ALGM: Adaptive Local-then-Global Token Merging for Efficient Semantic Segmentation with Plain Vision Transformers

Supplementary Material

1. Overview

In this document, we provide the following additional material:

- More elaborate implementation details (see Sec. 2).
- Additional experimental results (see Sec. 3).
  - Comparisons with existing work for a range of token reduction settings, to evaluate the trade-off between efficiency and segmentation quality (see Sec. 3.1).
  - Additional comparisons with existing token reduction methods (see Sec. 3.2 and Sec. 3.3).
  - An analysis of the effect of different similarity thresholds for more datasets (see Sec. 3.4).
  - An analysis of obtaining ALGM* on different segmentation models (see Sec. 3.5).
  - Additional ablations (see Sec. 3.6).
- Qualitative results (see Sec. 4).

2. Implementation details

For our main experiments, we implement ALGM on top of the publicly available code of Segmenter[1] [17], SETR[2] [22], SegViT[3] [20], and EVA[4] [9]. For Segmenter, SETR, and SegViT, we ensure a fair comparison by training all networks using the original hyperparameters and official implementations. When training models with ViT-T/S/B backbones, we utilize 2 Nvidia A100 GPUs, and for ViT-L, we use 4 Nvidia A100 GPUs. When training EVA [9], which comprises 40 transformer layers and 1.0B parameters, a batch size of 32 necessitates 32 GPUs with 40GB VRAM. Given our compute limitations, instead, we fine-tune this model using 4 Nvidia A6000 GPUs with a batch size of 8 for an additional 10k iterations.

GBM module position. Tab. 1 lists the positions of the CLAP and GBM modules in different segmentation networks. As can be seen, the CLAP module is always applied in layer 1 only. To determine the layer where the GBM module should be applied, we implement GBM in several layers on a pre-trained segmentation model without fine-tuning or re-training, and find where it yields the best results. Specifically, we select the earliest layer for which the baseline mIoU is maintained, since (a) our objective is to maintain the segmentation quality, and (b) the potential efficiency gain is higher if GBM is applied earlier, since more layers will process a reduced set of tokens. This approach aligns with strategies followed in existing works [3, 10, 11, 13, 19]. For EVA [9], we apply the GBM module in more than one layer, which is further discussed in Sec. 3.6. The similarity threshold τ is calculated automatically for all models, given the strategy explained in Sec. 3.3 of the main manuscript.

Network	Backbones	CLAP Layer	GBM Layers
Segmenter [17]	ViT-T/S/B	1	5
Segmenter [17]	ViT-L	1	7
SegViT [20]	ViT-L	1	12
SETR [22]	ViT-B	1	5
SETR [22]	ViT-L	1	13
EVA [9]	ViT-G	1	11, 21, 31

Table 1. **CLAP and GBM module positions** in transformer-based semantic segmentation networks Segmenter [17], SegViT [20], SETR [22], and EVA [9].

Adaptive token merging. Our method adaptively determines the number of merged tokens for each image using the similarity threshold τ. As a result, each image in the training and validation sets has a different number of remaining tokens N' and N''. This variability introduces challenges in batch processing. To solve this during training, ALGM is adaptive on a batch level by using the largest value of N' or N'' in the batch. In simpler terms, it merges the same number of tokens for each image in a batch, and this number is the minimum number of mergeable tokens across all images in a batch. This ensures that token reduction is guided by the most complex image in the batch, retaining essential details and eliminating the need for padding. During inference, with a batch size of 1, which is common in real-world situations, ALGM is adaptive per image.

Throughput evaluation. As mentioned in the previous section, the adaptive nature of our method results in a different number of remaining tokens N' and N'' for each image in the validation set. This variability complicates batch processing, which we use for stable throughput evaluation following existing work [14]. Specifically, if we would pad the sets of tokens, or use the largest N' or N'' in the batch like during training, this would give an incorrect image of the obtainable throughput. To solve this, and still allow for

[1] https://github.com/rstrudel/segmenter
[2] https://github.com/fudan-zvg/SETR
[3] https://github.com/zbwxp/SegVit
[4] https://github.com/baaivision/EVA/tree/master/EVA-01/seg

batched evaluation, we use batches of 32 duplicates of the same image, so that the number of reduced tokens is equal throughout the batch. We apply this to each image, and calculate the average throughput over the entire validation set. To ensure a fair comparison, we apply the same approach to evaluate the throughput of existing work. The image crop size used for these calculations is the same that is used during training.

3. Additional results

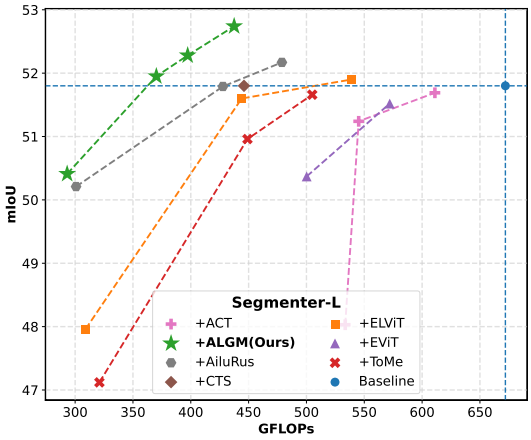
3.1. Comparison with existing work across token reduction settings

In Sec. 5.1 of the main manuscript, we present our main results in which, for different existing token reduction methods, we report the version that achieves the highest efficiency while still maintaining the segmentation quality as much as possible. For a more comprehensive comparison, we compare our ALGM with these existing methods across a range of different token reduction settings, essentially evaluating the trade-off between efficiency and segmentation quality. For methods ELViT [11], EViT [12], ToMe [1], and ACT [21], we follow the different token reduction settings specified by Liang *et al*. [11]. For AiluRus [10], we report their results across three token reduction settings. For our method, ALGM, we present the results for various similarity threshold values during inference. The results for these experiments are provided in Fig. 1 for ADE20K, Cityscapes and Pascal-Context.

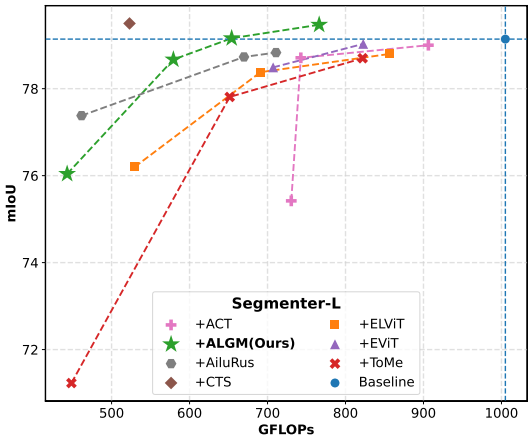
On the ADE20K and Pascal-Context datasets, our ALGM consistently outperforms other methods and achieves a better balance between mIoU and computational efficiency. On the ADE20K dataset, ALGM achieves a mIoU of 51.9, slightly surpassing the baseline, while operating with a 45% reduction in GFLOPs. When compared to its closest competitor, AiluRus [10], our method achieves the same segmentation quality with 14% fewer GFLOPs. On the other datasets, ALGM also achieves a considerably better balance between the mIoU and GFLOPs than existing works. The only exception is CTS [14] on the Cityscapes dataset. As explained in Sec. 5.1 of the main manuscript, this is due to the visual homogeneity of the images of this dataset.

