MPM: Mutual Pair Merging for Efficient Vision Transformers

Supplementary Material

In the supplementary material, we provide additional implementation details (Sec. 1), more results and analysis (Sec. 2), and further ablations (Sec. 3). We also briefly discuss approaches that we explored but that were unsuccessful (Sec. 4), and provide details on the configurations of other methods used in our comparisons (Sec. 5). Finally, we provide qualitative results of MPM on various datasets in Sec. 6.

1. Implementation Details

1.1. Training

For the main results in Table 1 of the main paper and for other methods using the Segmenter [10] model, we used the official checkpoint when possible. When no checkpoint was provided (for example, for ViT-S/16 on Cityscapes), we trained each model using the official Segmenter training recipe from the authors' GitHub repository. We did not change any hyperparameter that could affect final performance. However, to speed up training and improve efficiency, we used mixed precision and FlashAttention-2 [3]. We observed no noticeable difference in final performance between mixed precision and full precision.

We did not use FlashAttention-3 [9] for the main results, and instead relied on FlashAttention-2 because, at the time of writing, FlashAttention-3 is still in beta according to the official GitHub repository. We did benchmark FlashAttention-3 and observed no significant latency difference compared to FlashAttention-2 on Hopper. We believe this is due to the relatively short sequence lengths of vision models compared to large language models.

1.2. EVA

For the EVA-01 [5] backbone with a ViT Adapter [1] and a Mask2Former [2] head, we insert MPM before the 3rd, 13th, 22nd, and 31st transformer blocks of the EVA backbone so that tokens are merged once for each interaction block of the adapter [1]. The images are resized to 896x896 before being passed to the model, which results in a token sequence of length 3136 (896/16 $\times$ 896/16). The model is evaluated on the ADE20K validation set, and latency is measured at batch size 4 on a single A100 GPU in half precision. We use the pretrained weights from [5]. The base token sequence is reconstructed at the end of each interaction block, before being extracted by the adapter module. This means that merges do not compound over the entire backbone, but only within each segment of the backbone separated by adapter interaction blocks.

2. Additional Results

To analyze the real overhead of MPM, we profile its runtime under different model settings. In Tab. 3, we report the runtime of a single image through the model together with the MPM overhead. MPM takes about 2 ms per pass (including reconstruction), while the speedup is about 1.6 ms per block after the first MPM module and an additional 1.1 ms per block after the second MPM module. In total, the average speedup for a full forward pass is just over 20 ms, which is consistent with Tab. 4, where we profile a standalone ViT-S/16 forward pass at 512x512.

Next, we investigate whether MPM remains useful when combined with an optimized kernel such as FlashAttention-2 [3]. We use the same profiling setup on ViT-S/16, but increase the image resolution to 1024x1024 to lengthen the token sequence and provide some scaling insight for higher resolutions. The result in Tab. 5 shows that the speedup is much more modest but still present. The first MPM module yields about 6 ms of speedup per block, while the second module adds only about 0.8 ms. The total speedup is about 1.3 ms per pass when compared with Tab. 6.

To better understand the latency of different operations involved in MPM and attention mechanisms, we provide a detailed benchmark on an NVIDIA H100 GPU in Tab. 2. This table includes timings for operations such as Argmax, Argsort, Batch Matrix Multiplication (BMM), standard Attention, and FlashAttention for various input shapes. The results highlight the efficiency of FlashAttention compared to standard attention mechanisms, especially as the token count increases. Additionally, the overhead of MPM operations is relatively low compared to the overall attention computation time, making it a viable option for token reduction in Vision Transformers.

We also provide the full latency results (in FPS) for all models and backbones on ADE20K with FlashAttention-2 in Tab. 7. We report the mean and standard error over batches. For Tiny and Small backbones, MPM is on par with ToMe, while for Base and Large backbones MPM outperforms ToMe.

For a more direct fairness check, we also compare MPM and ToMe under a matched token budget on ADE20K in Fig. 1. ToMe's keep rate is set to match the average post-merge token count of MPM. Under this constraint, MPM achieves higher FPS with comparable mIoU for the settings we tested.

