ProMerge: Prompt and Merge for Unsupervised Instance Segmentation *Supplementary Materials*

We first provide pseudo-code for our approach (Sec. 1) along with further details about our experiments (Sec. 2) and additional ablation studies (Sec. 3). We also visualize qualitative results in Sec. 4.

1 Pseudo-code for ProMerge

In this section, we provide pseudo-code for our method (see Algorithm 1) along with brief descriptions for four components: (i) prompting, (ii) background aggregation, (iii) Cascade filtering, and (iv) merging.

Prompting. Our approach generates initial mask proposals by employing point-prompting on visual features extracted from an image using an image encoder, such as a ViT [4], and creating affinity matrices based on cosine similarities between selected *prompt tokens* and all patch tokens. In this prompting stage, we use a hyperparameter for stride, representing the distance between two neighboring prompt tokens. We use a default value of 4. In our standard implementation, we use DINO [2] features of spatial dimensions 60×60 and generate 225 mask proposals, which we then classify into background and foreground categories.

We use another hyperparameter, τ_b, to threshold the cosine similarity between the embeddings of the seed prompt token and the other patches in the affinity matrix. This step allows us to translate continuous affinities into discrete masks for each seed prompt. For our experiments, we set $\tau_b = 0.2$, unless otherwise stated.

Following the prompting, we split connected components from each foreground mask into separate masks, which helps separate a single large mask covering multiple instances.

Background aggregation. After prompting visual features, binary masks are classified as foreground or background. Background masks are identified by the presence of numerous positive pixels along multiple image edges, and a representative background mask is created using a pixel-wise voting scheme from these candidates.

Cascade filtering. In Cascade filtering, prompted masks are first sorted by area in ascending order, then processed sequentially to track new pixels added by each mask. Masks that significantly overlap with the background, as determined by Intersection over Area (IoA) and feature similarity, are excluded from the merging process. For this, we use hyperparameters, $\tau^{\text{bg}}_{\text{IoA}}$ and τ^{bg}_{f}. If a new mask mainly introduces pixels that intersect with the background or shares a high feature similarity with the background, we do not include it in the merging process. We set $\tau^{\text{bg}}_{\text{IoA}} = 0.8$ and $\tau^{\text{bg}}_{f} = 0.1$ by default.

Algorithm 1 Pseudo-code of ProMerge in a PyTorch-like style

```
# F: features (HxWxC)
# s: stride
# mm: matrix multiplication
# cc_split: connected components splitting
# merge_masks : merge clusters and proposal mask

# Prompting
bg_masks, fg_masks = [], []
for i in range(0, H, s):
    for j in range(0, W, s):
        prompt_token = F[i, j, :]
        mask = mm(F, prompt_token) > τ_b
        if is_background(mask):
            bg_masks.append(mask)
        else:
            fg_masks.append(cc_split(mask))

# Background aggregation
for bg_mask in bg_masks:
    voted_bg += bg_mask
voted_bg = round(voted_bg / len(bg_masks)) # pixel-wise voting

# Cascade filtering
filtered_fg = []
sort(fg_masks, x=lambda x:sum(x)) # sort by size (asc.)
fg_seen = zeros(H,W)
bg_ft = mean(F[voted_bg, :], axis=0)
for fg in fg_masks:
    fg_unseen = fg - fg[fg_seen>0]
    intersection = fg_unseen & voted_bg
    fg_ft = mean(F[fg, :], axis=0)
    if sum(intersection) / sum(fg_unseen) < τ_IoA^bg and
        (bg_ft @ fg_ft) < τ_f^bg:
        fg_seen += fg_unseen
        filtered_fg.append(fg)

# Merging
clusters = set()
sort(filtered_fg, x=lambda x:sum(x), reverse=True)
for fg in filtered_fg:
    masks_to_merge = []
    for c in clusters:
        c_mean = mean(F[c, :], axis=0)
        fg_mean = mean(F[fg, :], axis=0)
        if mm(fg_mean, c_mean) > τ_f^merge:
            masks_to_merge.append(c)
        elif (sum(fg & c) / sum(fg)) > τ_IoA^merge:
            masks_to_merge.append(c)
        if len(masks_to_merge) == 0:
    if len(masks_to_merge) == 0:
        clusters.add(fg)
        continue
    merged_mask = merge_masks(masks_to_merge, fg)
    clusters.replace(masks_to_merge, merged_mask)

# Postprocessing
for mask in clusters:
    mask = dense_crf(mask)
return clusters
```

Merging. In our iterative clustering approach, filtered prompted masks are processed and merged in descending order of area, using the IoA metric and feature similarity. Smaller masks merge with larger ones if their IoA area overlap exceeds $\tau_{\text{IoA}}^{\text{merge}}$ or if their similarity in feature space exceeds τ_f^{merge}. We set $\tau_{\text{IoA}}^{\text{merge}} = 0.1$ and $\tau_f^{\text{merge}} = 0.1$ for our experiments.

