DSFlash: Comprehensive Panoptic Scene Graph Generation in Realtime

Supplementary Material

7. Caveats When Evaluating PSGG Methods

Recent work [2] has discussed a critical flaw in the evaluation protocol of recent PSGG methods. In our paper, we follow their introduced SingleMPO protocol which ensures a correct and fair comparison across models.

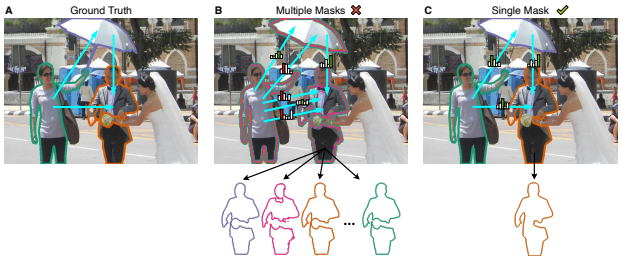

Figure 8. Illustration of the impact of multiple masks for the same ground truth mask. The model in figure **B** predicts multiple very similar masks together with separate relation predictions. However, this gives the model multiple attempts to predict the ground truth relation, essentially ignoring the definition for *mR@k*. Adapted from [2].

Some recent methods that report SOTA performance on PSGG, output multiple masks for the same object, as illustrated in Fig. 8. During evaluation, these methods treat the multiple masks as individual instances and therefore don't merge them. Consequently, multiple relation triplets are predicted for the same ground truth subject-object pair. Per definition of the *mR@k* metric, there is only one prediction per subject-object pair allowed. The faulty implementations do recognize this and ensure that there is only one relation prediction per predicted subject-object pair. However, since the models essentially predict the same subject-object pair multiple times with slightly different masks, their implementation treats them as separate subject-object pairs which are not filtered. Consequently, the model gets multiple attempts to correctly predict the ground truth relation.

For DSFlash, we ensure that multiple masks per subject/object are prevented. This ensures that we don't circumvent the definition of the *mR@k* metric.

8. Intuition and Clarification for the Gating Mechanism

Intuition We use gating because it is a straightforward approach to partition information. DSFlash is forced to split information from the shared feature vector to the forward and backward branch. The same can be achieved with two separate projections (slightly more parameters), but it needs to be learned. In practice we've noticed that both approaches achieve similar performance. The performance improvement in Tab. 2 comes from the ability to reuse information from both relation directions, injested using the training procedure shown in Fig. 4. Without it, only 23.8 mR@50 is achieved, shown in Tab. 2 as "Bidir. Pred."

Feature consistency with asymmetric embeddings This paragraph adds some more clarification for the equations in Sec. 3.3 and why enforcing feature consistency works even though we are using asymmetric embeddings.

In Sec. 3.3, think of x as an embedding that also captures direction, while $t^{\rightarrow}$ and $t^{\leftarrow}$ do not. By disentangling x into $t^{\rightarrow}$ and $t^{\leftarrow}$ through the gate, the directional information is not encoded anymore. Since x' captures the reverse direction: $x \neq x'$. When disentangling x', we can now enforce $t^{\rightarrow} \approx t'^{\leftarrow}$ and $t^{\leftarrow} \approx t'^{\rightarrow}$, because those features don't capture direction anymore.

9. DSFormer Mask Embedding

Two-stage SGG/PSGG methods require information about the location of subject and object boxes/masks. A very common approach is to use RoI align [1] to extract spatial features from the associated region in the feature tensor.

DSFormer replaces this approach with an approach somewhat similar to positional embedding. For each subject-object pair, the model computes an embedding mask that is added to the spatial tokens. This embedding mask is derived from the overlap ratios of the patches.

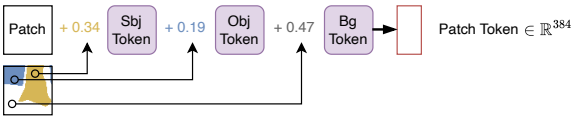

Figure 9. Illustration of the mask embedding. Subject, object, and background tokens are learnable tokens that are added to the patch embedding with a weighted sum. The weights are determined from the proportion of the patch area that is covered by the respective segmentation mask. Adapted from [2].

10. Efficient Mask Embedding

DSFormer's mask embedder implementation uses a convoluted sequence of PyTorch operations including stack, split, flatten, and mean to compute the patch overlap ratios. As Fig. 10 illustrates, this sequence can be simplified by using a simple average pooling layer.

