Training-Free Class Purification for Open-Vocabulary Semantic Segmentation

Supplementary Material

Table 1. Comparison of computation overheads. We report cumulative FPS for each stage on a single NVIDIA 3090 GPU.

Methods	VOC21 (21)	Object (81)	ADE (150)
Baseline	20.0	16.7	4.8
+ Refine	10.0	3.6	2.1
+ Refine + RP	9.1	3.2	1.9
+ Refine + RP + AP	9.1	3.0	1.6

1. Computation overhead

To evaluate the computational cost of our method, we present the cumulative FPS for each stage in Tab. 1. The results show that the main computational overhead comes from the refinement process, while the purification strategy adds minimal cost. Furthermore, as the number of categories increases, the overall computation time also rises. However, the gap between our method and the baseline becomes smaller, demonstrating the efficiency of our approach at larger scales. Besides the network forward, we acknowledge that using LLMs introduces some computational cost. In many practical scenarios for OVSS, the exact set of classes is unknown in advance. However, this can be effectively mitigated by precomputing descriptions using LLM for a very large vocabulary (e.g., ImageNet-21K) offline.

2. Parameters Analysis

We analyze the effect of two thresholds T_{rp} and T_{ap}, designed in RP and AP, respectively. As shown in Tab. 2 and Tab. 3, for object-centric datasets like VOC21 and Object, where the targets are relatively singular, the threshold T_{rp} can often be set to a higher value. However, for semantically complex classes in datasets like ADE and Stuff, a high threshold T_{rp} will filter out a large number of true classes, leading to a decline in performance. Additionally, compared to the VOC, ADE and COCO Stuff are more sensitive to the threshold T_{ap}, resulting in about 2% fluctuations.

Table 2. Parameter Analysis on VOC21 and Object.

T_{rp}	T_{ap}	VOC21 (mIoU)	Object (mIoU)
0.10	0.4	63.9	36.2
0.10	0.5	64.0	36.2
0.10	0.6	64.2	36.2
0.15	0.4	65.4	37.0
0.15	0.5	65.5	37.1
0.15	0.6	**65.8**	**37.2**
0.20	0.4	65.1	36.0
0.20	0.5	65.2	36.1
0.20	0.6	65.2	36.1

Table 3. Parameter Analysis on ADE and Stuff.

T_{rp}	T_{ap}	ADE (mIoU)	Stuff (mIoU)
0.03	0.6	18.1	24.2
0.03	0.7	18.1	24.4
0.03	0.8	17.9	24.1
0.04	0.6	**18.4**	24.6
0.04	0.7	**18.4**	24.7
0.04	0.8	18.2	24.6
0.05	0.6	**18.4**	24.7
0.05	0.7	**18.4**	**24.9**
0.05	0.8	18.3	24.8
0.06	0.6	**18.4**	**24.9**
0.06	0.7	**18.4**	**24.9**
0.06	0.8	18.3	**24.9**
0.07	0.6	18.3	24.7
0.07	0.7	18.3	24.8
0.07	0.8	18.3	24.8
0.08	0.6	18.2	24.4
0.08	0.7	18.2	24.5
0.08	0.8	18.1	24.4
0.10	0.6	17.4	23.8
0.10	0.7	17.4	23.8
0.10	0.8	17.4	23.8

3. LLM-Generated Description

The prompts used for generating descriptions are as follows:

- "How can you visually identify the {class} from its similar objects? Please answer in one sentence using the format: 'The {class} has A, B, C,...', where A, B, and C are noun phrases."
- "What does the {class} look like? Please answer in one sentence using the format: 'The {class} has A, B, C,...', where A, B, and C are noun phrases."
- "What is the unique visual appearance of {class}? Please answer in one sentence using the format: {class} is A, B, C,..., where A, B and C are adjective phrases to describe {class}."

We show some generated samples in Tab. 4:

It can be seen that compared to a simple prompt template, like 'A photo of {class}', the fine-grained descriptions generated by large language models can offer more

Table 4. LLM generated description samples.

Class	LLM generated description samples
Wall	A *wall* can be visually identified by its rectangular shape, typically made of materials like brick, wood, or concrete, and often serving as a barrier or divider. A *wall* has a structure of vertical and horizontal layers of building materials, such as bricks, blocks, or concrete, often with a flat surface for attachment of wallpaper, plaster, or other finishing materials.
Plant	A *plant* can be visually identified by observing its distinct features such as its leaf shape, flower color, stem texture, and overall growth habit. A *plant* has leaves, stems, and roots, which are typically green and surrounded by soil or a growing medium.
Skyscraper	A *skyscraper* has a tall, rectangular structure with setbacks, and typically features a steel frame, curtain walls, and a spire or antenna. A *skyscraper* is a tall, impressive building that reaches into the sky and has multiple stories, glass windows, and a distinctive architecture that varies depending on its location and time period.
bench	A *bench* has a rectangular seat and backrest, usually supported by four legs, often made of wood or metal, and may have armrests or a storage compartment underneath. A *bench* is typically a long, narrow piece of furniture with a flat, horizontal surface and a backrest, often made of wood or metal, and may have armrests or a canopy.

