PanSt3R: Multi-view Consistent Panoptic Segmentation

Supplementary Material

In the supplementary, we first provide more details on the training procedure in Appendix A. In Appendix B, we provide additional ablative studies of PanSt3R. In Appendix C, we add a comparison with a two-stage approach that first predicts a point cloud reconstruction and then performs 3D point cloud panoptic segmentation. We then show additional qualitative examples of raw 3D panoptic reconstruction results (labeled point clouds) in Appendix D as well as accompanying videos corresponding to PanSt3R+LUDVIG uplifting (Fig. 3 of the main paper). Finally, we present the pseudocode of the PanSt3R model architecture in Appendix E.

A. Additional training details

To select the views for each scene during training, we follow the procedure of MUSt3R [1]. We randomly sample between $N_{\text{min}} = 2$ and $N_{\text{max}} = 10$ views for each batch. During training and inference, the memory of MUSt3R is initialized with 2 views, followed by per-view memory updates for the remaining views. Then we *re-render* all the views using the complete memory to obtain higher-accuracy features and point maps as in [1]. To ensure robustness to different input aspect ratios, we randomly sample from a fixed set of training resolutions. To match predicted instances with GT instances we adapt the Hungarian matching procedure used in Mask2Former [2] to the multi-view setting – mask cost for each pair is computed across all views. We set $\lambda_c = 2, \lambda_d = 5$ and $\lambda_b = 5$ as in [2]. PanSt3R is trained on $8\times$ A100 GPUs, with a total batch size of 16. We train for 200 epochs, with a linear warm-up period of 10 epochs, followed by a cosine learning rate decay from $\eta_{\text{start}} = 10^{-4}$ to $\eta_{\text{end}} = 10^{-6}$.

B. Additional ablative studies

LUDVIG regularization. We study the importance of using regularization during the 3DGS optimization ($\mathcal{L}_{reg}$ loss in Sec. 3.3), before uplifting labels with LUDVIG. As shown in Tab. 6, the regularization helps in many cases (more for the standard mask merging strategy than for QUBO), but not always. Considering visual examples (Fig. 6), we observe that regularization is more useful in the presence of small objects and visually homogeneous regions (*e.g.* walls), where regularization prevents Gaussians from spanning over the boundary of semantic classes or object instances (*e.g.* wall to ceiling).

Number of keyframes. As mentioned in Sec. 4.1, MUSt3R selects a subset of images called keyframes to reduce the computational and memory load at inference time. In Tab. 7, we study the impact of the number of keyframes selected. To assess the influence of this parameter, we evaluate PQ directly the output of PanSt3R on the "seen" set without LUDVIG uplifting. As expected, the performance significantly decreases when there are too few keyframes. Conversely, we observe the performance is plateauing beyond 50 keyframes, indicating that additional information becomes mostly redundant at that scale of scenes in ScanNet++.

Number of "seen" images for building NeRF/GS. In Tab. 8 we ablate the effect of reducing the number of images available for building the GS/Nerf and uplifting the panoptic segmentations, to 100, 50, 25 and 10 images. PanoLift [3] already struggles when the number of "seen" images is reduced to 50, while PanSt3R, both with and without LUDVIG, is less sensitive to shrinking the set of "seen" images, its performance dropping only when very few images (10) are available.

Table 6. Impact of the QUBO merging strategy (Sec. 3.2) and panoptic regularization for 3DGS (Sec 3.3). Results (PQ scores) are reported for PanSt3R+LUDVIG.

Reg	QUBO	Hypersim	Replica	ScanNet	ScanNet++
x	x	58.1	60.8	60.2	50.8
x	✓	66.7	60.7	67.3	52.0
✓	x	59.3	61.2	60.6	55.2
✓	✓	66.3	60.6	67.5	54.7

Table 7. Varying the number of keyframes on ScanNet++ dataset.

# Keyframes	10	20	30	50	100
PanSt3R	49.3	55.0	56.3	58.2	58.2

Table 8. Varying the number of "seen" images for building GS/NeRF on ScanNet++ dataset.