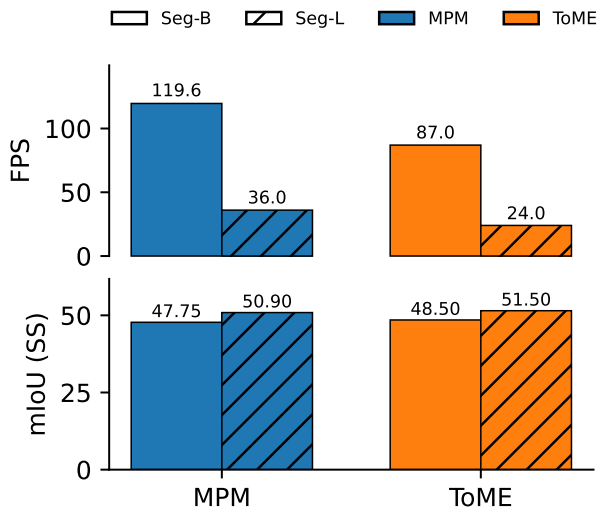

Figure 1. MPM vs. ToMe under a matched token budget on ADE20K (568 tokens for Seg-B and 880 tokens for Seg-L). ToMe's keep rate is set to match the adaptive token count produced by MPM after merging.

Dataset	Mean	Median	P75	P90	P95	Max
ADE20K	1.66	1.0	2.0	3.0	4.0	48

Table 1. Spatial distance of merged token pairs. Manhattan distance on the patch grid for MPM (layers 2 and 5) on ADE20K. Distances are reported in patch units.

To better understand the effect of global matching, we measure the Manhattan distance between merged token pairs on the patch grid in Tab. 1. Even without an explicit locality constraint, the merges are predominantly local: the median distance is 1 patch, the mean is 1.66, and 90% of merges occur within 3 patches.

To visualize the effect of MPM on segmentation quality, we provide qualitative results on ADE20K, Cityscapes, and Pascal Context in Sec. 6. The results show that MPM preserves most segmentation details.

Operation	**Input Shape**	**Time (ms)**
Argmax	N = 1024	0.014
Argsort	N = 512	0.039
BMM	32 x (512x768 @ 768x512)	0.040
BMM	32 x (1024x768 @ 768x1024)	0.096
Attention	(B=32, N=1024, D=768), 12 heads	0.724
Attention	(B=32, N=512, D=768), 12 heads	0.369
FlashAttention	(B=32, N=1024, D=768), 12 heads	0.381
FlashAttention	(B=32, N=512, D=768), 12 heads	0.133

Table 2. GPU benchmark results on NVIDIA H100 80GB HBM3 (50 iterations, 10 warm-up steps).

Component	**Mean Time (ms)**	**Tokens**
MPM merge (total per pass)	1.988	
MPM (mean for one pass)	0.994	
reconstruct gather (mean)	0.064	
Transformer Blocks		
Block 0	5.816	1025
Block 1	5.705	1025
Block 2 (MPM)	4.171	789
Block 3	4.147	789
Block 4	4.146	789
Block 5 (MPM)	3.004	620
Block 6	2.975	620
Block 7	2.973	620
Block 8	2.973	620
Block 9	2.973	620
Block 10	2.973	620
Block 11	2.974	620
Overall batch mean	**75.134**	

Table 3. Computation profile for **MPM(2,5) ViT-S/16** on **ADE20K**. MPM timing is not included in the block timings. Measured on NVIDIA H100 80GB HBM3, batch size 32, full precision after 10 warm-up steps.

Component	**Mean Time (ms)**	**Tokens**
Transformer Blocks		
Block 0	5.821	1025
Block 1	5.733	1025
Block 2	5.731	1025
Block 3	5.730	1025
Block 4	5.730	1025
Block 5	5.731	1025
Block 6	5.730	1025
Block 7	5.732	1025
Block 8	5.734	1025
Block 9	5.732	1025
Block 10	5.730	1025
Block 11	5.729	1025
Overall batch mean	**96.828**	

Table 4. Computation profile for **ViT-S/16** baseline on **ADE20K**. Same settings as Tab. 3.