3.2. Comparison with DToP on Pascal-Context

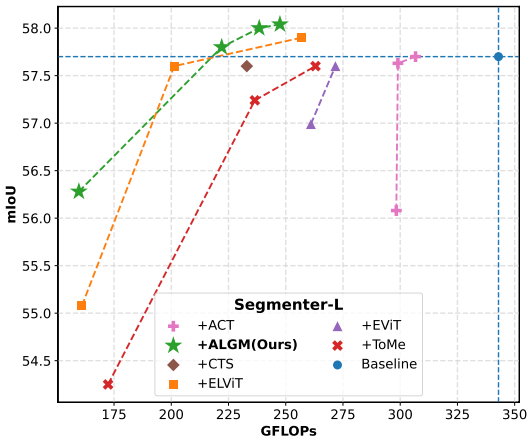
As mentioned in Sec. 5.1 of the main manuscript, the performance of SETR [22] without token reduction as reported by DToP [18] does not align with the results we obtain from the official code of SETR. Additionally, DToP has not made its code publicly available. As a result, we can only compare to this method in terms of the relative performance differences resulting from token reduction. In addition to the results presented in the main manuscript, Tab. 2 compares



(a) **ADE20K validation [23].**



(b) **Cityscapes val [6].**



(c) **Pascal-Context validation [15].**

Figure 1. **Comparison with state-of-the-art methods.** All methods are applied to Segmenter [17] with ViT-L [8]. We compare ALGM to AiluRus [10], CTS [14], ELViT [11], ToMe [1], EViT [12], and ACT [21] across different token reduction settings.

ALGM to DToP on the Pascal-Context dataset [15]. For SETR-B, we observe that ALGM achieves a better segmen-

	No token reduction		With token reduction	
	mIoU↑	GFLOPs↓	mIoU↑	GFLOPs↓
SETR-B + DToP [18]	58.1	92	58.2 (**+0.1**)	69 (**-25%**)
SETR-B + ALGM (ours)	52.2	92	52.7 (**+0.5**)	69 (**-25%**)
SETR-B + ALGM* (ours)	52.2	92	52.3 (**+0.1**)	59 (**-36%**)
SegViT-L + DToP [18]	63.0	315	62.7 (**-0.3**)	224 (**-29%**)
SegViT-L + ALGM (ours)	64.1	334	64.2 (**+0.1**)	259 (**-22%**)
SegViT-L + ALGM* (ours)	64.1	334	64.1 (**+0.0**)	237 (**-29%**)

Table 2. **ALGM vs. DToP [18] on Pascal-Context [15]**, applied to SETR [22], and SegViT [20]. ALGM* is the same trained model as ALGM, but uses the threshold τ during inference that achieves the best efficiency while maintaining the mIoU w.r.t. the baseline.

Method	mIoU (%)↑	Δ mIoU↑	Δ Im/sec
Seg-S [17]	45.3	-	-
+ PAUMER [7]	40.7	-4.6	+50%
+ PAUMER [7]	34.6	-10.7	+100%
+ ALGM (ours)	45.4	+0.1	+50%
+ ALGM (ours)	43.4	-1.9	+100%

Table 3. **ALGM vs. PAUMER [7].** All models are applied to Segmenter (Seg) [17] with ViT-S [8] and evaluated on the ADE20K [23] validation set; Δ indicates differences.

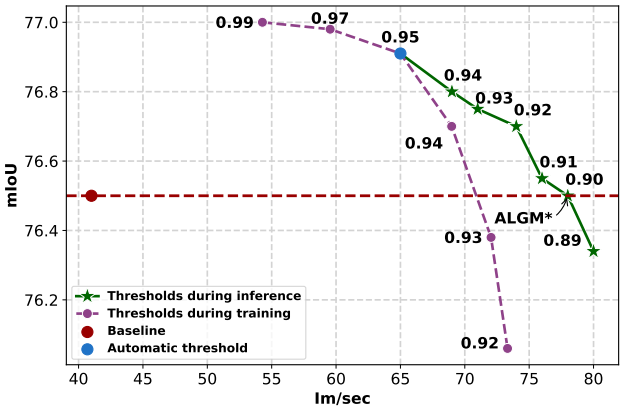
tation quality improvement with the same efficiency. Moreover, ALGM* can achieve a much better efficiency while obtaining the same segmentation quality improvement as DToP. Applied to SegViT-L [20], we find that ALGM* can maintain the mIoU while achieving the same efficiency improvement that DToP obtains while that method causes a mIoU drop. Again, these comparisons highlight that ALGM achieves a better trade-off between segmentation quality and efficiency.

3.3. Comparison with PAUMER

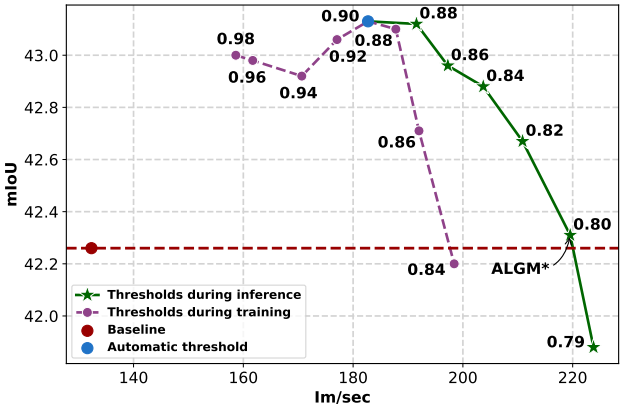
When comparing our method to PAUMER [7], we observe similar challenges as for DToP [18], so we can only compare in terms of relative throughput changes. Tab. 3 presents a comparative analysis between ALGM and PAUMER when applied to Segmenter [17] on the ADE20K validation set. The results show that, at equal throughput improvements, ALGM achieves significantly better mIoU scores. The difference is especially notable when the throughput is increased by +100%, where PAUMER causes a mIoU drop of -10.7 but ALGM can limit the decrease to -1.9.

3.4. Different similarity thresholds

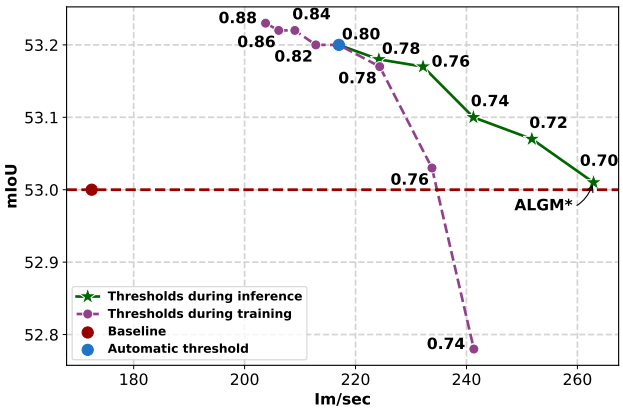
In Sec. 5.4 of the main manuscript, we explore the impact of different similarity thresholds τ on the model's performance on ADE20K. Here, we conduct these experiments also for other datasets. Fig. 2 shows the results for the Cityscapes, COCO-Stuff and Pascal-Context datasets. For all datasets, it is clear that automatic thresholds offer a good balance between efficiency and segmentation quality. Similar to the findings for the ADE20K dataset, we observe that employing a lower threshold during training leads to a significant



(a) **Cityscapes val [6].**



(b) **COCO-Stuff validation [2].**



(c) **Pascal-Context validation [15].**

Figure 2. **Different similarity thresholds for token merging during training and inference.** ALGM is applied to Segmenter [17] with ViT-S [8]. ALGM* is the same trained model as ALGM, but uses the threshold τ during inference that achieves the best efficiency while maintaining the mIoU w.r.t. the baseline.