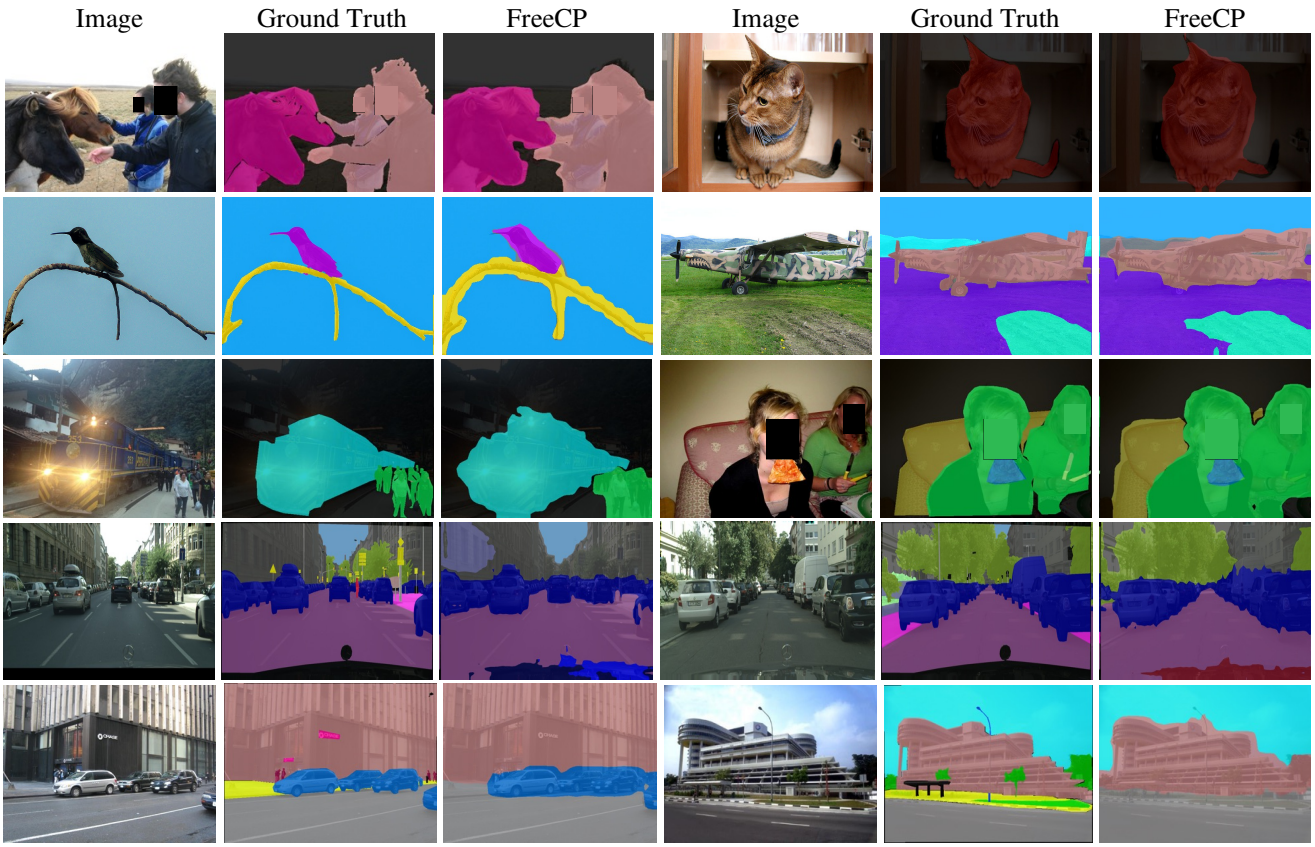

Figure 1. Qualitative results. From top to down are Pascal VOC, Pascal Context, COCO object, Cityscapes and ADE dataset, respectively.

comprehensive and meticulous descriptions of the characteristics of the categories.

4. Results

We also provide visualizations of the segmentation results, as shown in Fig. 1. Our method achieves relatively accurate boundaries and predictions. However, some smaller classes are occasionally missed, highlighting areas for further improvement.

5. PyTorch-like Code

We show the pytorch-like code of FreeCP in Algorithm 1.

Algorithm 1 PyTorch-Like Code for FreeCP

```
def forward(img, text_ori, text_llm, t_rp, t_ap):
    # extract activation maps
    ft_ori, ft_llm = clip_text_encoder(text_ori,
    text_llm)
    fv_class, fv_patch, affinity =
    clip_visual_encoder(img)
    activ_map = Similarity(ft_ori, fv_patch)
    activ_map_ref = matmul(activ_map, affinity)

    # RP
    for m in classes:
        sc_intra = cal_iou(activ_map[m],
    activ_map_ref[m])
        removal_classes = where(sc_intra < t_rp)
        activ_map_ref[removal_classes] = 0

    # AP
    sc_inter = cal_iou(activ_map_ref)
    graph = [1 if sc_inter > t_ap else 0]
    ambigous_group = dfs(graph)
    for n in ambigous_group:
        img_crop = find_conflict_area(img,
    ref_activ_map[n].mean())
        fv_class_crop, _, _ = clip_visual_encoder
    (img_crop, text_llm[n])
        final_class = Argmax(Similarity(ft_llm[n
    ], fv_class_crop))
        activ_map_ref[!final_class] = 0

    return activ_map_ref.argmax()
```