3. Additional Ablations

We provide the full results of applying MPM to DeiT backbones on ADE20K in Tab. 8. We also provide results of applying MPM to plain ViT backbones with a linear segmentation head [10] in Tab. 9.

MPM layer ablations on ViT-Small and ViT-Tiny backbones are shown in Fig. 2 and Fig. 3, respectively. The results are consistent with those in Figure 3 of the main paper for the ViT-Base model. We also observe that merging before any transformer block (layer = 0) does not provide a

Component	Mean Time (ms)	Tokens
MPM merge (total per pass)	8.997	
MPM (mean for one pass)	4.498	
reconstruct gather (mean)	0.183	
Transformer Blocks		
Block 0	5.349	4097
Block 1	5.239	4097
Block 2 (MPM)	4.631	3508
Block 3	4.497	3508
Block 4	4.492	3508
Block 5 (MPM)	4.061	3020
Block 6	3.885	3020
Block 7	3.882	3020
Block 8	3.880	3020
Block 9	3.876	3020
Block 10	3.863	3020
Block 11	3.877	3020
Overall batch mean	**97.721**	

Table 5. Computation profile for **MPM(2,5) ViT-S/16** on ADE20K with images resized to 1024x1024, FlashAttention-2, batch size 32, and BFloat16 precision.

Component	Mean Time (ms)	Tokens
Transformer Blocks		
Block 0	5.365	4097
Block 1	5.240	4097
Block 2	5.239	4097
Block 3	5.236	4097
Block 4	5.236	4097
Block 5	5.232	4097
Block 6	5.230	4097
Block 7	5.228	4097
Block 8	5.231	4097
Block 9	5.221	4097
Block 10	5.199	4097
Block 11	5.219	4097
Overall batch mean	**99.022**	

Table 6. Computation profile for ViT-S/16 with FlashAttention-2 at 1024x1024 resolution (batch size 32, AMP).

Model	FPS (batch size 32)
ViT-Tiny Segmenter	1170 ± 40
CTS	730 ± 80
ALGM	1030 ± 50
ToMe(r=30)	$\mathbf{1260 \pm 130}$
MPM (2,5)	1190 ± 90
ViT-Small Segmenter	1010 ± 40
CTS	620 ± 60
ALGM	890 ± 40
ToMe(r=30)	$\mathbf{1100 \pm 100}$
MPM (2,5)	1030 ± 70
ViT-Base Segmenter	735 ± 6
CTS	480 ± 40
ALGM	640 ± 70
ToMe(r=30)	750 ± 50
MPM (2,5)	$\mathbf{781 \pm 24}$
ViT-Large Segmenter	375 ± 8
CTS	314 ± 19
ALGM	352 ± 7
ToMe(r=15)	351 ± 11
MPM (2,5)	$\mathbf{457 \pm 9}$

Table 7. Full latency (FPS) results for batch size 32 on ADE20K (512x512 resolution, BFloat16, FlashAttention-2). Values are mean $\pm$ standard error, rounded following significant-digit conventions.

Model	mIoU	GFLOPs	FPS BS=32
DeiT-T	37.2	25	664 ± 3
MPM (2,5)	36.8	$\sim$17.8	$\mathbf{817 \pm 10}$
DeiT-S	43.2	77	336 ± 5
MPM (2,5)	42.7	$\sim$54	$\mathbf{431 \pm 4}$
DeiT-B	46.2	259	130 ± 5
MPM (2,5)	45.6	$\sim$187	$\mathbf{175 \pm 3}$

Table 8. Results obtained by applying MPM to DeiT-Tiny through DeiT-Base backbones on ADE20K at 512x512 resolution in FP32.

consistent latency boost and significantly hurts performance. This is likely because merging too early removes too much information before the transformer blocks have processed it. At that stage, the features are still too raw, and the tokens have not yet interacted with one another.