2 Further implementation details

Here, we provide additional details about the datasets used in our paper and training of ProMerge+.

Datasets. We evaluate our methods on six benchmarks, including COCO2017 [10], COCO20K [13], LVIS [7], KITTI [5], subsets of Objects365 [12] and SA-1B [9].

COCO2017 and COCO20K are the standard datasets for object detection and segmentation. COCO2017 is composed of 118K and 5K images for training and validation splits respectively, while COCO20K is composed of 20K images. For all results on COCO2017, we use the 5K images in the validation split.

LVIS is a more challenging dataset for object detection and segmentation, with densely-annotated instance masks. We test our performance on the validation set, which contains 245K instances on 20K images. For KITTI and Objects365, we evaluate on 7K images following [14] and a subset of 44K images in the val split, respectively. Lastly, for SA-1B, we assess on a subset of 11K images, which come with 100+ annotations per image on average.[1]

Training details of ProMerge+. For training ProMerge+, we follow the same training protocol as described in CutLER [14], and compare its performance with CutLER after a single training cycle. For a fair comparison, we reimplement a single round training of CutLER using the official codebase. Specifically, we use Cascade Mask-RCNN [1] and initialize the image encoder (i.e., ResNet-50 backbone [8]) with DINO pretrained weights [2]. We also leverage the copy-and-paste augmentation [6]. We train the detector on ImageNet [3] images with their pseudo-labels obtained via ProMerge, after removing noisy masks whose area is smaller than 5% of the size of the corresponding input image. For training, we use a base learning rate of 0.005 for 80K steps that drops to 0.001 for the remaining 80K iterations and use a weight decay of $5e^{-5}$. The training is based on the Detectron2 framework [16].

[1]The subset for SA-1B can be downloaded from https://ai.meta.com/datasets/segment-anything-downloads/

regular grid		random		iterative		attentive		MHA	
AP^{mk}	AR^{mk}_{100}	AP^{mk}	AR^{mk}_{100}	AP^{mk}	AR^{mk}_{100}	AP^{mk}	AR^{mk}_{100}	AP^{mk}	AR^{mk}_{100}
2.4	**7.5**	**2.4**	7.3	**2.4**	5.3	2.0	5.6	2.2	6.3

Table 1: Effect of prompting methods. The regular grid prompting is used by default in our method. Default setting is marked in gray.

3 Further ablation studies

In this section, we conduct further ablation studies regarding the proposed method.

Difference between Cascade filtering and non-maximum suppression. We note that, at a glance, the proposed Cascade filtering (CF) approach can bear resemblance to the commonly used non-maximum suppression (NMS). However, there are crucial differences: CF (i) filters mask proposals by comparing them to a background mask, whereas NMS does so by comparing the proposals to each other; (ii) considers the new pixel regions that are not part of any previously accepted proposals combined; and (iii) takes into account feature similarity with the background as well as pixel overlap. Indeed, we observe the stark difference in AP^{mk}—2.4% for CF vs 0.8% for NMS on COCO2017, showing the importance of pruning noisy background masks via CF.

Prompting methods. In our paper, we use features equally spaced in a regular 2D grid as prompt tokens to obtain initial mask proposals. Here, we explore alternative prompting methods including random, iterative, attentive, and multi-head attention (MHA) prompting.

Random prompting selects patches randomly from all of the patch tokens (extracted from the image encoder) and uses them as prompt tokens. Iterative prompting uses the initial grid of prompts, but shifts the prompt center over multiple iterations. This prompting method first takes the prompt token from a regular grid, and finds the spatial center of the tokens in the initial mask proposal. The token at the mask center is then selected as the new prompt token. This process is repeated three times to find the optimal set of prompt tokens, with the goal of seeking prompts that represent the central components of objects. Attentive prompting identifies distinctive patch tokens, which serve as prompt tokens, by using a mode-seeking clustering algorithm (FINCH [11]). MHA prompting leverages the observation from [2] that the last self-attention layer of the DINO-ViT groups foreground objects. We first compute cosine similarities between the [CLS] token and query features from each head in the last self-attention layer of the DINO-ViT, producing multiple affinity maps. We then sum all the affinity maps and identify 2D coordinates whose cumulative affinities are within the top 5%. We then use these corresponding patch tokens as prompt tokens, and run inference without additional modifications to the ProMerge pipeline.