# Number of images	10	25	50	100
PanoLift [3]	3.3	20.2	30.6	38.5
PanSt3R	28.4	51.8	55.8	61.8
PanSt3R + LUDVIG	41.7	56.1	61.6	65.0

Input resolution. We evaluate the impact of the (training and inference) input image resolution in Tab. 9. For this experiment, we train PanSt3R models on ScanNet++

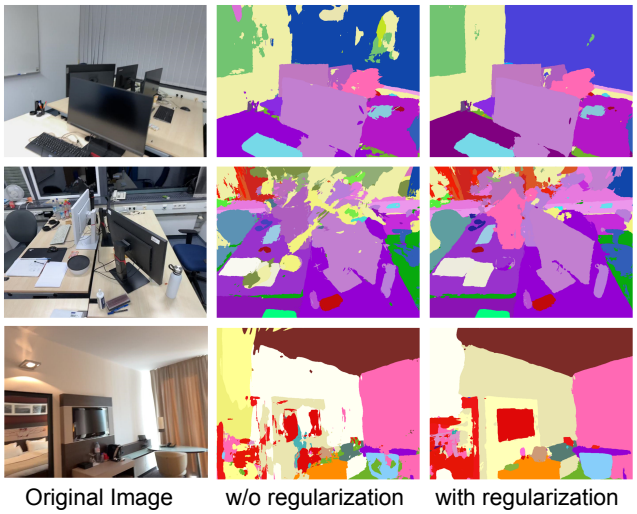

Figure 6. Visual comparison of the effects of panoptic 3DGS regularization on ScanNet++. Results are shown for PanSt3R+LUDVIG using the standard masking strategy (w/o QUBO).

Table 9. Results of PanSt3R trained and evaluated on ScanNet++. We compare models with different resolutions (224 and 512), and different prediction strategies: (i) directly inferring the panoptic segmentation on original test images (orig), (ii) inferring panoptic segmentation on test-view images rendered with vanilla 3DGS (rendered), and (iii) using LUDVIG to uplift PanSt3R predictions. Results marked with † are affected by the square aspect ratio of the 224 model, resulting in missing predictions on the borders (see Fig. 7).

Method	res	PQ	PQ_{th}	PQ_{st}
PanSt3R† (orig)	224	39.0	36.4	48.2
PanSt3R† (rendered)	224	32.5	29.2	41.5
PanSt3R + LUDVIG	224	50.4	45.4	61.1
PanSt3R (orig)	512	57.3	51.7	70.4
PanSt3R (rendered)	512	46.7	43.2	55.8
PanSt3R + LUDVIG	512	54.8	52.4	62.4

only, starting from pre-trained $MUSt3R_{512}$ and $MUSt3R_{224}$ models respectively. We compare PQ scores with and without LUDVIG. We used the QUBO mask merging strategy in these experiments.

Note that the results with $PanSt3R_{224}$ are particularly low due to its limitation of using square input aspect ratios (due to how $MUSt3R_{224}$ was trained). Cropping is thus required for this model that results in missing predictions on the boundary. LUDVIG is able to alleviate this issue via the multi-view consolidation of predictions (see examples in Fig. 7).

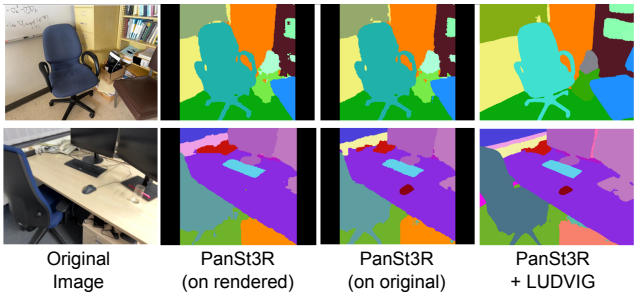

Figure 7. Qualitative panoptic segmentation results with the $PanSt3R_{224}$ model.

C. Comparison with a two-stage approach

We compare our single-feed forward approach with a two-stage approach of first predicting a point cloud and running one off-the-shelf 3D point cloud panoptic segmentation method to label it. For this, we adopt SGIFormer [4], the top performing method with code available from the 3D Instance Segmentation Benchmark Leaderboard[1].To compare with PanSt3R's labeling of the point cloud, SGIFormer is run on the output of 3D MUSt3R head (reconstructed 3D point cloud) and directly on the 3D GT for reference. To evaluate w.r.t. the ground truth, the labels of the predicted point cloud are remapped to the GT point cloud via nearest-neighbor sampling. The results are reported in Tab. 10 and a qualitative comparison is given in Fig. 8. We observe that SGIFormer (and similar methods that learn to directly segment the point cloud) works well on the clean ground-truth point clouds but is very sensitive to noise commonly present in predicted 3D point clouds.

Table 10. Point cloud instance segmentation on ScanNet++V2.

Method	**AP50**	**AP25**	**Pr**	**Re**
MUSt3R pcd + SGIFormer [4]	1.8	3.0	27.5	2.5
PanSt3R	19.1	32.9	45.7	22.9
GT pcd + SGIFormer [4]	33.2	40.5	54.0	39.8

D. More qualitative results

Fig. 9 presents additional qualitative results for direct panoptic prediction on a set of test images. In this case we simply color the predicted 3D points obtained with the 3D head with unique colors corresponding to the predicted object instance IDs.