4. Negative Results

4.1. Training with MPM

We tried training models from scratch with MPM modules inserted before the early transformer blocks. However, training was unstable and the final performance was significantly worse than training without MPM and inserting it only at inference time. We hypothesize that this is because MPM is a hard merging method, and during training the model needs time to adapt to merged tokens. A softer or more gradual merging schedule, for example starting with no merging and progressively increasing the amount of merging, could help stabilize training. We did not explore this direction further since our main goal was to develop a training-free method.

Model	mIoU	GFLOPs	FPS BS=32
ViT-T/16	39.1	21	765 ± 5
MPM (2,5)	38.7	∼13	$\mathbf{1003 \pm 25}$
ViT-S/16	45.0	64	396.55 ± 1.64
MPM (2,5)	44.6	∼41	$\mathbf{543 \pm 10}$
ViT-B/16	47.8	214	160 ± 4
MPM (2,5)	46.7	∼141	$\mathbf{228 \pm 2}$
ViT-L/16	51.4	724	53 ± 4
MPM (2,5)	50.0	∼414	87 ± 4

Table 9. Results applying MPM to plain ViT backbones with a linear segmentation head [10].

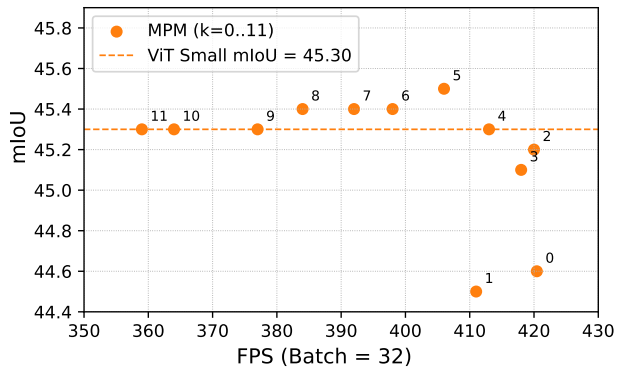

Figure 2. Accuracy vs. FPS for MPM using different insertion layers in ViT-Small [4] on ADE20K [11].

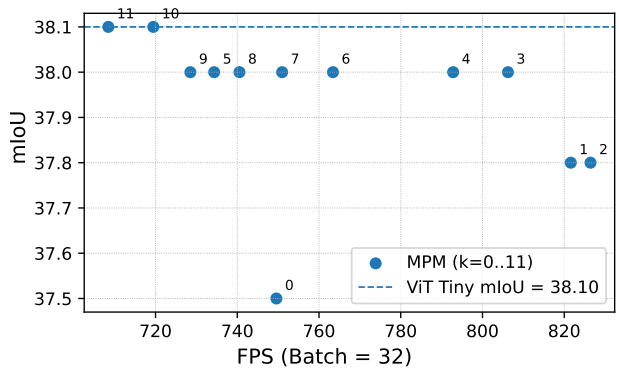

Figure 3. Accuracy vs. FPS for MPM using different insertion layers in ViT-Tiny [4] on ADE20K [11].

4.2. Region-based MPM

Because MPM merges tokens based on similarity, we also explored merging tokens only within a local region (4x4 window), similar to methods such as ALGM [8] or CTS [7]. The goal was to preserve more spatial information by preventing distant tokens from being merged. However, we found that this approach did not improve performance compared to global MPM merging. We hypothesize that this is because MPM merges are already spatially local. As shown in Fig. 2, similar patches in images tend to be spatially close, and in plain ViTs the positional encoding is absolute, which means that tokens closer in space are also closer in the positional embedding space.

4.3. Hierarchical backbones

We tried applying MPM to hierarchical backbones such as Swin [6] with a Mask2Former head [2]. However, we found that MPM provides no speedup on Swin and even worsens all metrics. This was expected because Swin Transformers scale linearly with the number of tokens due to their windowed attention mechanism, unlike plain ViTs, which scale quadratically. As a result, reducing the number of tokens has little effect on total computation time. We provide results for MPM applied to a Swin-T backbone with a Mask2Former head on ADE20K in Tab. 10.