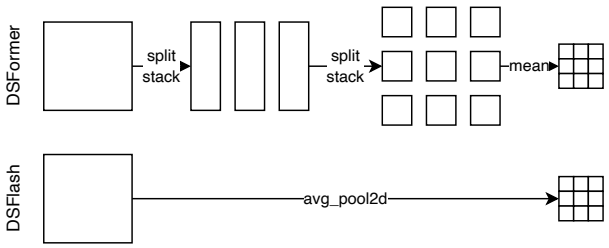

Figure 10. DSFormer's inefficient ratio computation can be represented by a simple average pooling layer.

As discussed in Sec. 9, we need to compute the overlap ratios for every subject-object pair. To this end, DSFormer first copies the segmentation mask for each pair and then compute the ratios for each pair, as illustrated in Algorithm 1. For DSFlash, we first compute the ratios for all segmentation masks and then copy the ratios for each pair instead, depicted in Algorithm 2.

Algorithm 1 DSFormer's mask embedding implementation

Require: image shape (H, W), target shape (H', W'), number of masks N, binary masks $S \in \{0,1\}^{N \times H \times W}$, pairs $P \subset \{1, \dots N\}^2$

$E \leftarrow \emptyset \subset \mathbb{R}^{H' \times W'}$
for $p \in P$ **do** ▷ Performed in parallel
  $s_0 \leftarrow S_{p_0}$ ▷ Allocates additional memory
  $s_1 \leftarrow S_{p_1}$
  $r_0 \leftarrow pool(s_0) \in \mathbb{R}^{H' \times W'}$
  $r_1 \leftarrow pool(s_1) \in \mathbb{R}^{H' \times W'}$
  $E \leftarrow E \cup \{make\_embedding(r_0, r_1)\}$
end for
return E

Algorithm 2 Our mask embedding implementation

Require: image shape (H, W), target shape (H', W'), number of masks N, binary masks $S \in \{0,1\}^{N \times H \times W}$, pairs $P \subset \{1, \dots N\}^2$

$E \leftarrow \emptyset \subset \mathbb{R}^{H' \times W'}$
$R \leftarrow pool(S) \in \mathbb{R}^{H' \times W'}$
for $p \in P$ **do** ▷ Performed in parallel
  $r_0 \leftarrow R_{p_0} \in \mathbb{R}^{H' \times W'}$
  $r_1 \leftarrow R_{p_1} \in \mathbb{R}^{H' \times W'}$
  $E \leftarrow E \cup \{make\_embedding(r_0, r_1)\}$
end for
return E

Since ratios are reused across different pairs, this saves us many calls to the pooling layer while also having to copy less data since the ratio tensors are smaller ($W' < W$ and $H' < H$) than the segmentation mask tensors. While DSFormer copies segmentation masks and computes ratios $2|P|$ times, DSFlash computes N ratios and only copies the ratios $2|P|$ times.

11. Implementation Details

We use the AdamW optimizer with a learning rate of 10^{-5}, weight decay of 0.02, and a cosine annealing scheduler with linear warmup. During training, we clip the gradient norm to a maximum of 1. To achieve best results, we train for 20 epochs.

For five ground truth relation, we sample one negative relation from the dataset, *i.e*. mask-mask pairs that have no annotation in the ground truth.

In addition to the patch tokens and the classification token, DSFlash also uses the same location token from DSFormer. We *don't* use DSFormer's semantic token and *don't* include a frequency bias.

12. Additional SGG Measurements

PredCls Results Tab. 3 contains *mR@50* scores obtained with the Predicate Classification Protocol (*PredCls*). While the base (B) and large (L) variants perform very similar on *PredCls*, their performance on *SGDet* differs considerably. Since the L variant contains a more capable segmentation model, it misses less relations in the *SGDet* setting. For *PredCls*, the ground truth segmentation masks are provided and the difference in the quality of the segmentation model is irrelevant.

Table 3. Scene graph performance, evaluated as *mR@50* on the PSG dataset [4] using Predicate Classification (*PredCls*) and Scene Graph Detection (*SGDet*, aka *SGGen*).

Method	PredCls	SGDet
DSFlash-S	39.27	26.62
DSFlash-B	41.30	28.80
DSFlash-L	41.69	30.90

Additional mR@k measurements Tab. 4 contains additional measurements of selected methods on mR@20, mR@50 and mR@inf. mR@inf is the best possible mR@k that can be achieved based on segmentation performance.

13. Additional Backbone Ablation

In addition to Sec. 4.3, we provide additional backbone ablation results. Fig. 11 shows how DSFlash's scene graph performance correlates with the panoptic quality (*PQ*) of the segmentation model. Using EoMT2 models (*i.e*. pretrained

Table 4. Scene graph performance, evaluated as *mR@20*, *mR@50*, and *mR@inf* on the PSG dataset [4] using Predicate Classification (*PredCls*) and Scene Graph Detection (*SGDet*, aka *SGGen*).