[1] https://kaldir.vc.in.tum.de/scannetpp/benchmark/insseg

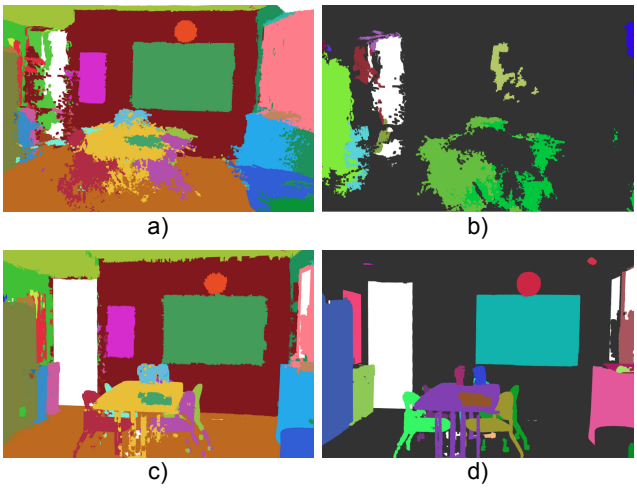

Figure 8. Point cloud labeling: (a) output of PanSt3R, (b) SGIFormer prediction on MUSt3R point cloud, (c) PanSt3R output remapped to the GT point cloud, (d) SGIFormer on the GT point cloud. (Instance colors are not aligned between PanSt3R and SGIFormer)

E. PanSt3R pseudocode

We provide pseudocode of the PanSt3R model architecture in Figures 10, 11, 12 and 13.

References

[1] Yohann Cabon, Lucas Stoffl, Leonid Antsfeld, Gabriela Csurka, Boris Chidlovskii, Jérôme Revaud, and Vincent Leroy. MUSt3R: Multi-view Network for Stereo 3D Reconstruction. In *CVPR*, 2025. 1

[2] Yu Du, Fangyun Wei, Zihe Zhang, Miaojing Shi, Yue Gao, and Guoqi Li. Masked-attention Mask Transformer for Universal Image Segmentation. In *CVPR*, 2022. 1

[3] Songyou Peng and Kyle Genova. OpenScene: 3D Scene Understanding with Open Vocabularies. In *CVPR*, 2023. 1

[4] Lei Yao, Yi Wang, Moyun Liu, and Lap-Pui Chau. SGIFormer: Semantic-Guided and Geometric-Enhanced Interleaving Transformer for 3D Instance Segmentation. *IEEE Transactions on Circuits and Systems for Video Technology*, 35(3), 2025. 2

