across different models. If we

Model	Backbone	ImageSize	r	n	k	mAP ↑	NDS ↑	mATE ↓	mASE ↓	mAOE ↓	mAVE ↓	mAAE ↓
StreamPETR	ResNet50	704x256	-	-	-	38.01%	0.4822	0.6781	0.2763	0.6401	0.2831	0.2007
			2000	5	150	38.04%	0.4824	0.6787	0.2760	0.6388	0.2838	0.2007
				4	175	38.04%	0.4826	0.6795	0.2761	0.6367	0.2826	0.2014
				3	150	37.91%	0.4813	0.6819	0.2761	0.6385	0.2832	0.2025
				2	175	37.91%	0.4817	0.6787	0.2758	0.6390	0.2844	0.2016
				1	900	37.96%	0.4822	0.6828	0.2757	0.6373	0.2794	0.2009
	VovNet	1600x640	-	-	-	48.89%	0.5732	0.6096	0.2601	0.3882	0.2603	0.1944
			12000	5	900	48.92%	0.5734	0.6089	0.2603	0.3884	0.2601	0.1942
				4	900	48.90%	0.5735	0.6082	0.2601	0.3871	0.2598	0.1946
				3	900	48.89%	0.5736	0.6076	0.2599	0.3870	0.2595	0.1940
				2	900	48.95%	0.5741	0.6074	0.2605	0.3858	0.2594	0.1937
				1	175	48.85%	0.5738	0.6078	0.2603	0.3813	0.2613	0.1941
			18000	5	900	48.95%	0.5743	0.6059	0.2605	0.3833	0.2602	0.1944
				4	900	48.94%	0.5744	0.6064	0.2604	0.3821	0.2600	0.1943
				3	900	48.67%	0.5738	0.6054	0.2617	0.3701	0.2627	0.1955
				2	900	48.85%	0.5738	0.6054	0.2617	0.3701	0.2627	0.1955
				1	900	48.62%	0.5734	0.6087	0.2613	0.3705	0.2628	0.1942
			21000	5	900	48.92%	0.5742	0.6063	0.2604	0.3820	0.2611	0.1942
				4	900	48.85%	0.5756	0.6015	0.2614	0.3659	0.2616	0.1966
				3	175	48.21%	0.5702	0.6089	0.2619	0.3751	0.2669	0.1957
				2	900	48.71%	0.5731	0.6052	0.2622	0.3822	0.2618	0.1933
				1	900	47.87%	0.5695	0.6071	0.2635	0.3625	0.2692	0.1952
OPEN	ResNet50	704x256	-	-	-	47.02%	0.5657	0.5676	0.2702	0.4221	0.2321	0.2019
			2000	5	900	46.98%	0.5648	0.5669	0.2706	0.4303	0.2319	0.2015
				4	900	47.03%	0.5649	0.5680	0.2705	0.4296	0.2321	0.2017
				3	900	47.02%	0.5642	0.5685	0.2706	0.4345	0.2324	0.2028
				2	175	46.85%	0.5637	0.5682	0.2705	0.4311	0.2325	0.2031
				1	900	46.88%	0.5636	0.5687	0.2721	0.4332	0.2315	0.2026
	VovNet	800x320	-	-	-	52.07%	0.6128	0.5250	0.2566	0.2811	0.2148	0.1982
			3000	5	175	52.07%	0.6126	0.5261	0.2566	0.2818	0.2147	0.1985
				4	175	52.06%	0.6124	0.5267	0.2566	0.2823	0.2144	0.1987
				3	175	52.08%	0.6127	0.5254	0.2565	0.2814	0.2149	0.1986
				2	175	52.09%	0.6129	0.5249	0.2569	0.2808	0.2146	0.1986
				1	175	52.12%	0.6130	0.5250	0.2569	0.2810	0.2148	0.1985
	ResNet101	1408x512	-	-	-	51.80%	0.6043	0.5314	0.2679	0.3457	0.2095	0.1922
			10000	5	900	51.78%	0.6033	0.5339	0.2694	0.3509	0.2101	0.1916
				4	900	51.75%	0.6029	0.5333	0.2693	0.3544	0.2109	0.1907
				3	900	51.58%	0.6021	0.5372	0.2695	0.3496	0.2107	0.1914
				2	900	51.63%	0.6017	0.5386	0.2713	0.3542	0.2102	0.1906
				1	900	51.68%	0.6032	0.5357	0.2691	0.3443	0.2110	0.1921
			12000	5	900	51.75%	0.6030	0.5346	0.2698	0.3525	0.2109	0.1901
				4	900	51.78%	0.6024	0.5343	0.2700	0.3593	0.2114	0.1900
				3	900	51.40%	0.5991	0.5425	0.2712	0.3634	0.2113	0.1902
				2	175	51.57%	0.6019	0.5368	0.2726	0.3493	0.2102	0.1905
				1	175	51.47%	0.6018	0.5356	0.2691	0.3446	0.2131	0.1931