As shown in Tab. 1, random prompting performs best among these alternative methods, showing notably higher recall than iterative, attentive, and MHA prompting. We conjecture that random prompting tends to cover diverse regions of an image, thus lifting recall. However, the default regular grid prompting that

we employ in ProMerge shows slightly higher recall than random prompting, as it is guaranteed to cover the entire image area, provided that the prompt tokens are sufficiently dense.

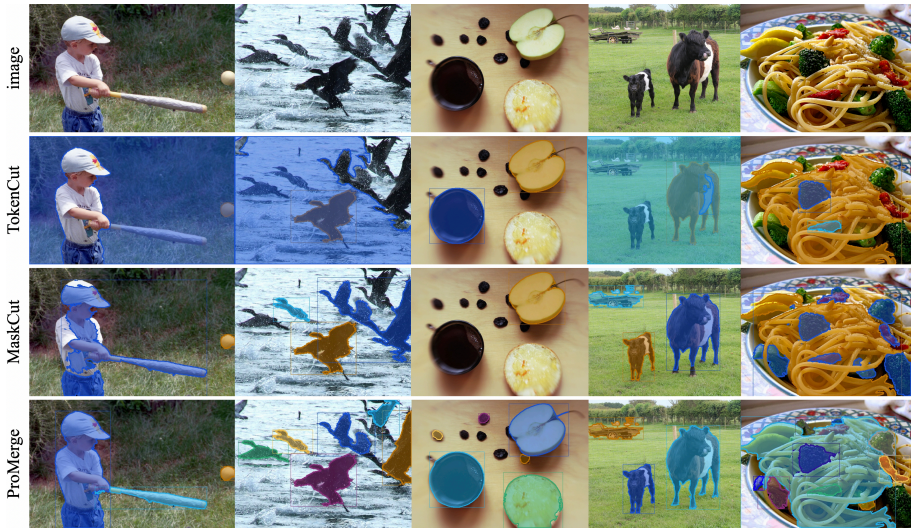

Fig. 1: Qualitative comparison between training-free unsupervised methods. TokenCut [15] and MaskCut [14] fail to appropriately segment multiple instances. In contrast, ProMerge (ours) successfully identifies multiple objects in an image. Zoom in for detail.

4 Further visualizations

Here, we first showcase qualitative examples of training-free methods including ProMerge, TokenCut [15], and MaskCut [14]. Then, we visualize successful and failure cases of our approach.

4.1 Qualitative comparison

In Fig. 1, we can see that both TokenCut and MaskCut struggle with segmenting multiple instances in an image due to their reliance on a predefined number of predictions per image (set to 3 in the original paper), whereas our approach flexibly segments numerous objects according to the input image.