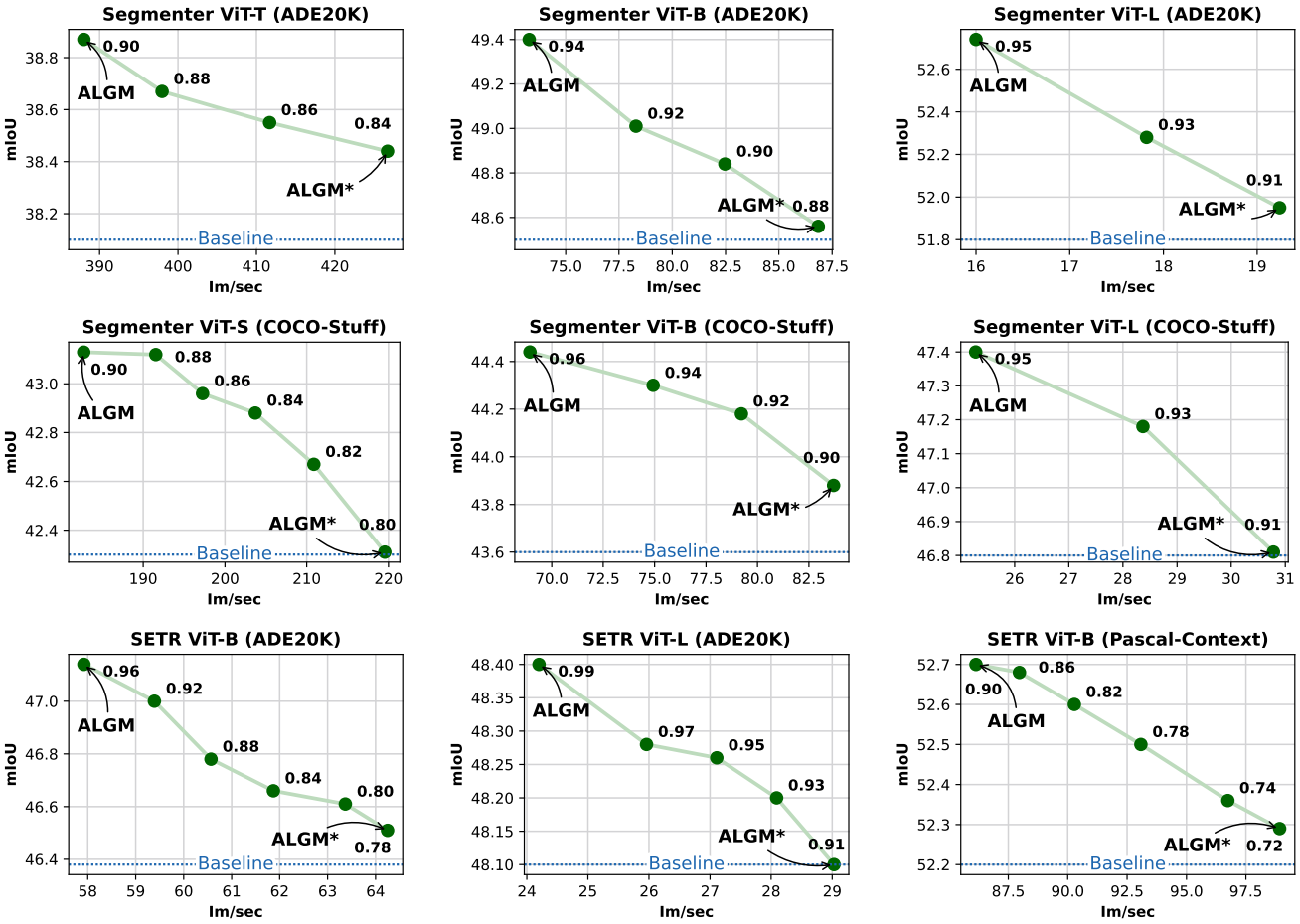


Figure 3. **Obtaining ALGM*.** ALGM is applied to Segmenter [17] and SETR [22] with various backbones (ViT-T, ViT-S, ViT-B and ViT-L) [8] on the ADE20K [23], COCO-Stuff [2] and Pascal-Context [15] validation sets. These figures show the values of thresholds τ used during inference for the ALGM and ALGM* versions. ALGM* is the same trained model as ALGM, but uses the threshold τ during inference that achieves the best efficiency while maintaining the mIoU w.r.t. the baseline.

decrease in mIoU. However, using a lower threshold during inference results in a more modest decline in mIoU while notably improving efficiency. These findings enable a valuable strategy, where we train ALGM with the automatic threshold, but can reduce the threshold τ during inference to improve the efficiency with minimal impact on the mIoU. This how we obtain ALGM*. This also demonstrates the versatility of our method, as it is suitable for various applications with different demands for efficiency and accuracy.

Notably, the automatically calculated threshold τ for the Cityscapes dataset is relatively high compared to the thresholds obtained for other datasets. This observation aligns with our results in Sec. 5.1 of the main manuscript, where we explained that the visual homogeneity of Cityscapes images causes tokens to have high cosine similarities in the first transformer layer, even when they do not depict the same category. This necessitates a higher merging threshold, consequently limiting the potential for efficiency improvement.

3.5. Obtaining ALGM*

In our main experiments, we present two versions of our method: (1) **ALGM**, which consistently applies an automatic threshold during both training and inference, and (2) **ALGM***, which is the same trained model as ALGM but uses the lowest possible threshold τ during inference for which the mIoU remains above the baseline. This ALGM* version is designed to optimize efficiency while maintaining the segmentation quality. To illustrate the process of obtaining ALGM*, Fig. 3 shows the results of ALGM with different thresholds during inference, and compares this with the baseline mIoU performance without token reduction.

Layer	mIoU (%)↑	Im/sec↑	GFLOPs↓
Baseline	*45.3*	*134*	*38.6*
5 & 7	45.4	214	23.1
5	46.4	192	26.3

(a) **Multiple GBM modules.**

Position	mIoU (%)↑	Im/sec↑	GFLOPs↓
Baseline	*45.3*	*134*	*38.6*
After MLP	46.0	187	26.9
Betw. MHSA & MLP	46.4	192	26.3

(b) **Merging module placement.**

Merging method	mIoU (%)↑	Im/sec↑	GFLOPs↓
Baseline	*45.3*	*134*	*38.6*
Pick random token	45.0	193	26.1
Take average	46.4	192	26.3

(c) **Token merging operation.**

Layer	mIoU (%)↑	Im/sec↑	GFLOPs↓
Baseline	*45.3*	*134*	*38.6*
+ ALGM (Direct)	45.3	193	26.2
+ ALGM (Fine-tuning)	45.7	192	26.3
+ ALGM (Training)	46.4	192	26.3

(d) **Effect of training.**

Layer	mIoU (%)↑	Im/sec↑	GFLOPs↓
Baseline	*45.3*	*134*	*38.6*
1	46.4	192	26.3
2	46.4	176	28.5
3	46.3	168	29.8

(e) **CLAP module position.**

Feature	mIoU (%)↑	Im/sec↑	GFLOPs↓
Baseline	*45.3*	*134*	*38.6*
K (Key)	45.9	200	25.5
Q (Query)	45.9	200	25.5
V (Value)	46.0	192	26.1
T (Token)	46.4	192	26.3

(f) **Feature selection.**

Table 4. **Ablations.** We evaluate different settings for ALGM. We apply ALGM to Segmenter [17] with ViT-S [8] and evaluate on the ADE20K validation set [23]. MHSA = Multi-head self-attention.