Table 10. More results with different r, n and k for StreamPETR and OPEN.

set a 1% mAP drop as the threshold, the maximum number of keys that can be pruned varies depending on the model, and the same applies to n. When r is too large or $n = 1$, some models may experience a performance drop exceeding 1%. However, the model performance does not completely degrade into an unusable state but remains at a functional level.

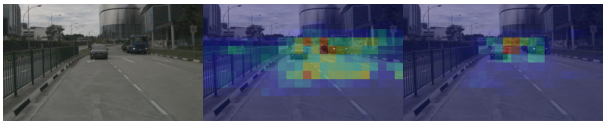
We have conducted extensive experiments for each model, but due to space limitations, only a portion of the results can be presented here. Please refer to our GitHub repository for additional experimental results.

Model	Pruning Method	r	n	mAP↑	NDS↑
StreamPETR	-	2000	2	38.01%	0.4822
	tgGBC			37.93%	0.4817
	DART			33.88%	0.4513
	RANDOM			26.87%	0.3982

Table 11. Comparison to More baselines.

C.5. Visualization of Attention Focus

The comparison of attention maps before and after pruning is presented in Fig. 11. It can be observed that tgGBC removes irrelevant keys, thereby making the attention more concentrated on the objects of interest.



(a) Camera image (b) Attention focus before pruning using tgGBC (c) Attention focus after pruning using tgGBC

Figure 11. Comparison of attention scores between before and after pruning.

D. Further Exploration

D.1. Why Some Models Improve in Performance

We can observe that for some models, the mAP and NDS increase rather than decrease after pruning, such as FocalPETR-vov-800x320 in Tab. 1 and StreamPETR-r50-704x256 with n=4 in Tab. 10. We believe this is related to the redundant information in the image features. As shown in Fig. 8, the key is the image feature extracted by the backbone, which inevitably contains background information (such as sky, buildings, etc.) that is ineffective for object detection. These keys interact with the query and affect the detection performance.

In the original 3D detectors, for each transformer decoder layer, the query is continuously updated, while the key and value remain unchanged. Therefore, the background key repeatedly influences the query. In fact, it can be argued that the "unimportant keys" pruned by our method are essentially background tokens. It is precisely because these keys, which interfere with detection, are pruned that the phenomenon of increased model performance occurs.

D.2. Pruning Queries

While it is possible to prune both keys and queries at runtime, the latter's involvement in self-attention operations limits the extent of pruning. To ensure that the mAP degradation does not exceed 1%, we cannot prune 300 queries, offering only marginal speed improvements. Conversely, to achieve significant acceleration in model speed, pruning 600 queries would result in a sharp decline in mAP. The results are shown in Tab. 12.

We believe that pruning the key is more effective than pruning the query for the following reasons: The key does not have explicit self-attention. In contrast, after interacting with the key, the query is fed into the self-attention mechanism of the next layer. This introduces internal dependencies, meaning that even if a query generates a low classification score, its value may influence queries with high classification scores through self-attention. Therefore, pruning the query can have a significant impact on the remaining queries, thereby degrading model performance. In contrast, the dependencies of the key are indirect, so pruning the key has a lower impact on model performance. Moreover, keys contain redundant information more than queries. Please see the analysis in Appendix D.1.

D.3. Pruning Fully Converged Models

To ensure a fair comparison with prior work while considering training efficiency, many previous experiments use a 24-epoch training schedule, which often does not achieve full convergence. To assess whether tgGBC remains effective after full convergence, we trained a StreamPETR-ResNet50-704x256 model for 120 epochs. As shown in Tab. 14, tgGBC preserves the model's performance even after full convergence, with only a 0.08% decrease in mAP and a 0.0007 reduction in NDS.

D.4. Training with tgGBC

If there is a new model, one can also train it with tgGBC from the beginning, as shown in Tab. 15. Our method is capable of reducing training time. For example, training StreamPETR-vov-1600x640 with $r = 21000$ and $n = 1$ for 30 epochs takes less time than training without tgGBC for 24 epochs while achieving a better mAP.

