

## Supplementary

In this supplementary, we provide more materials as follows,  
Sec. A details the implementation, training, and inference;  
Sec. B presents additional experiments and analyses with tight parameter budgets  
Sec. C offers more qualitative visualizations.

### A Details of Implementation, Training, and Inference

Our implementation is based on the open-source codebase Pointcept (here) and follows the official implementations for Sonata [76], PPT [79], as well as PTv3 [77].

Leveraging the parameter efficiency of our method, we train on a single 4090 GPU for much fewer epochs to obtain the reported performance. For example, we train on ScanNet for 100 epochs, in contrast to the 800 epochs in the released Sonata configuration. Our code is available at <https://github.com/LiyaoTang/GEM>.

Tab. 7 summarizes empirical latency and memory usage. We intentionally refrain from deployment-specific optimizations such as LoRA weight merging. Although the global attention and local convolution modules introduce extra cost, the runtime and memory footprint stay within the same order of magnitude as other PEFT baselines.

### B More Experiments and Analysis

In spite of the promising results shown in Sec. 3, we suggest that it can better demonstrate the full potential of PEFT methods under broader and more challenging settings.

**PEFT for shape analysis.** While GEM is motivated by PEFT for large-scale 3D scenes, most existing 3D PEFT methods target object-level understanding, as discussed in Sec. 2. To assess cross-domain generalization and compare against these existing 3D-specific PEFT methods, we further evaluate GEM on 3D shape datasets, ShapeNetPart [85]. As shown in Tab. 8, GEM remains competitive and achieves the best performance. We also observe that common backbones for shape analysis, such as ReCon [55], attach large MLP heads that account for approximately 20% of all parameters, rendering fine-tuning inefficient. To broaden the applicability of backbone models for 3D shapes, an interesting direction for future work is to develop architectures with more parameter-efficient heads that are better suited to fine-tuning.

**PEFT for 3D convolutional networks.** While our primary experiments apply GEM to state-of-the-art backbones, mainly transformer models, we also verify its generality on convolutional architectures. In particular, we integrate GEM into SparseUNet [10], a 3D convolutional model, following the MSC [80] protocol to pre-train on ScanNet [12] and fine-tune on ScanNet200 [61]. As reported in Tab. 9, GEM improves performance by +4.6 mIoU over linear probing and outperforms other PEFT baselines, demonstrating robustness beyond transformer models. We notice the gains are narrower than those on transformer backbones, largely due to the limited capacity of convolutions, as also indicated by a near-collapse linear probing performance in this setting.

**PEFT under tight budgets.** To stress parameter efficiency, we constrain all methods to the same limited learnable-parameter budgets, including enforcing rank  $r = 1$ , allowing 0.1% parameters, and 1% parameters. As reported in Tab. 10, GEM delivers consistently higher accuracy, whereas several baselines fail when the budget becomes extremely stringent.

### C More Visualizations

We provide more qualitative results regarding the attention map generated by our latent tokens in Fig. 4. Despite shared across the stages, the latent tokens appear to focus on different parts of the scenes, providing different scene context to the backbone models at different stages.

Methods	Params.	Memory	Latency
<i>Base model</i>			
Sonata (full.) [76]	124.8M	8.2G	61.8ms
Sonata (ft.)	108.5M	6.4G	50.2ms
<i>PEFT methods</i>			
Sonata (lin.)	0.02M	<b>6.4G</b>	<b>50.2ms</b>
+Adapter [25]	2.8M	6.7G	50.6ms
+LoRA [26]	1.9M	7.9G	52.8ms
+Prompt Tunning [37]	5.5M	7.3G	51.5ms
<b>+GEM (ours)</b>	1.8M	7.1G	57.8ms

Table 7. **Inference efficiency of PEFT methods.** We benchmark with the batch size fixed to 1.

Tight Budgets	Params	ScanNet200 Val [61]			
Methods	Learn.	Pct.	mIoU	mAcc	allAcc
<i>Training from scratch</i>					
SparseUnet [10]	39.2M	100%	25.0	32.9	80.4
<i>Full fine-tuning</i>					
SparseUnet (ft.) [80]	0.02M	0.05%	32.0	41.6	82.3
<i>PEFT methods with rank=1</i>					
SparseUnet (lin.)	0.02M	0.05%	1.5	2.5	53.6
+BitFit [6]	0.03M	0.07%	4.8	6.9	62.7
+Adapter [25]	0.6M	1.5%	9.5	13.4	69.5
+LoRA [26]	0.8M	2.0%	13.2	17.6	74.7
<b>+GEM (ours)</b>	0.6M	1.5%	<b>15.2</b>	<b>22.9</b>	<b>75.6</b>