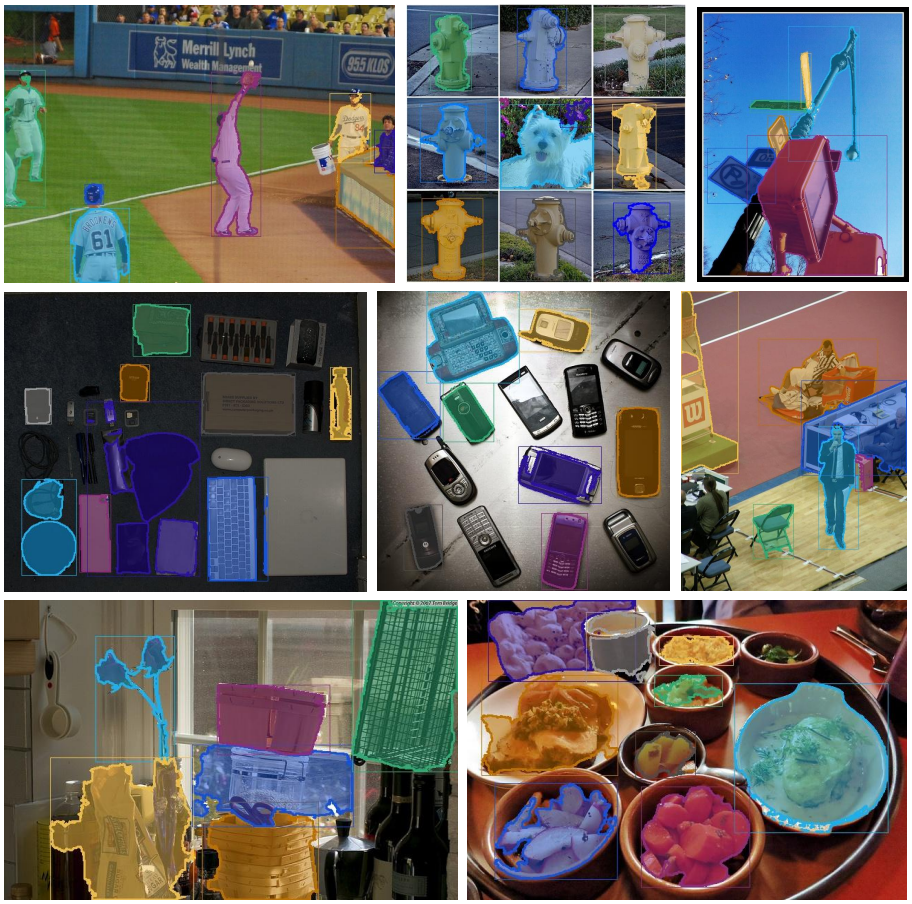

Fig. 2: Successful cases of ProMerge. We provide additional visualizations that highlight ProMerge's ability to segment multiple distinct objects per image.

4.2 Success and failure cases of ProMerge

We show successful cases with multiple instance masks per image in Fig. 2. In Fig. 3, on the other hand, we visualize typical failure cases in which ProMerge undersegments multiple neighboring instances or oversegments an instance due to occlusion. We attribute these artifacts to using visual features that are not explicitly trained for grouping pixels of an object based on an underlying semantic understanding. That is, ProMerge is not aware of the semantic boundaries of a single instance and is thus inclined to make mistakes in regions where multiple objects of the same category, sharing the same color or texture, are adjacent, or different parts of an object are located remotely.

Fig. 3: Typical failure cases of ProMerge. As ProMerge does not use features with an explicit understanding of a concept (i.e., class), ProMerge predicts lower quality masks for multiple instances of the same concept that are densely packed. ProMerge also struggles with recognizing a single object whose parts are scattered across different image regions due to occlusion. For example, adjacent teddy bears with similar textures are segmented as one (bottom left). In the case in which a chair is occluded by a person sitting, ProMerge generates four separate annotations for different parts of the same chair (right image in the second row).

References

1. Cai, Z., Vasconcelos, N.: Cascade r-cnn: Delving into high quality object detection. In: CVPR (2018) 4
2. Caron, M., Touvron, H., Misra, I., Jégou, H., Mairal, J., Bojanowski, P., Joulin, A.: Emerging properties in self-supervised vision transformers. In: ICCV (2021) 1, 4, 5
3. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: Imagenet: A large-scale hierarchical image database. In: CVPR (2009) 4
4. Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., Houlsby, N.: An image is worth 16x16 words: Transformers for image recognition at scale. In: ICLR (2021) 1
5. Geiger, A., Lenz, P., Urtasun, R.: Are we ready for autonomous driving? the kitti vision benchmark suite. In: CVPR (2012) 4
6. Ghiasi, G., Cui, Y., Srinivas, A., Qian, R., Lin, T.Y., Cubuk, E.D., Le, Q.V., Zoph, B.: Simple copy-paste is a strong data augmentation method for instance segmentation. In: CVPR (2021) 4
7. Gupta, A., DollÃąr, P., Girshick, R.: Lvis: A dataset for large vocabulary instance segmentation. In: CVPR (2019) 4
8. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: CVPR (2016) 4
9. Kirillov, A., Mintun, E., Ravi, N., Mao, H., Rolland, C., Gustafson, L., Xiao, T., Whitehead, S., Berg, A.C., Lo, W.Y., Dollar, P., Girshick, R.: Segment anything. In: ICCV (2023) 4
10. Lin, T.Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., Zitnick, C.L.: Microsoft coco: Common objects in context. In: ECCV (2014) 4
11. Sarfraz, M.S., Sharma, V., Stiefelhagen, R.: Efficient parameter-free clustering using first neighbor relations. In: CVPR (2019) 5
12. Shao, S., Li, Z., Zhang, T., Peng, C., Yu, G., Zhang, X., Li, J., Sun, J.: Objects365: A large-scale, high-quality dataset for object detection. In: ICCV (2019) 4
13. Vo, H.V., Pérez, P., Ponce, J.: Toward unsupervised, multi-object discovery in large-scale image collections. In: ECCV (2020) 4
14. Wang, X., Girdhar, R., Yu, S.X., Misra, I.: Cut and learn for unsupervised object detection and instance segmentation. In: CVPR (2023) 4, 7
15. Wang, Y., Shen, X., Hu, S.X., Yuan, Y., Crowley, J.L., Vaufreydaz, D.: Self-supervised transformers for unsupervised object discovery using normalized cut. In: CVPR (2022) 7
16. Wu, Y., Kirillov, A., Massa, F., Lo, W.Y., Girshick, R.: Detectron2. https://github.com/facebookresearch/detectron2 (2019) 4