D.5. 2D Object Detection Models with tgGBC

As described in Sec. 2.2, the number of keys in ViT-based methods is significantly smaller than that in 3D object detection methods. Similarly, the number of keys in 2D object detection is around 1,000 (e.g., in ConditionalDETR). This is also why we focus on 3D object detection rather than extensively studying 2D object detection methods. Moreover, current 2D object detection methods are rapidly evolving and highly mature. Methods based on DETR are not the absolute mainstream, as other approaches, such as the YOLO series, are still widely used in 2D object detection tasks.

Additionally, 3D object detection is a highly practical and valuable task. Therefore, from the very beginning, we focused on pruning 3D object detection models. However, for some DETR-based 2D detection methods, tgGBC can still be applied. Here, we take ConditionalDETR as an example to verify the effectiveness of tgGBC on 2D object

Model	r	q	mAP ↑	NDS ↑	mATE ↓	mASE ↓	mAOE ↓	mAVE ↓	mAAE ↓	Dec. Time (ms)
StreamPETR	-	-	48.89%	0.5732	0.6096	0.2601	0.3882	0.2603	0.1944	64.93
	21000	-	48.55%	0.5730	0.6033	0.2626	0.3771	0.2611	0.1941	34.98
		300	48.42%	0.5703	0.6055	0.2633	0.3912	0.2620	0.1958	34.09
		600	47.46%	0.5606	0.6239	0.2661	0.3952	0.2826	0.1992	29.42
OPEN	-	-	51.80%	0.6043	0.5314	0.2679	0.3457	0.2095	0.1922	46.39
	12000	-	51.57%	0.6019	0.5368	0.2726	0.3493	0.2102	0.1905	28.99
		300	51.48%	0.6014	0.5389	0.2727	0.3460	0.2088	0.1939	26.31
		600	50.24%	0.5922	0.5542	0.2732	0.3524	0.2144	0.1959	25.97

Table 12. Results of pruning queries. We use StreamPETR-vov-1600x640 and OPEN-r101-1408x512.

Model	Backbone	TgGBC	mAP ↑	AP_{50} ↑	AP_{75} ↑	AP_s ↑	AP_m ↑	AP_l ↑	Inf. Time (ms) ↓
DETR	ResNet50	-	0.421	0.623	0.442	0.214	0.460	0.610	42.06
		✓	0.414	0.620	0.436	0.205	0.455	0.603	35.24(-16.21%, 1.19x)
	ResNet101	-	0.435	0.638	0.463	0.218	0.480	0.480	54.61
		✓	0.426	0.635	0.453	0.211	0.470	0.608	47.01(-13.92%, 1.16x)
ConditionalDETR	ResNet50	-	0.409	0.619	0.434	0.207	0.442	0.595	43.88
		✓	0.400	0.615	0.427	0.196	0.436	0.587	40.13 (-8.55%, 1.09x)
	ResNet101	-	0.428	0.636	0.459	0.218	0.467	0.610	69.97
		✓	0.424	0.635	0.454	0.215	0.463	0.605	55.66 (-20.45%, 1.26x)

Table 13. Results of DETR and ConditionalDETR with tgGBC. Lines with "tgGBC" remaining blank are the original results without pruning.

r	mAP ↑	NDS ↑	mATE ↓	mASE ↓	mAOE ↓	mAVE ↓	mAAE ↓
-	43.07%	0.5389	0.6023	0.2686	0.4238	0.2597	0.2105
2000	42.99%	0.5382	0.6035	0.2696	0.4230	0.2609	0.2099

Table 14. Pruning fully converged models. We train StreamPETR-r50-704x256 for 120 epochs to ensure its full convergence. The first line remaining r blank is the original model's results without pruning.

r	mAP ↑	NDS ↑	mATE ↓	mASE ↓	mAOE ↓	mAVE ↓	mAAE ↓	Training Time
-	48.89%	0.5732	0.6096	0.2601	0.3882	0.2603	0.1944	2d 14h
21000	49.42%	0.5787	0.5982	0.2579	0.3651	0.2698	0.1926	2d 13h

Table 15. Training StreamPETR-vov-1600x640 with tgGBC, while tgGBC is applied, n and k are set to 1 and 175, respectively.

detection methods, as shown in Tab. 13.

In ConditionalDETR, the number of keys is not always the same. Therefore, we adopt a configuration similar to MV2D, using r to represent the pruning ratio and set a threshold t. When the current number of keys exceeds t, pruning is performed. Through experiments, tgGBC can reduce the model's inference time by 19.17% ($1.24\times$) while keeping the mAP degradation below 1%.