Sparse Refinement for Efficient High-Resolution Semantic Segmentation *Supplementary Material*

Zhijian Liu[1,2,*] Zhuoyang Zhang[3,*] Samir Khaki[4] Shang Yang[1]
Haotian Tang[1] Chenfeng Xu[5] Kurt Keutzer[5] Song Han[1,2]

[1]MIT [2]NVIDIA [3]Tsinghua University [4]University of Toronto [5]UC Berkeley
https://sparserefine.mit.edu

0 Table of Contents

This supplementary material includes:

- Sec. 1: More model details of the entropy selector and the sparse feature extractor.
- Sec. 2: More ablation details of the learnable selector, entropy ensembler, and mIoU improvement for different object sizes.
- Sec. 3: More latency results on NVIDIA RTX 3090 and Jestion AGX Orin.
- Sec. 4: More visualizations.

1 More Model Details

1.1 Entropy Selector

In Figure 2(b) of the main paper, we present the recall-density curve of HRNet-W48 on Cityscapes to demonstrate the effectiveness of the entropy selector. Additionally, in Figure 1, we provide the recall-density curves of HRNet-W48 on the remaining four datasets we reported in the main paper. Our entropy selector consistently achieves high recall rates at low density, showcasing its overall effectiveness.

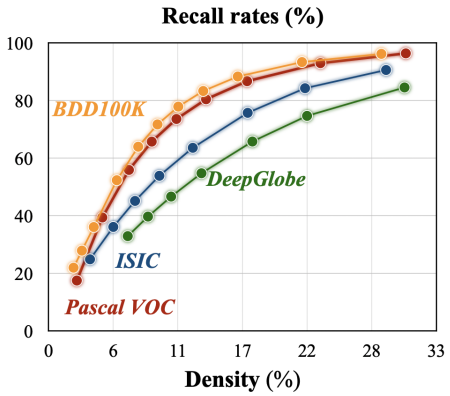

Fig. 1: The entropy selector consistently achieves high recall rates at low density on different datasets.

In Table 1, we present the entropy thresholds ( ) chosen for different baseline models on Cityscapes. Our guiding principle for selection is based on the desired recall ratio. In this work, we determine a threshold for each model to target a recall rate of at least 80%.

Table 1: Entropy threshold, recall rate, precision, and density for each baseline model.

	Entropy	Recall	Precision	Density
HRNet-W48	0.3	84.9%	22.8%	11.8%
SegFormer-B5	0.1	84.7%	20.1%	13.9%
Mask2Former-T	0.05	93.4%	13.1%	24.1%
Mask2Former-L	0.005	97.3%	9.1%	35.0%
SegNeXt-L	0.1	80.8%	24.2%	10.0%

1.2 Sparse Feature Extractor

By default, we use a 6-stage sparse feature extractor where channel dimensions for each stage are defined as 32, 64, 128, 256, 512 and 1024, respectively. For Mask2former-T, we use a 5-stage sparse feature extractor where channel dimensions for each stage are defined as 32, 64, 128, 256 and 512, respectively. Due to space limit, we illustrate the detailed 5-stage model architecture in Figure 2. At each stage, we employ two ResNet basic blocks before downsampling and an additional two after upsampling. These ResNet basic blocks incorporate sparse convolution, with a kernel size of 3 used for each sparse convolution operation. This choice enables us to capture ample spatial information while maintaining a manageable model size.

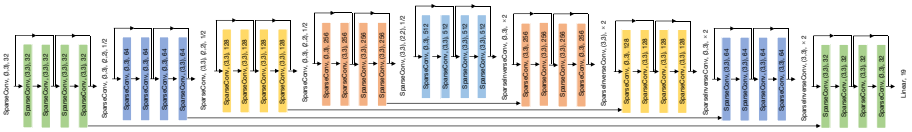

Fig. 2: Detailed model architecture of the sparse feature extractor.