For Swin, we applied MPM just before the attention path when there is no downscaling, and we reconstructed the full token sequence immediately afterward. This was necessary because windowed attention requires a fixed token grid. This also means that MPM does not compound over multiple blocks as it does in plain ViTs, which further reduces its effectiveness.

Model	mIoU	fwIoU	mACC	pACC	sec/iter
Swin-T	47.9	72.5	61.6	82.8	0.02
MPM	47.3	72.2	60.9	82.6	0.17

Table 10. Result of MPM applied to a Swin-T with a Mask2Former head on ADE20K.

5. Other Method Configurations

5.1. ALGM

To compare our method with ALGM, we use the open-source implementation. However, the provided speedtest script uses batches of 32 duplicates of the same image (i.e., a tensor of shape (1, H, W, C) is expanded to (32, H, W, C) to form a batch of size 32). Because we found this behavior unconventional, we instead benchmark ALGM within our own framework using batches of 32 random images through the PyTorch *DataLoader*. For dynamic methods such as ALGM, using random batches is naturally slower because batches are padded to match the image with the lowest amount of reduction, and ALGM is highly dynamic (i.e., *hard* images are barely compressed). We acknowledge that random batches of size 32 therefore do not leverage the full capability of ALGM. Nevertheless, we believe this setup is more realistic,

since in real-world applications batches will contain diverse images.

5.2. ToMe

For ToMe, we used the official implementation from the authors. To maintain sufficiently good reconstruction quality, we did not use the aggressive setting from the original paper, where nearly all tokens are eventually merged. Instead, we adjusted the number of merged tokens to align with our MPM configurations. This leads us to merge 30 tokens per block ($r = 30$) for ViT-Tiny through ViT-Base, and 15 tokens per block for ViT-Large. For ADE20K with 512×512 input images, this results in 664 tokens after the last transformer block.

5.3. CTS

For CTS, we used the official implementation from the authors. We used the recommended configuration for segmentation with ViT backbones. This means we reduce the number of tokens by 30% for ADE20K and Pascal Context, and 44% for Cityscapes. The checkpoints were not available in their GitHub repository, so we trained each model from scratch using their official training recipe. We used mixed precision and FlashAttention-2 [3] to speed up training and reduce memory consumption.

6. Qualitative Results

References

[1] Zhe Chen, Yuchen Duan, Wenhai Wang, Junjun He, Tong Lu, Jifeng Dai, and Yu Qiao. Vision transformer adapter for dense predictions. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. 1

[2] Bowen Cheng, Ishan Misra, Alexander G. Schwing, Alexander Kirillov, and Rohit Girdhar. Masked-attention mask transformer for universal image segmentation. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2022, New Orleans, LA, USA, June 18-24, 2022*, pages 1280–1289. IEEE, 2022. 1, 4

[3] Tri Dao. Flashattention-2: Faster attention with better parallelism and work partitioning. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. 1, 5

[4] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. 4

[5] Yuxin Fang, Wen Wang, Binhui Xie, Quan Sun, Ledell Wu, Xinggang Wang, Tiejun Huang, Xinlong Wang, and Yue Cao. EVA: exploring the limits of masked visual representation learning at scale. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2023, Vancouver, BC, Canada, June 17-24, 2023*, pages 19358–19369. IEEE, 2023. 1

[6] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *2021 IEEE/CVF International Conference on Computer Vision, ICCV 2021, Montreal, QC, Canada, October 10-17, 2021*, pages 9992–10002. IEEE, 2021. 4

[7] Chenyang Lu, Daan de Geus, and Gijs Dubbelman. Content-aware token sharing for efficient semantic segmentation with vision transformers. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2023, Vancouver, BC, Canada, June 17-24, 2023*, pages 23631–23640. IEEE, 2023. 4