Method	mR@20	mR@50	mR@inf
VCTree	16.3	17.6	45.9
Pair-Net	18.0	19.6	56.4
HiLo-L	17.5	18.3	65.5
DSFormer	27.2	30.7	65.3
DSFlash-L	26.9	30.9	73.6

with DINOv2) as the backbone for DSFlash results in slightly worse results than EoMT3 models. We also observe a very strong correlation of 0.99 between *mR@inf* and *PQ*.

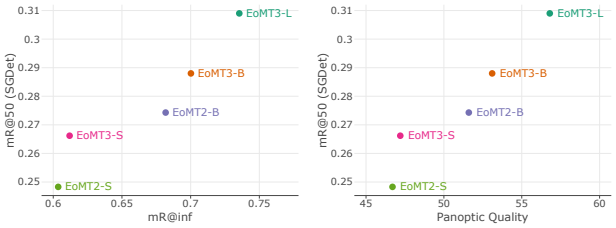

Figure 11. Correlation of scene graph performance (*mR@50*) and backbone capabilities (measured in *mR@inf* and panoptic quality).

14. Training With Inferred Masks

From prior experiments, we've concluded that inferred masks during training don't really improve SGG performance. To prove this for the rebuttal, we have re-run a training of DSFlash-B using inferred masks during training. Counterintuitively, this new model performs worse (22.1 mR@50) than the original DSFlash-B (28.8 mR@50), even when evaluated with *SGGen*, where DSFlash infers its masks on the test set. There are two reasons for this: First, since segmentation models have become very good, ground truth and inferred segmentation masks are quite similar, allowing the model to train on similar masks. Second, when using inferred masks, some ground truth instances are missed and not all ground truth relations can be reassigned, resulting in fewer relation annotations to train with.

15. Train With Pruned Patches/Token Merging

Tab. 5 shows RPS measurements on different hardware. Considering the drop in *mR@50*, there are only minor throughput gains on an H100 and RTX 3090. On a GTX 1080, RPS improves by about 25% compared to the baseline experiment when combining patch pruning and token merging.

Our results in Sec. 4.5 investigate how patch pruning negatively impacts *mR@50* performance of DSFlash. However, when patch pruning is also enabled during training,

Table 5. Impact of Token Merging (*ToMe*) and mask-based dynamic patch pruning (*Prune*, Sec. 3.4) on DSFlash's RPS, measured on a H100, RTX 3090, and GTX 1080 GPU.

Prune	ToMe	H100	3090	1080	mR@50
×	0%	28,663	11,491	671	28.80
✓	0%	26,734	11,933	727	26.67
✓	30%	25,817	11,667	852	26.51
×	40%	26,298	11,154	823	25.82
×	50%	26,475	11,266	845	24.87
×	60%	27,062	11,559	810	21.93

Table 6. Impact of Token Merging (*ToMe*) and mask-based dynamic patch pruning (*Prune*, Sec. 3.4) on DSFlash's latency with batch size 1, measured on a H100, RTX 3090, and GTX 1080 GPU.

Prune	ToMe	H100	3090	1080	mR@50
×	0%	19 ms	29 ms	230 ms	28.80
✓	0%	20 ms	29 ms	205 ms	26.67
✓	30%	20 ms	30 ms	173 ms	26.51
×	40%	21 ms	29 ms	180 ms	25.82
×	50%	20 ms	29 ms	167 ms	24.87
×	60%	21 ms	29 ms	155 ms	21.93

DSFlash retains its scene graph generation capabilities even with pruned patches as shown in Fig. 12, although at a lower overall performance.

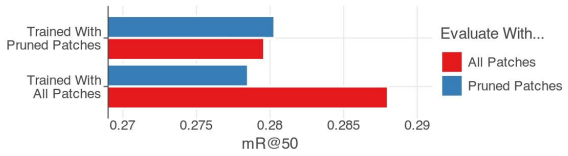

Figure 12. Comparison of a model trained with pruned patches and one without when pruning patches during evaluation.

16. Qualitative Results

Fig. 14 shows some qualitative results. If a label on an arrow has only one line, DSFlash correctly predicted the ground truth predicate. If there are two lines, the bottom line shows DSFlash's prediction while the top line shows the ground truth predicate. The number in parantheses shows at what (0-based) rank DSFlash positions the ground truth predicate. For example, `GT: beside (2)` means that DSFlash's third guess would be "crossing".

Fig. 13 shows some failure cases when using DSFlash. In some cases, DSFlash confuses subjects and objects when predicting the relation. We encourage future work to address this issue and improve DSFlash for these scenarios, for example using contrastive losses [5].