2 More Ablation Details

2.1 Learnable Selector

In Section 3.2 of the main paper, we introduced the learnable selector, as an alternative to the entropy selector for identifying *less confident predictions*. The learnable selector was designed to use the features available at the selection point of the SparseRefine pipeline, this includes the original RGB image and the low-resolution baseline logits. We feed the RGB image through 2 dense convolutions to gain a feature representation. We then concatenated these features with the logits before feeding it into an MLP. We include the corresponding PyTorch descriptor code below:

```
class LearnableSelector(nn.Module):
    def __init__(self, threshold: float):
        self.conv = nn.Sequential(
            nn.Conv2d(3, hidden_channels, 3, 1, 1),
            nn.ReLU(),
            nn.Conv2d(hidden_channels, output_channels, 3, 1, 1),
            nn.ReLU()
        )
        self.mlp = nn.Sequential(
            nn.Linear(num_classes+out_channels, mlp_hidden_size),
            nn.ReLU(),
            nn.Linear(mlp_hidden_size, mlp_hidden_size),
            nn.ReLU(),
            nn.Linear(mlp_hidden_size, 1)
        )
        self.sig = nn.Sigmoid()
        self.threshold = threshold

    def forward(self, image, logits):
        image_feats = self.conv(image)
        concat_feats = torch.cat([image_feats, logits], dim=-1)
        out_feats = self.mlp(concat_features)
        return self.sig(out_feats) > self.threshold
```

Our lightweight learnable selector incurs a relatively similar latency compared to the entropy selector. Using the above PyTorch code, we set `hidden-channels=16`, `output-channels=16`, `mlp-hidden-size=32` achieving a selector latency of 4.0 ms.

In Section 5a of the main paper, we empirically show that the lightweight learnable selector offers a marginal performance improvement (1.0% in recall) over the entropy selector, however at roughly 2× the latency and no miou improvement. Given that our SparseRefine pipeline is built to achieve a competitive latency-to-accuracy trade off, we can conclude that the entropy selector works sufficiently well for the identification of *uncertain predictions*, particularly given that the downstream mIoUs are similar.

2.2 Entropy Ensembler

We provide further details about the entropy-based ensembler mentioned in Section 4.3 of the main paper. We show the PyTorch implementation below:

```
class EntropyEnsembler(nn.Module):
    def forward(self, l1: torch.Tensor, l2: torch.Tensor) -> torch.Tensor:
        p1 = torch.softmax(l1, dim=-1)
        e1 = -torch.sum(p1 * torch.log(p1.clamp(min=1e-5)), dim=-1, keepdim=True)
        p2 = torch.softmax(l2, dim=-1)
        e2 = -torch.sum(p2 * torch.log(p2.clamp(min=1e-5)), dim=-1, keepdim=True)
        return torch.where(e1 < e2, l1, l2)
```

2.3 mIoU for Different Object Scale.

Figure 3 shows mIoU improvements from the low-resolution baseline, for various object sizes in the Cityscapes dataset. We bin the objects into small, medium and large based on the percentage of the frame they occupy. Cumulatively we show that our method disproportionately improves the segmentation of smaller objects. Table 2 includes a subset of our per-class mIoUs from Table 8 (main), with the relative class size per frame. Notably, we exhibit larger mIoU improvements for smaller objects such as traffic lights, and riders.

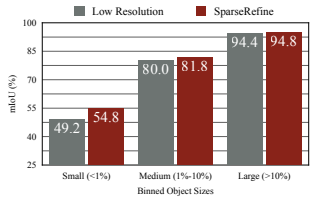

Fig. 3: Comparing the mIoU for low resolution and SparseRefine on different sized objects in the Cityscapes Dataset.

3 More Latency Results

In the main paper, we reported the latency of SparseRefine using FP16 precision and a batch size of 4 on an NVIDIA RTX 3090. To provide a more

Table 2: Average class size and mIoU per frame on Cityscapes

	Traffic Light	Rider	Road	Building
Low Resolution (mIoU)	70.0	62.3	98.3	92.8
+ SparseRefine (mIoU)	75.0	67.4	98.4	93.4
Relative Size (%)	0.34	0.42	38.44	22.83

comprehensive understanding of latency, Table 3 offers a detailed measurement of latency across various precision modes and batch sizes. It's noteworthy that SparseRefine consistently reduces latency across different precision modes and batch sizes. As the batch size increases, we generally observe a decrease in the inference latency of SparseRefine. This phenomenon can be attributed to larger batch sizes leading to higher GPU utilization during sparse convolution operations. Furthermore, it's worth mentioning that some larger models encounter out-of-memory errors when performing inference on high-resolution inputs with a large batch size. In contrast, SparseRefine does not face this issue, underscoring its efficiency and robustness in handling resource constraints.