[8] Narges Norouzi, Svetlana Orlova, Daan de Geus, and Gijs Dubbelman. ALGM: adaptive local-then-global token merging for efficient semantic segmentation with plain vision transformers. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2024, Seattle, WA, USA, June 16-22, 2024*, pages 15773–15782. IEEE, 2024. 4

[9] Jay Shah, Ganesh Bikshandi, Ying Zhang, Vijay Thakkar, Pradeep Ramani, and Tri Dao. Flashattention-3: Fast and accurate attention with asynchrony and low-precision. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024. 1

[10] Robin Strudel, Ricardo Garcia, Ivan Laptev, and Cordelia Schmid. Segmenter: Transformer for semantic segmentation. In *2021 IEEE/CVF International Conference on Computer Vision, ICCV 2021, Montreal, QC, Canada, October 10-17, 2021*, pages 7242–7252. IEEE, 2021. 1, 2, 4

[11] Bolei Zhou, Hang Zhao, Xavier Puig, Sanja Fidler, Adela Barriuso, and Antonio Torralba. Scene parsing through ADE20K dataset. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 5122–5130. IEEE Computer Society, 2017. 4

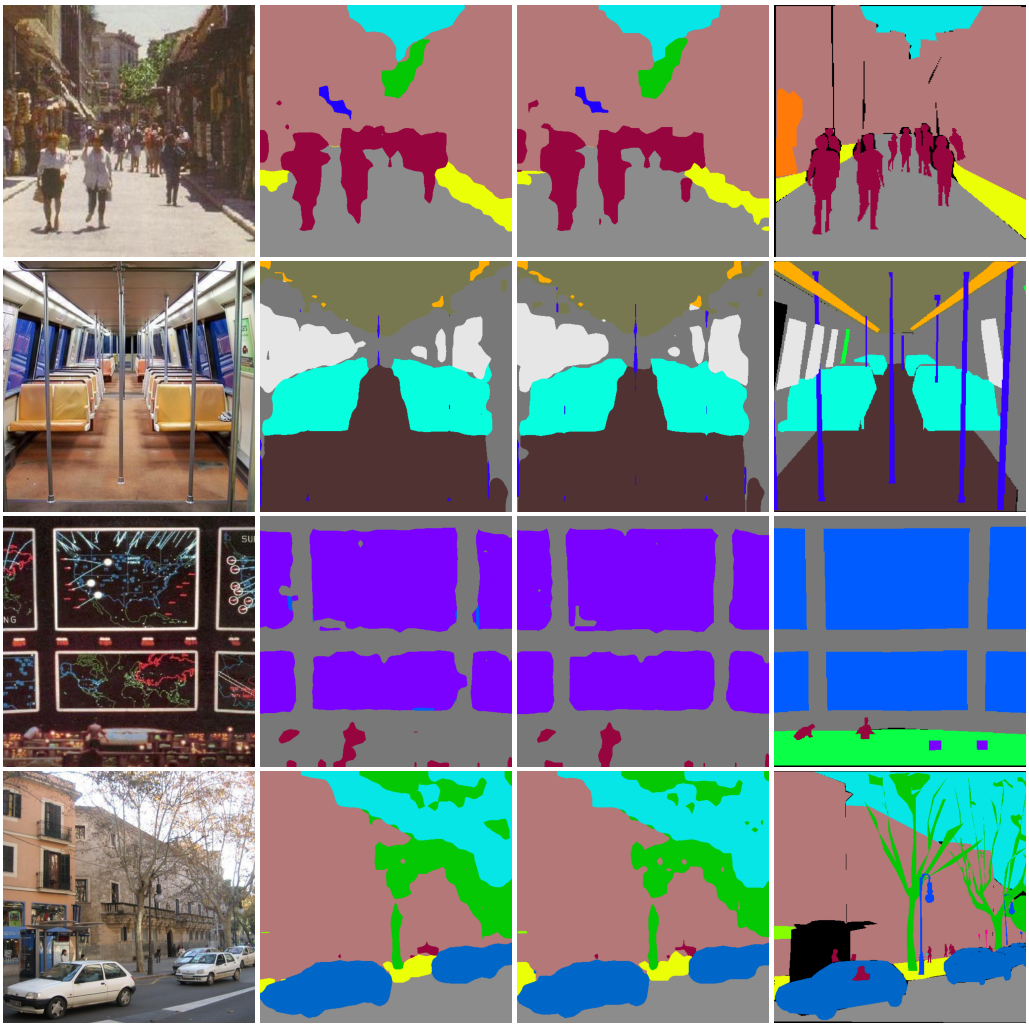

Figure 4. Visualization of segmentation results on ADE20K with ViT-S/16 backbone. From left to right: Input image, Baseline Segmenter, Segmenter with MPM (2,5), Ground Truth.

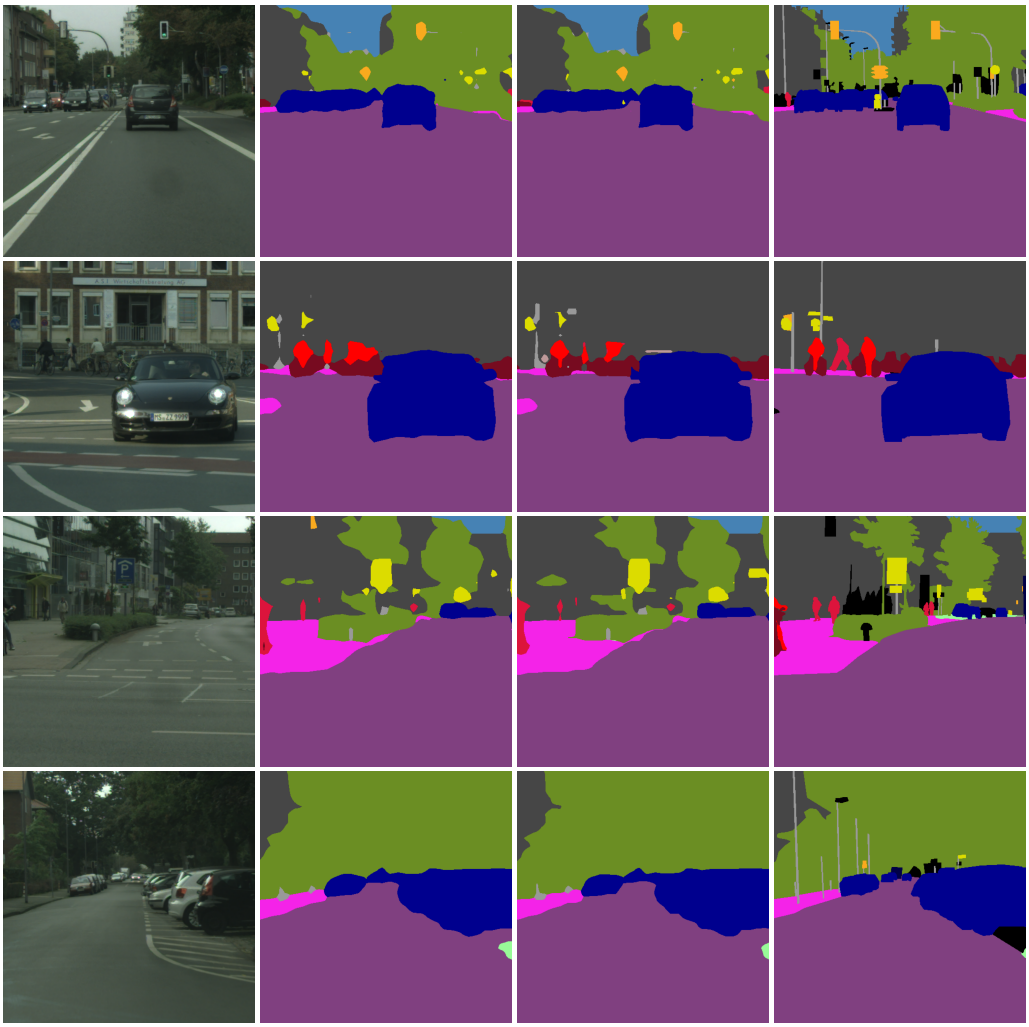

Figure 5. Visualization of segmentation results on Cityscapes with ViT-B/16 backbone. From left to right: Input image, Baseline Segmenter, Segmenter with MPM (2,5), Ground Truth.