Method	Layers	mIoU (%)↑	Im/sec↑	GFLOPs↓
EVA [9]	-	61.5	1.9	4080
+ ALGM‡	11, 21, 31	61.5	2.4	3538
+ ALGM‡	11, 21	61.4	2.1	3722
+ ALGM‡	11	61.4	2	3872

Table 5. **Multiple GBM modules in EVA.** ALGM is applied to SOTA method EVA + ViT-Adapter + Mask2Former [4, 5, 9] and evaluated on the ADE20K validation set, with single-scale testing. ‡Directly applied to the backbone without fine-tuning.

3.6. Additional ablations

Multiple GBM modules. In Tab. 4a, we examine the effect of applying the GBM module in more than one layer. The results indicate that while the application of the GBM module in both the 5^{th} and 7^{th} layer significantly increases throughput, it also results in a noticeable reduction in mIoU compared to its sole application in the 5^{th} layer. This implies that overly aggressive global token merging using the GBM module negatively impacts segmentation quality.

For EVA [9], which is a much larger model with 40 transformer layers, we conduct a similar experiment in Tab. 5. Here, we find that applying the GBM module multiple times does not cause a drop in mIoU. We hypothesize that a very large model like EVA introduces considerable additional redundancies in its many layers, which GBM can then reduce without harming the segmentation quality. However, further research is required to explore this in more detail.

Merging module placement. We conduct an ablation to identify the optimal location for the merging modules within a transformer layer. As shown in Tab. 4b, placing them between the multi-head self-attention (MHSA) block and the MLP yields the best performance in terms of both the segmentation quality and the efficiency.

Token merging operation. Tab. 4c compares the performance of different token merging operations. *Pick random token* represents the operation where a single random token is picked from each set of tokens that can be merged, and is used to replace these tokens. This approach results in the loss of important information because the selected token might not be the best representation of the collective set of tokens. On the other hand, taking the average of all tokens in each set yields a much better performance. It causes the merged token to be better representative of the original tokens, because it consolidates the information from all tokens in the set. Moreover, it can denoise these token embeddings, as discussed in Sec. 5.4 of the main manuscript.

Effect of training. Since our method introduces no additional learnable parameters, ALGM can easily be integrated with off-the-shelf pre-trained ViT-based networks to run inference directly while reducing tokens. To assess the impact of training the models after module integration, we explore three scenarios: (a) directly applying the module during inference without any additional training, (b) fine-tuning the model for an additional 16 epochs, resuming training from the model pre-trained on the ADE20K dataset, and (c) training the model for 64 epochs, starting from the model pre-trained on the ImageNet dataset [16]. These are situations that have been evaluated before in earlier work [18]. The results of these approaches are presented in Tab. 4d. We observe that applying ALGM improves efficiency across all scenarios. While direct integration maintains the baseline mIoU, further training, particularly training from scratch, significantly improves the segmentation quality. As we discuss in Sec. 5.4 of the main manuscript, these results indicate that training the model with the ALGM module leverages the benefits of attention balancing and token denoising during training, leading to improvements in segmentation quality.

For further insights into the effect of training, we also compare ALGM against ELViT [11], ToMe [1], and AiluRus [10], which are explicitly designed for direct application without additional training. As shown in Tab. 6,

Method	mIoU (%)↑	Im/sec↑	GFLOPs↓
Seg-L	51.8	10	672
+ ALGM (Direct)	51.8	**16**	**436**
+ ELViT [11] (Direct)	51.9	12	539
+ AiluRus [10] (Direct)	**52.2**	-	479
+ ToMe [1] (Direct)	51.7	14	505
+ ALGM (Training)	**52.7**	**16**	**438**
+ ELViT [11] (Training)	51.4	12	539
+ ToMe [1] (Training)	51.6	14	505
+ AiluRus [10] (Fine-tuning)	52.1	-	-

Table 6. **Effect of training with ALGM and existing work.** All methods are applied to Segmenter (Seg) [17] with ViT-L [8] and evaluated on ADE20K [23]. We evaluate three settings. (1) Direct: direct application without further training. (2) Training: training the model from scratch for 160k iterations. (3) Fine-tuning: resuming training from a pre-trained model for 160k iterations.

although ALGM is primarily designed for training from scratch, it still achieves competitive mIoU results without training, while being more efficient. Furthermore, we observe that training these existing training-free methods does not cause them to perform better than when applied directly. This shows that training a network with token reduction methods does not automatically give a mIoU boost, and that it is the design of our approach that enables this.

CLAP module position. In this experiment, we investigate the impact of applying the CLAP module in different transformer layers. We keep the GBM module at the 5$^{\text{th}}$ layer. The findings, as outlined in Tab. 4e, show that token embeddings in the first layer are sufficiently representative of class correspondence for effective local merging. Delaying the application of CLAP to the second or third layers does not significantly impact the mIoU, but it does negatively affect the efficiency. Thus, applying the CLAP module in the first layer achieves the optimal balance of efficiency and accuracy.

Feature selection. As mentioned in the main manuscript, our approach utilizes the cosine similarity of token embeddings T to identify tokens for merging. Here, we examine the effect of using cosine similarity of other features – *i.e.*, keys, queries, and values – to determine which tokens can be merged. Table 4f shows the results. Although using keys or queries results in the highest throughput, using the tokens T yields a considerably higher mIoU and a throughput that is close to the throughput obtained by using keys or queries. This differs from the findings for ToMe [1], where the keys are identified as the best option to identify mergeable tokens for the image classification task. We hypothesize that this can be attributed to the fact that all token embeddings are directly used to make the final semantic segmentation prediction, whereas image classification networks only use a single *CLS* token for the final class prediction [8]. This gives the tokens a more important role for semantic segmentation, making them the most appropriate feature to use for token reduction.

4. Qualitative results

4.1. Merging operations

In Fig. 4, Fig. 5, Fig. 6, and Fig. 7, we show qualitative examples of the effect of CLAP local merging and GBM global merging. The similarity map displays the average cosine similarity between tokens within a 2×2 local window, which is used to determine which tokens can be merged in the CLAP module. These figures demonstrate that for all datasets, CLAP predominantly merges tokens with a high visual similarity. On the other hand, GBM also merges tokens that depict the same category but have lower visual similarity, after acquiring more informative embeddings in the middle network layers. Notably, from row 3 and 4 in Fig. 7, depicting Cityscapes examples, it becomes clear that the high visual homogeneity of Cityscapes images causes tokens to have high cosine similarities also if they do not depict the same class. As mentioned in Sec. 5.1 of the main manuscript and Sec. 3.1 of this document, this high average cosine similarity causes the automatic threshold τ to be quite high, limiting the efficiency improvement of the ALGM method on this dataset.

4.2. Semantic segmentation predictions

In Fig. 8 and Fig. 9, we show examples of semantic segmentation predictions by Segmenter [17] with ViT-S [8], both with and without our ALGM.