We also conducted latency evaluations of SparseRefine on an NVIDIA Jetson AGX Orin, a widely adopted platform for autonomous driving, and the outcomes are summarized in Table 4. These results offer crucial insights into the practical applicability of our approach. In general, it is observed that the inference latency on the NVIDIA Jetson AGX Orin platform is roughly four times slower compared to the NVIDIA RTX 3090. Despite this discrepancy, SparseRefine manages to achieve a noteworthy speedup on the NVIDIA Jetson AGX Orin, affirming its promise and suitability for real-world applications in the context of autonomous driving.

4 More Visualizations

We present visualizations of the prediction differences between the high-resolution and low-resolution baseline models on Cityscapes in Figure 4. These visualizations clearly illustrate that the differences in their predictions primarily emerge in a sparse set of pixels. This further strengthens the claim we made at the beginning of Section 3.2 in the main paper.

Table 3: Latency under different precision and batch size on NVIDIA RTX 3090. SparseRefine benefits from high utilization of sparse convolution, particularly as the batch size increases.

	Input Resolution		FP16				FP32			
			B1	B2	B4	B8	B1	B2	B4	B8
HRNet-W48	1024	2048 (D)	46.3	54.1	53.4	51.5	95.2	96.5	90.9	89.7
HRNet-W48	512	1024 (D)	20.2	15.1	14.5	14.1	26.8	25.8	24.4	23.8
+ SparseRefine	1024	2048 (S)	43.1	34.0	32.4	30.6	69.3	64.2	60.8	58.7
SegFormer-B5	1024	2048 (D)	149.5	141.1	140.6	141.3	311.5	302.1	299.2	OOM
SegFormer-B5	512	1024 (D)	39.0	20.6	18.5	17.8	43.7	39.9	38.3	37.2
+ SparseRefine	1024	2048 (S)	65.4	41.8	38.5	36.0	91.5	83.0	79.2	76.7
Mask2Former-T	1024	2048 (D)	73.9	72.7	66.8	65.4	156.6	145.9	138.8	OOM
Mask2Former-T	512	1024 (D)	37.5	24.1	19.1	17.1	49.7	42.1	37.6	34.6
+ SparseRefine	1024	2048 (S)	67.9	51.7	44.8	44.4	111.9	99.2	93.7	89.3
Mask2Former-L	1024	2048 (D)	158.4	150.8	146.3	144.6	382.2	371.1	367.1	OOM
Mask2Former-L	512	1024 (D)	56.3	45.4	40.3	37.5	113.0	100.8	94.6	92.1
+ SparseRefine	1024	2048 (S)	106.4	90.6	80.4	76.4	213.9	195.3	186.4	182.2
SegNeXt-L	1024	2048 (D)	85.3	87.3	86.3	84.0	132.7	155.2	141.5	134.2
SegNeXt-L	640	1280 (D)	34.7	35.4	33.6	32.8	53.5	62.7	56.7	52.7
+ SparseRefine	1024	2048 (S)	56.9	53.7	50.9	48.7	91.3	97.5	89.0	83.7

Fig. 4: Visualization of the prediction difference between the high resolution and low resolution baseline models.

We have included additional visualization results of refinement in Figure 5. These supplementary results exhibit enhanced recognition of small, distant objects and finer details around object boundaries. These observations serve as further evidence of the efficacy of our SparseRefine.

Table 4: Latency under different precision and batch size on NVIDIA Jetson AGX Orin.

	Input Resolution		FP16			FP32		
			B1	B2	B4	B1	B2	B4
HRNet-W48	1024	2048 (D)	233.0	217.1	211.1	360.6	365.8	349.4
HRNet-W48	512	1024 (D)	78.6	63.9	60.5	98.9	93.6	90.5
+ SparseRefine	1024	2048 (S)	157.0	138.9	134.1	277.4	269.9	265.3
SegFormer-B5	1024	2048 (D)	751.5	721.0	719.5	1220.5	1172.5	1172.8
SegFormer-B5	512	1024 (D)	110.8	102.4	98.8	179.9	161.3	163.1
+ SparseRefine	1024	2048 (S)	197.5	186.7	183.1	386.1	363.7	362.3
Mask2Former-T	1024	2048 (D)	449.1	433.7	422.8	583.2	591.7	585.6
Mask2Former-T	512	1024 (D)	133.0	126.2	115.1	174.8	171.4	149.8
+ SparseRefine	1024	2048 (S)	259.0	250.5	241.0	486.1	479.1	453.7
Mask2Former-L	1024	2048 (D)	882.1	903.2	888.1	1217.7	1198.5	1201.8
Mask2Former-L	512	1024 (D)	262.2	243.7	226.6	356.7	334.0	317.6
+ SparseRefine	1024	2048 (S)	432.6	412.0	396.9	784.4	757.5	736.9
SegNeXt-L	1024	2048 (D)	525.4	529.3	539.1	712.0	743.7	731.8
SegNeXt-L	640	1280 (D)	209.9	214.6	202.7	280.3	296.9	288.5
+ SparseRefine	1024	2048 (S)	278.4	279.5	266.2	447.5	450.6	441.5