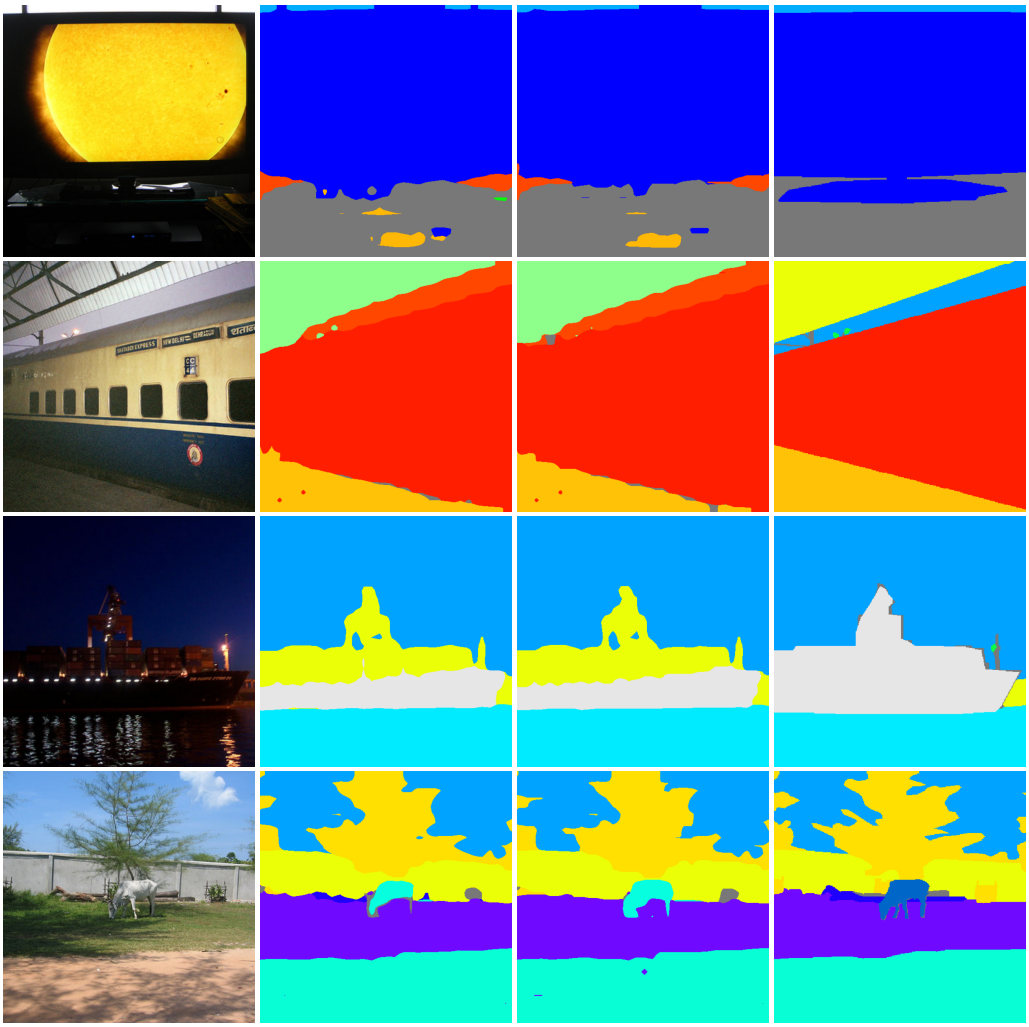

Figure 6. Visualization of segmentation results on Pascal Context with ViT-B/16 backbone. From left to right: Input image, Baseline Segmenter, Segmenter with MPM (2,5), Ground Truth.