References

[1] Daniel Bolya, Cheng-Yang Fu, Xiaoliang Dai, Peizhao Zhang, Christoph Feichtenhofer, and Judy Hoffman. Token Merging: Your ViT But Faster. In *ICLR*, 2023. 2, 5, 6

[2] Holger Caesar, Jasper Uijlings, and Vittorio Ferrari. COCO-Stuff: Thing and Stuff Classes in Context. In *CVPR*, 2018. 3, 4, 9

[3] Shuning Chang, Pichao Wang, Ming Lin, Fan Wang, David Junhao Zhang, Rong Jin, and Mike Zheng Shou. Making Vision Transformers Efficient from A Token Sparsification View. In *CVPR*, 2023. 1

[4] Zhe Chen, Yuchen Duan, Wenhai Wang, Junjun He, Tong Lu, Jifeng Dai, and Yu Qiao. Vision Transformer Adapter for Dense Predictions. In *ICLR*, 2023. 5

[5] Bowen Cheng, Ishan Misra, Alexander G. Schwing, Alexander Kirillov, and Rohit Girdhar. Masked-attention Mask Transformer for Universal Image Segmentation. In *CVPR*, 2022. 5

[6] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe

Franke, Stefan Roth, and Bernt Schiele. The Cityscapes Dataset for Semantic Urban Scene Understanding. In *CVPR*, 2016. 2, 3, 11

[7] Evann Courdier, Prabhu Teja Sivaprasad, and François Fleuret. PAUMER: Patch Pausing Transformer for Semantic Segmentation. In *BMVC*, 2022. 3

[8] 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 *ICLR*, 2021. 2, 3, 4, 5, 6

[9] 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 *CVPR*, 2023. 1, 5

[10] Jin Li, Yaoming Wang, Xiaopeng Zhang, Bowen Shi, Dongsheng Jiang, Chenglin Li, Wenrui Dai, Hongkai Xiong, and Qi Tian. AiluRus: A Scalable ViT Framework for Dense Prediction. In *NeurIPS*, 2023. 1, 2, 5, 6

[11] Weicong Liang, Yuhui Yuan, Henghui Ding, Xiao Luo, Weihong Lin, Ding Jia, Zheng Zhang, Chao Zhang, and Han Hu. Expediting Large-Scale Vision Transformer for Dense Prediction without Fine-tuning. In *NeurIPS*, 2022. 1, 2, 5, 6

[12] Youwei Liang, Chongjian Ge, Zhan Tong, Yibing Song, Jue Wang, and Pengtao Xie. Not All Patches are What You Need: Expediting Vision Transformers via Token Reorganizations. In *ICLR*, 2022. 2

[13] Sifan Long, Zhen Zhao, Jimin Pi, Shengsheng Wang, and Jingdong Wang. Beyond Attentive Tokens: Incorporating Token Importance and Diversity for Efficient Vision Transformers. In *CVPR*, 2023. 1

[14] Chenyang Lu, Daan de Geus, and Gijs Dubbelman. Content-aware Token Sharing for Efficient Semantic Segmentation with Vision Transformers. In *CVPR*, 2023. 1, 2

[15] Roozbeh Mottaghi, Xianjie Chen, Xiaobai Liu, Nam-Gyu Cho, Seong-Whan Lee, Sanja Fidler, Raquel Urtasun, and Alan Yuille. The Role of Context for Object Detection and Semantic Segmentation in the Wild. In *CVPR*, 2014. 2, 3, 4, 10, 13

[16] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *IJCV*, 115(3):211–252, 2015. 5

[17] Robin Strudel, Ricardo Garcia, Ivan Laptev, and Cordelia Schmid. Segmenter: Transformer for Semantic Segmentation. In *ICCV*, 2021. 1, 2, 3, 4, 5, 6

[18] Quan Tang, Bowen Zhang, Jiajun Liu, Fagiu Liu, and Yifan Liu. Dynamic Token Pruning in Plain Vision Transformers for Semantic Segmentation. In *ICCV*, 2023. 2, 3, 5

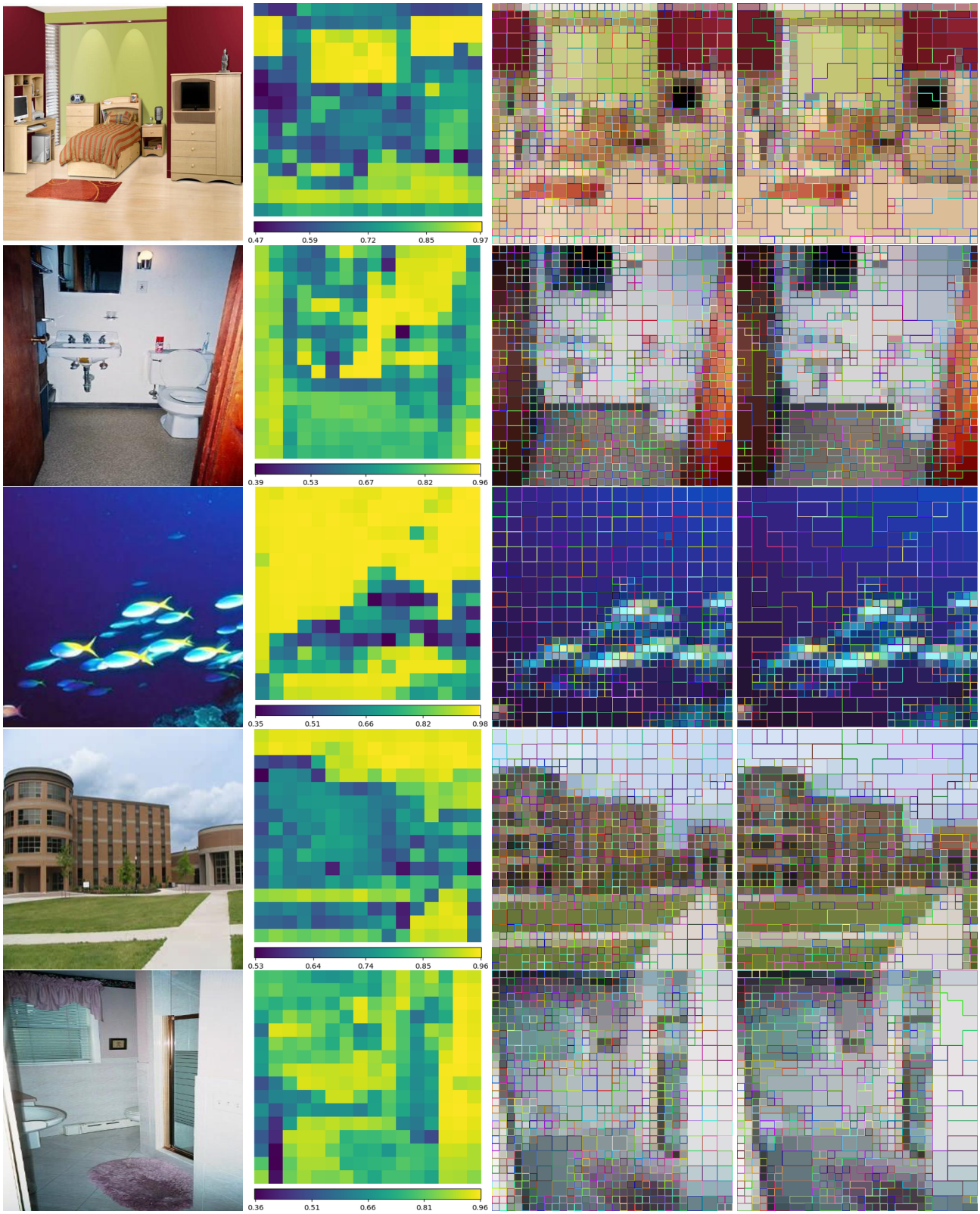
[19] Siyuan Wei, Tianzhu Ye, Shen Zhang, Yao Tang, and Jiajun Liang. Joint Token Pruning and Squeezing Towards More Aggressive Compression of Vision Transformers. In *CVPR*, 2023. 1