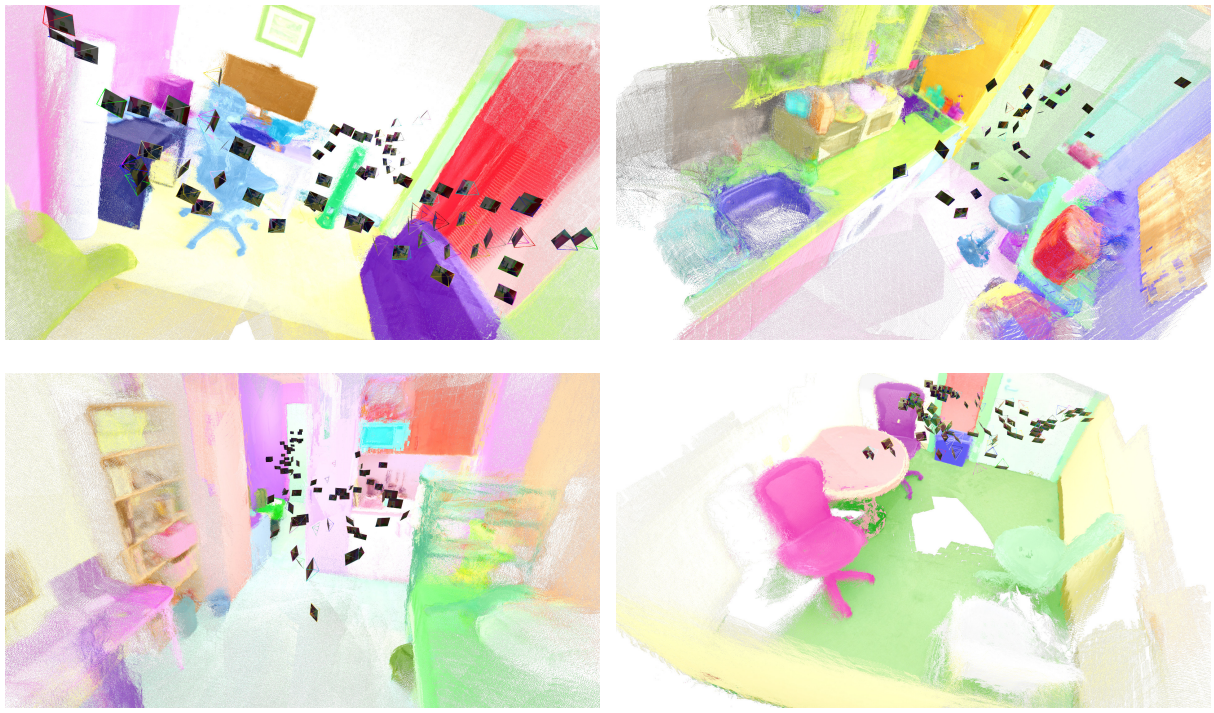

Figure 9. More qualitative examples of PanSt3R's segmented reconstruction. PanSt3R outputs a labeled point cloud in a single forward pass, without requiring any camera parameters nor test-time optimization.

```
def panst3r(frames, class_names):
   # frames: unordered collection of N frame RGB images [N, H=512, W=512, 3]
   # class_names: list of class names used in classification

   # 1.a Extract DINOv2 features (computed per-frame)
   dino_enc_feats = dinov2(frames) # [N, T=HxW/(16x16), D_dino=1024]

   # 1.b Extract MUSt3R features (multi-view)
   must3r_enc_feats, must3r_dec_feats = must3r(frames) # [N, T, D_enc=1024], [N, T, D_dec=768]

   # 2. Concatenate features along channel dimension
   feats = concatenate(dino_enc_feats, must3r_enc_feats, must3r_dec_feats) # [N, T, D_enc+D_dec+D_dino]

   # 3. Prepare frame tokens and mask features
   frame_tokens = MLP(feats) # [N, T, D=768]
   frame_tokens = concatenate(frame_tokens) # [T*N, D]

   mask_feats = upscaler(feats) # [N, H/2, W/2, D_mask=256]

   # 4. Mask transformer
   # in_queries - set of learnable instance queries [Q=200, D]
   output_queries = mask_transformer(in_queries, frame_tokens) # [Q, D]

   # 5. Prediction heads
   out_masks, out_classes = prediction_heads(output_queries, mask_feats, class_names) # [Q, N, H/2,
       W/2], [Q, num_classes]

   # 6. Postprocessing w/ QUBO
   instance_mask, class_mask = QUBO(out_masks, out_classes) # [N, H/2, W/2], [N, H/2, W/2]
```

Figure 10. Pseudo-code of PanSt3R.

```
def mask_transformer(in_queries, frame_tokens):
   queries = in_queries
   for i in range(num_layers):
      # 1. Cross attention with frame tokens
      queries = masked_cross_attention(queries, frame_tokens)

      # 2. Self attention
      queries = self_attention(queries)

      # 3. Feedforward
      queries = feedforward(queries)

   return queries
```

Figure 11. Pseudo-code of PanSt3R: mask transformer architecture.

```
def prediction_heads(output_queries, mask_features, class_names):
   # output_queries: refined queries, output of mask transformer [Q, D=768]
   # mask_features: upscaled feature maps [N, H/2, W/2, D_mask=256]
   # class_names: list of class names used in classification [C]

   class_embeddings = SigLIP_text(class_names) # [C, D]

   # 1. Class prediction (per-query class probability distribution)
   queries_cls = project(output_queries) # [Q, D] -> [Q, D]
   class_probs = cosine_similarity(queries_cls, class_embeddings) # [Q, C]

   # 2. Mask prediction (for each query a set of masks for all frames segmenting the same object
       instance)
   queries_mask = project(output_queries) # [Q, D] -> [Q, D_mask]
   mask_preds = dot_product(queries_mask, mask_features) # [Q, N, H/2, W/2]

   return mask_preds, class_probs
```

Figure 12. Pseudo-code of PanSt3R: prediction heads.

```
def upscaler(feats):
   # Rearrange patch tokens to 2D grid
   feats2d = rearrange(feats) # [N, T=HxW/(16x16), D_enc+D_dec+D_dino=2816] -> [N, H/16, W/16,
       D_enc+D_dec+D_dino]

   for i in range(3):
      # Learnable upsampling, while decreasing the number of channels
      feats2d = upsample2x(feats2d) # [N, h, w, ...] -> [N, 2*h, 2*w, ...]

   return feats2d # [N, H, W, D_mask=256]
```

Figure 13. Pseudo-code of PanSt3R: upscaler architecture.