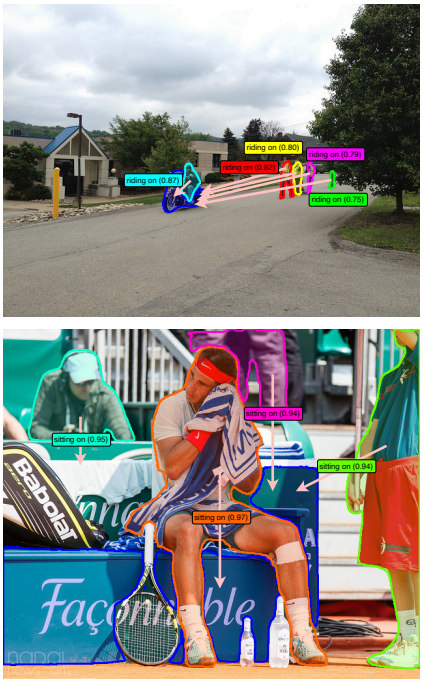

Figure 13. Failure cases when using DSFlash. Shown are predictions that are in the top 50 predictions for the respective image. The color in the boxes indicates to which subject the prediction is related to.

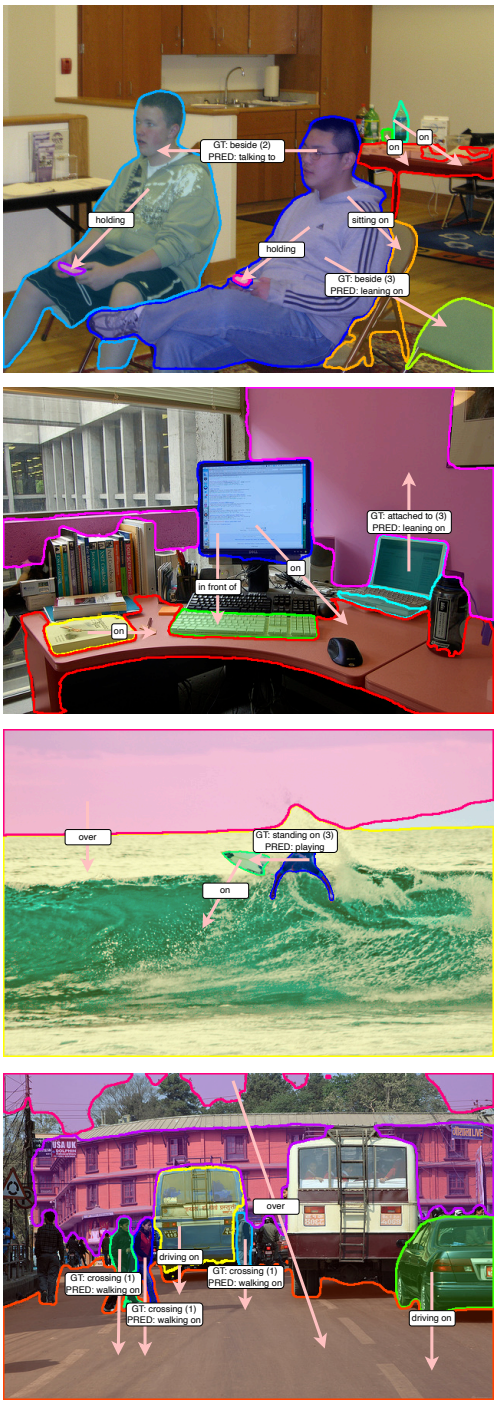

Figure 14. Qualitative results using DSFlash.

References

[1] Ross Girshick. Fast r-cnn. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1440–1448, 2015. 1

[2] Julian Lorenz, Alexander Pest, Daniel Kienzle, Katja Ludwig, and Rainer Lienhart. A fair ranking and new model for panoptic scene graph generation. In *Computer Vision – ECCV 2024*, pages 148–164, Cham, 2025. Springer Nature Switzerland. 1

[3] Maëlic Neau, Paulo E. Santos, Anne-Gwenn Bosser, Cédric Buche, and Akihiro Sugimoto. React: Real-time efficiency and accuracy compromise for tradeoffs in scene graph generation, 2025. 2

[4] Jingkang Yang, Yi Zhe Ang, Zujin Guo, Kaiyang Zhou, Wayne Zhang, and Ziwei Liu. Panoptic scene graph generation. In *ECCV*, 2022. 2, 3

[5] Ji Zhang, Kevin J. Shih, Ahmed Elgammal, Andrew Tao, and Bryan Catanzaro. Graphical contrastive losses for scene graph parsing. In *CVPR*, 2019. 4