[20] Bowen Zhang, Zhi Tian, Quan Tang, Xiangxiang Chu, Xiaolin Wei, Chunhua Shen, et al. SegViT: Semantic Segmentation with Plain Vision Transformers. In *NeurIPS*, 2022. 1, 3

[21] Minghang Zheng, Peng Gao, Renrui Zhang, Kunchang Li, Xiaogang Wang, Hongsheng Li, and Hao Dong. End-to-End Object Detection with Adaptive Clustering Transformer. In *BMVC*, 2021. 2

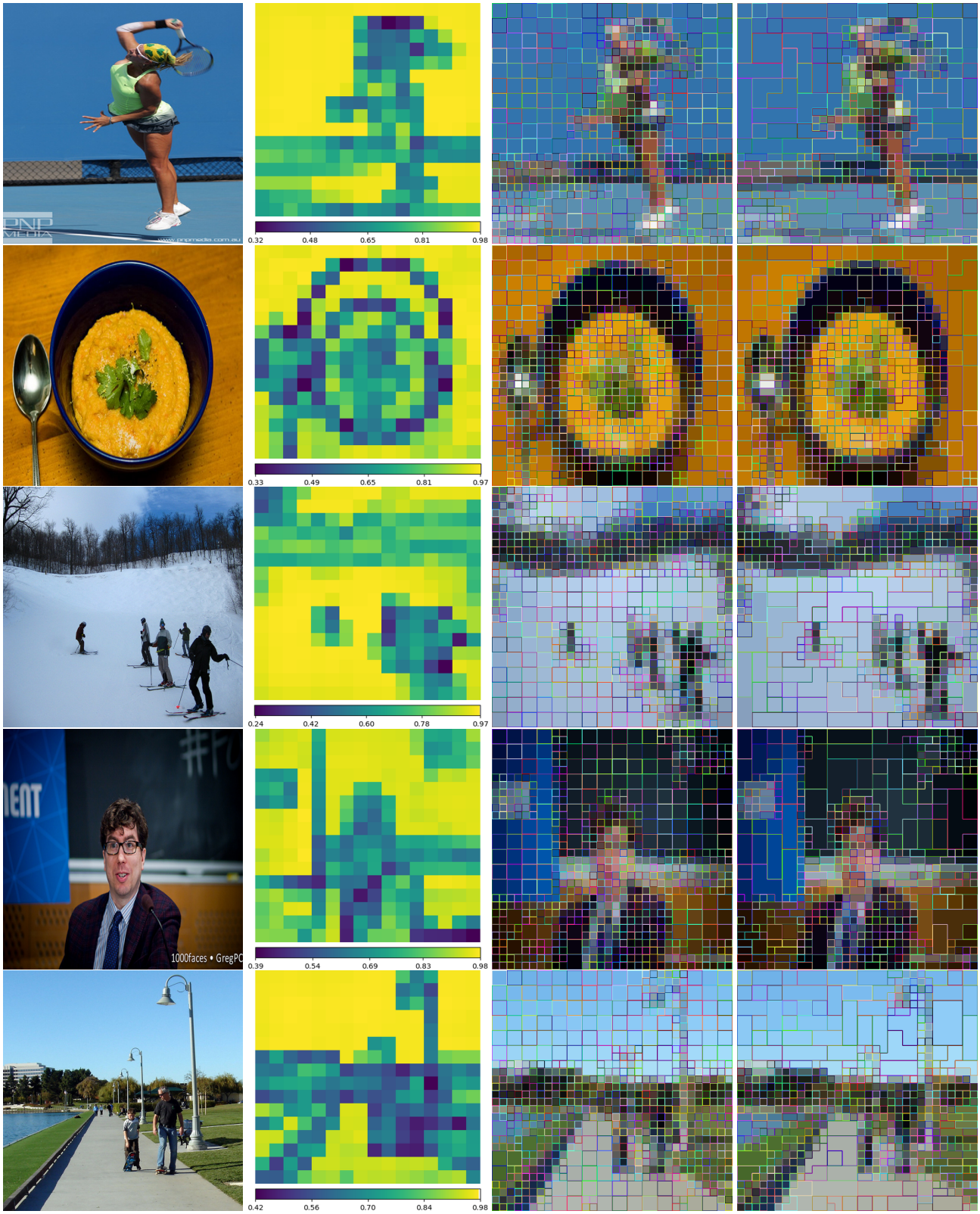
[22] Sixiao Zheng, Jiachen Lu, Hengshuang Zhao, Xiatian Zhu, Zekun Luo, Yabiao Wang, Yanwei Fu, Jianfeng Feng, Tao Xiang, Philip H.S. Torr, and Li Zhang. Rethinking Semantic Segmentation from a Sequence-to-Sequence Perspective with Transformers. In *CVPR*, 2021. 1, 2, 3, 4

[23] Bolei Zhou, Hang Zhao, Xavier Puig, Sanja Fidler, Adela Barriuso, and Antonio Torralba. Scene Parsing through ADE20K Dataset. In *CVPR*, 2017. 2, 3, 4, 5, 6, 8, 12



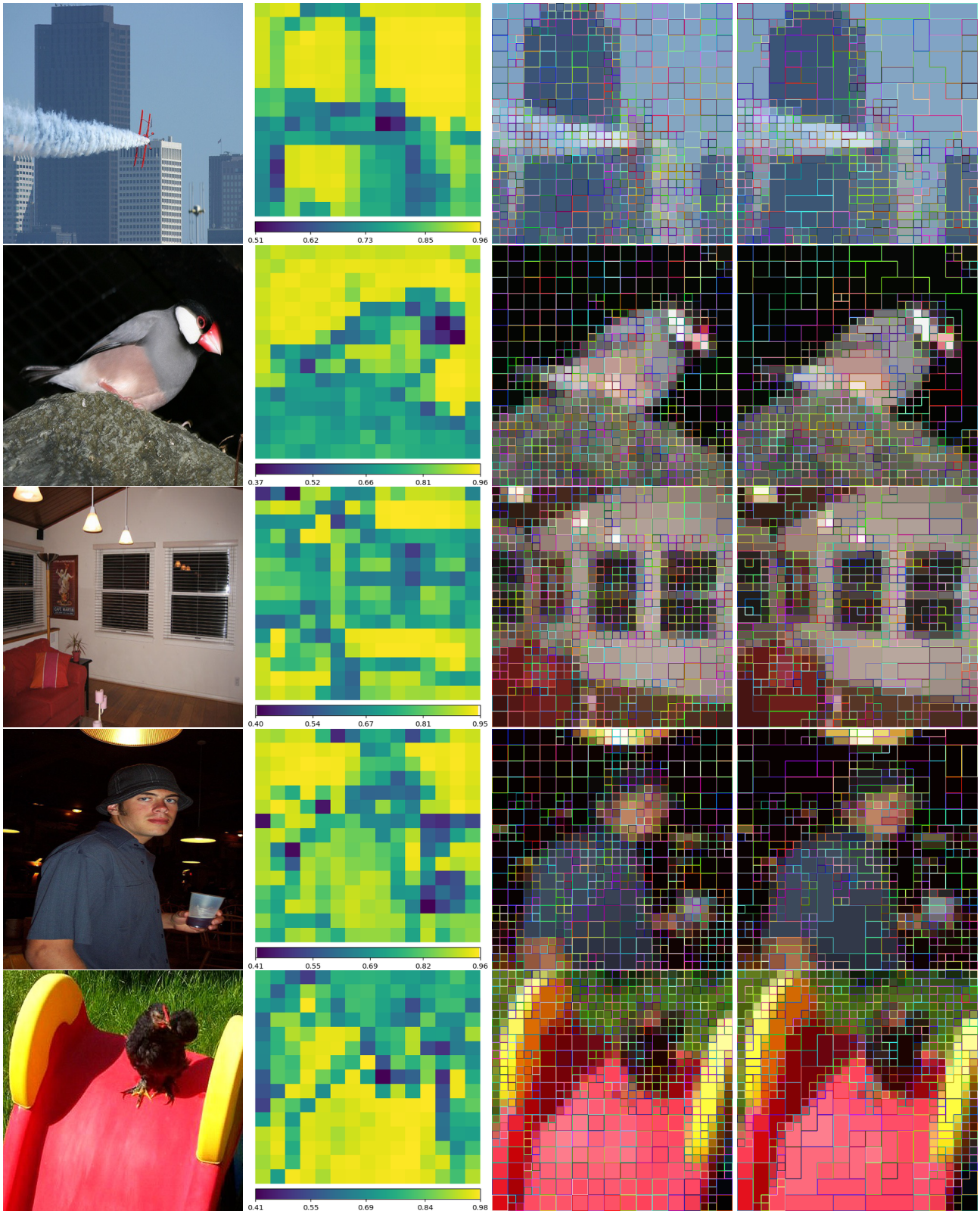
(a) Input image (b) Local token similarities (c) Local merging (d) Global merging