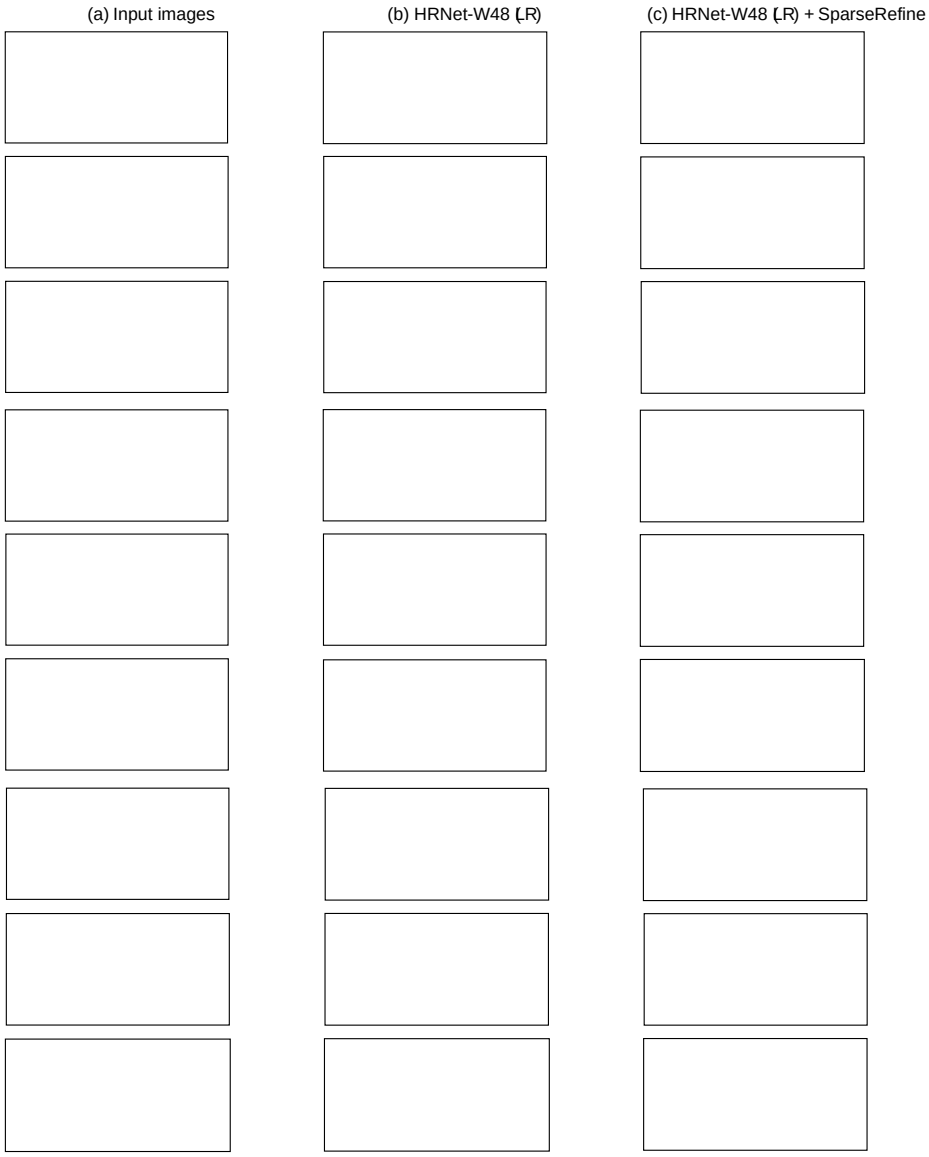

Fig. 5: SparseRefine improves the low-resolution (*LR*) baseline with substantially better recognition of small, distant objects and finer detail around object boundaries.