Table 9. **Generalizing to convolutional networks.**

Shape-Part Seg.	Params	ShapeNetPart [85]	
Methods	Learn.	Cls. mIoU	Inst. mIoU
<i>Full fine-tuning</i>			
ReCon (ft.) [55]	27.06	84.52	86.1
<i>PEFT methods</i>			
ReCon (lin.) [55]	5.23M	83.06	85.2
+PointLoRA [71]	5.63M	83.98	85.4
+PointGST [39]	5.59M	83.98	85.8
<b>+GEM (ours)</b>	5.58M	<b>84.02</b>	<b>85.8</b>

Table 8. **Generalizing to 3D shapes.**

Tight Budgets	Params	ScanNet Val [12]			
Methods	Learn.	Pct.	mIoU	mAcc	allAcc
<i>Training from scratch</i>					
PTv3 [77]	124.8M	100%	77.6	85.0	92.0
<i>Full fine-tuning</i>					
Sonata (full.) [76]	124.8M	100%	79.4	86.1	92.5
Sonata (ft.)	108.5M	100%	78.3	85.9	92.3
<i>PEFT methods with rank=1</i>					
Sonata (lin.)	0.02M	0.02%	72.5	83.1	89.7
+Adapter [25]	0.05M	0.04%	74.0	84.1	90.5
+LoRA [26]	0.05M	0.05%	42.5	55.4	75.1
+Prompt Tunning [37]	0.05M	0.05%	72.9	83.5	90.0
<b>+GEM (ours)</b>	0.07M	0.06%	<b>75.2</b>	<b>84.8</b>	<b>91.1</b>
<i>PEFT methods with 0.1% params.</i>					
Sonata (lin.)					
+Adapter [25]	0.1M	0.1%	75.1	84.7	91.1
+LoRA [26]	0.1M	0.1%	75.0	84.4	91.1
+Prompt Tunning [37]	0.1M	0.1%	73.5	83.9	90.3
<b>+GEM (ours)</b>	0.1M	0.1%	<b>76.5</b>	<b>85.5</b>	<b>91.6</b>
<i>PEFT methods with 1% params.</i>					
Sonata (lin.)					
+Adapter [25]	1.1M	1.0%	76.6	85.3	91.7
+LoRA [26]	1.1M	1.0%	76.7	85.6	91.7
+Prompt Tunning [37]	1.1M	1.0%	73.8	84.2	90.4
<b>+GEM (ours)</b>	1.1M	1.0%	<b>78.2</b>	<b>86.3</b>	<b>92.2</b>

Table 10. **Performance with tight budgets.**

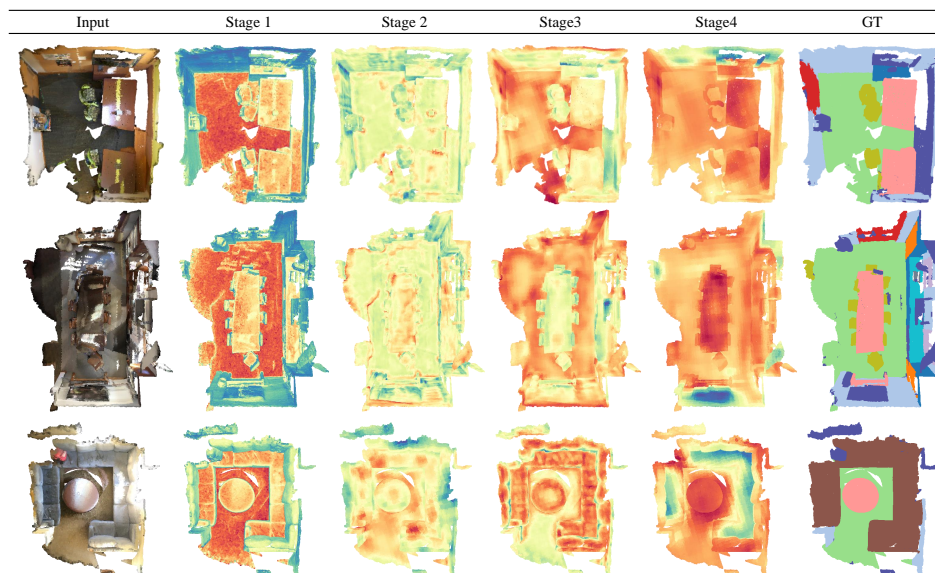


Figure 4. Attention maps of our latent tokens in context adapter.