Figure 4. **Qualitative results on ADE20K [23].** This figure displays (a) the input image, (b) the average cosine similarities between tokens in 2×2 local windows in the first transformer layer, (c) the merged tokens after CLAP local merging, (d) the merged tokens after GBM global merging.



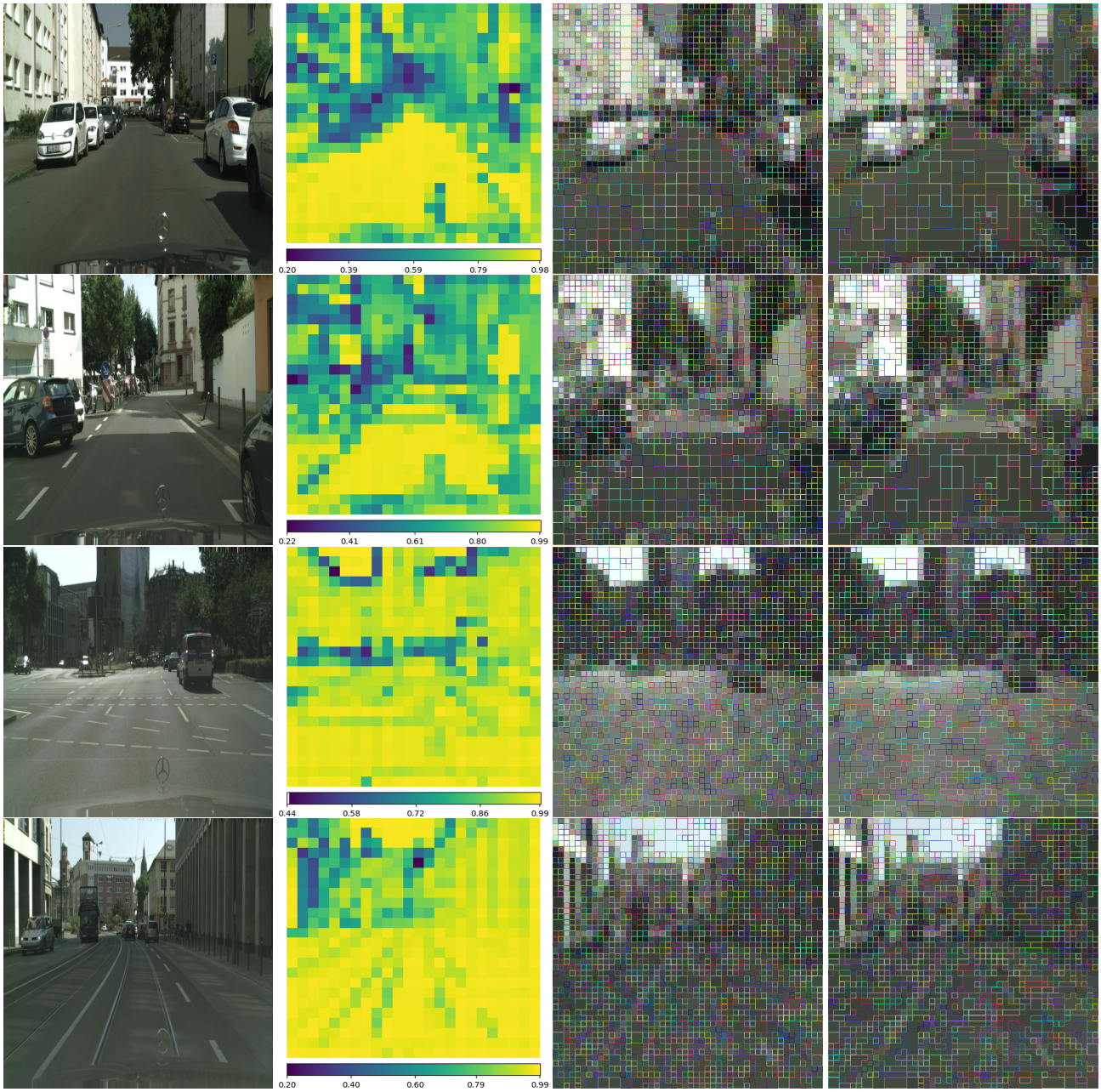
(a) Input image (b) Local token similarities (c) Local merging (d) Global merging

Figure 5. **Qualitative results on COCO-Stuff [2].** This figure displays (a) the input image, (b) the average cosine similarities between tokens in 2×2 local windows in the first transformer layer, (c) the merged tokens after CLAP local merging, (d) the merged tokens after GBM global merging.



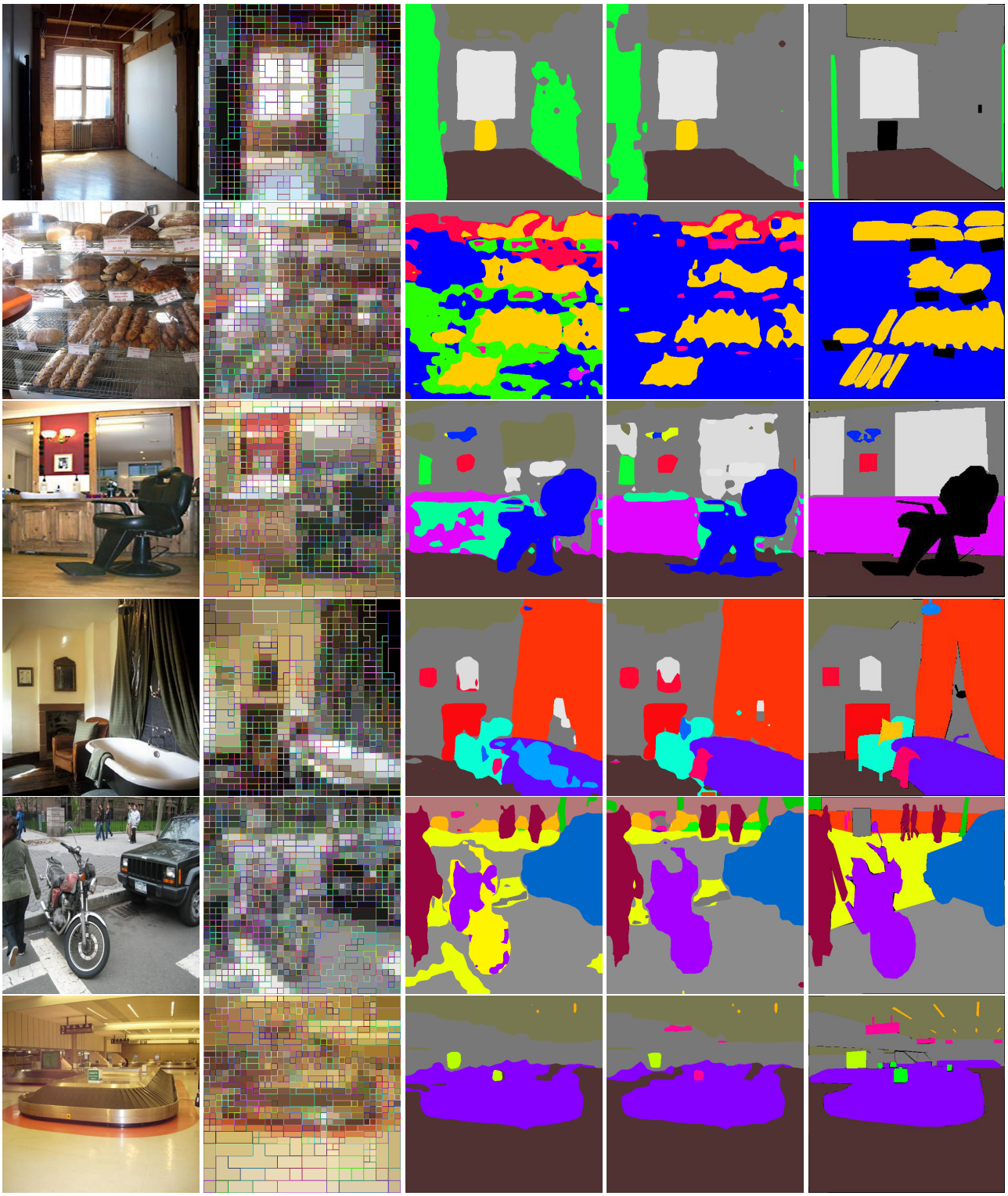
(a) Input image (b) Local token similarities (c) Local merging (d) Global merging

Figure 6. **Qualitative results on Pascal-Context [15].** This figure displays (a) the input image, (b) the average cosine similarities between tokens in 2×2 local windows in the first transformer layer, (c) the merged tokens after CLAP local merging, (d) the merged tokens after GBM global merging.



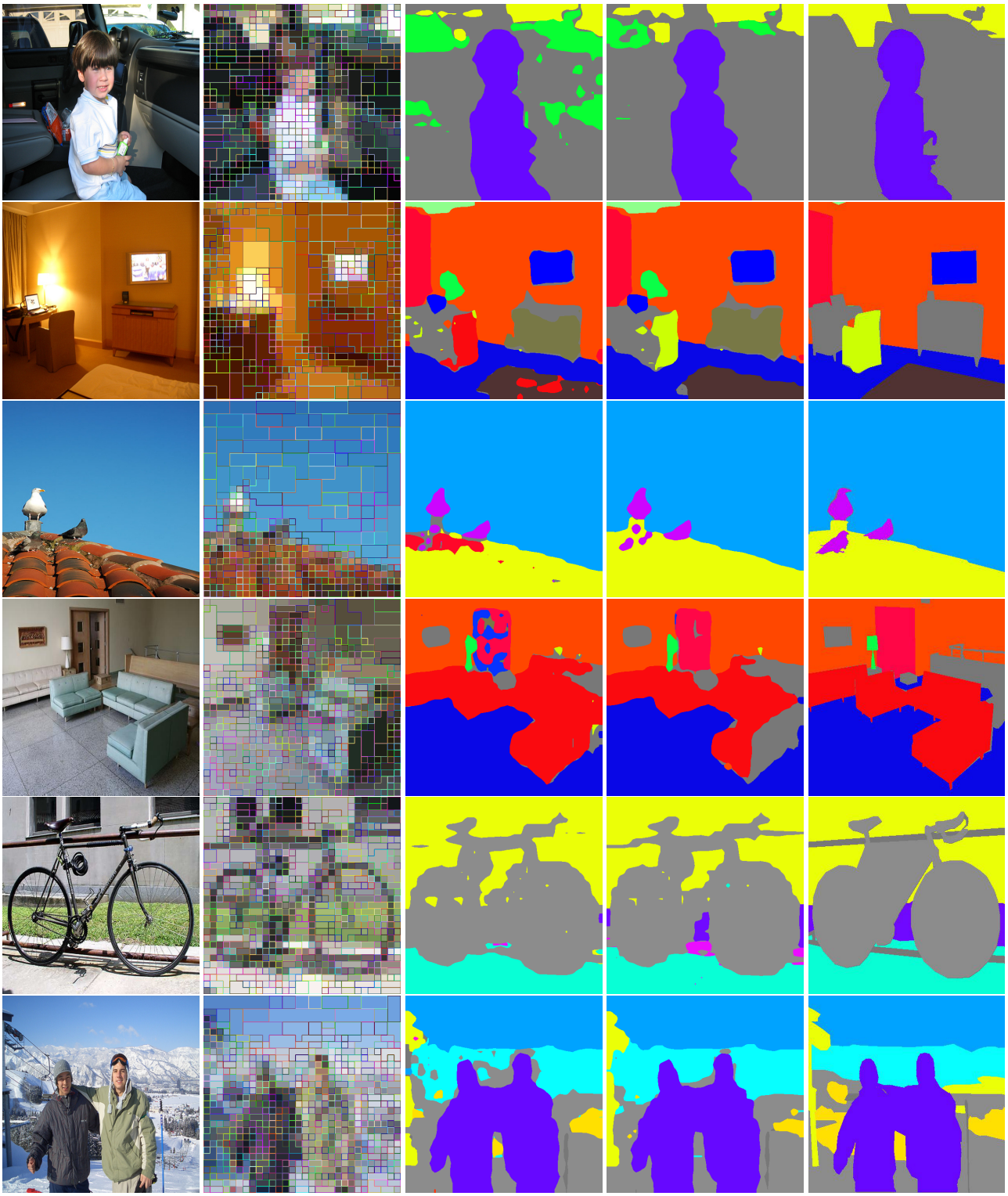
(a) Input image (b) Local token similarities (c) Local merging (d) Global merging

Figure 7. **Qualitative results on Cityscapes [6].** This figure displays (a) the input image, (b) the average cosine similarities between tokens in 2×2 local windows in the first transformer layer, (c) the merged tokens after CLAP local merging, (d) the merged tokens after GBM global merging.



(a) Input image (b) Token merging (c) Baseline (d) With ALGM (ours) (e) Ground truth

Figure 8. **Examples of predictions by Segmenter with ViT-S and ALGM on ADE20K [23].**



(a) Input image (b) Token merging (c) Baseline (d) With ALGM (ours) (e) Ground truth

Figure 9. **Examples of predictions by Segmenter with ViT-S and ALGM on Pascal-Context [15].**