A. Additional Qualitative Results

We provide additional visualizations of the extracted eigenvectors for both the COCO and Cityscapes datasets in Figures 8 and 9. These visualizations follow the same methodology as in Figures 5 and 6, where eigenvectors $\boldsymbol{U}$ are rendered in descending order in groups of three. Each channel of the RGB image corresponds to the value of a particular eigenvector at that coordinate.

B. Per-Image Experimental Details

B.1. Data Preprocessing

For models besides Stable Diffusion [53], and Masked Autoencoder (MAE) [27] image inputs are resized to have a short dimension of 448 pixels. This requires change to the resolution of the learned positional embeddings, which we do through bicubic interpolation, similar to MaskCLIP [77]. In the case of Stable Diffusion, we instead resize images to 512x512 to match the input dimensions of the original model. For MAE, we resize images to 224x224, then upsample the internal query and key matrices to match the spatial resolution of CLIP and DINO.

B.2. Optimization

We optimize features with Adam [33], with a learning rate of 3e-4 or 1e-3, and default PyTorch [50] betas $(0.9, 0.999)$. We take a number of gradient steps to convergence that depends on the model (1000 for CLIP, 2000 for others), but we typically find that 1000 steps is sufficient. Timing information for our method is available in Table 6.

Unlike other models, where there is only a single set of attention matrices per image, the sampling of t in the forward pass of Stable Diffusion introduces more noise and significantly more computation into the optimization. To address this, we cache attention matrices in a buffer of 5 at a time, where the chance to sample a new set of attention matrices is 1/4, and the oldest set in the buffer is replaced by this sample. We also accumulate gradients for 20 backward passes before taking an optimizer step.

B.3. Baselines

To extract regions from TokenCut [71] and MaskCut [70], a single affinity matrix is required. One choice is an affinity matrix constructed from features of the final layer, which is the original proposed matrix for these methods. Another is the final layer's attention matrix. A third alternative is to compute an average over all attention matrices across layers, so as to better compare to our method. We found the third option often led to an ill-conditioned matrix, which could not be solved.

Consequently, we present results for the first two choices. For methods except TokenCut, we find best results with $m = 15$ eigenvectors. For TokenCut we found the performance with $m = 15$ to be subpar, so we use $m = 8$ instead. Quantitative results are available in Table 1. Qualitative results comparing decoding methods can be seen in Figure 10. See Figure 3 for comparison between regions extracted from different models.

Model	Runtime (seconds)
Stable Diffusion 1.4 (w/ buffer) [53]	67
Stable Diffusion 1.4 (w/o buffer) [53]	155
DINO ViT-S/16 [8]	40
MAE/CLIP ViT-B/16 [51]	54

Table 6. **Computation time across models.** We benchmark region computation time for 1000 optimization steps using different models on an NVIDIA A40. 1000 steps are often not required for good results, thus it may be possible to significantly accelerate the pipeline.

B.4. Computational Cost

Table 6 shows the computational cost of running our method, benchmarked on an NVIDIA A40. The extremely long computation time for Stable Diffusion is due to many evaluations of the model during optimization, instead of simply caching the attention matrices from a single forward pass.

C. Full-Dataset Experimental Details

C.1. Data Preprocessing

All experiments take place on COCO-Stuff [5, 40] and Cityscapes [16]. We follow the same preprocessing protocol as adopted in PiCIE [14] and STEGO [24]: images are first resized so the minor edge is 320px and then cropped in the center to produce square images.

C.2. Optimization

In the per-image setting we choose each head to be an independent affinity graph, but that leads to extremely expensive experiments at the full-dataset level. To control this expense, we experiment with a few alternatives: considering each head independently and sampling random layers and heads per iteration of optimization, or concatenating the features for each head into one large vector, which reduces the number of graphs by a factor of 8. The second ultimately led to better results. Due to prohibitive memory costs, we also only consider attention layers with resolutions of 32x32 or coarser. This avoids the large graphs constructed by layers with 64x64 resolution. Due to the prohibitive cost associated with optimizing one set of features per image in the dataset, we restrict our dataset-level clustering to the validation set only.

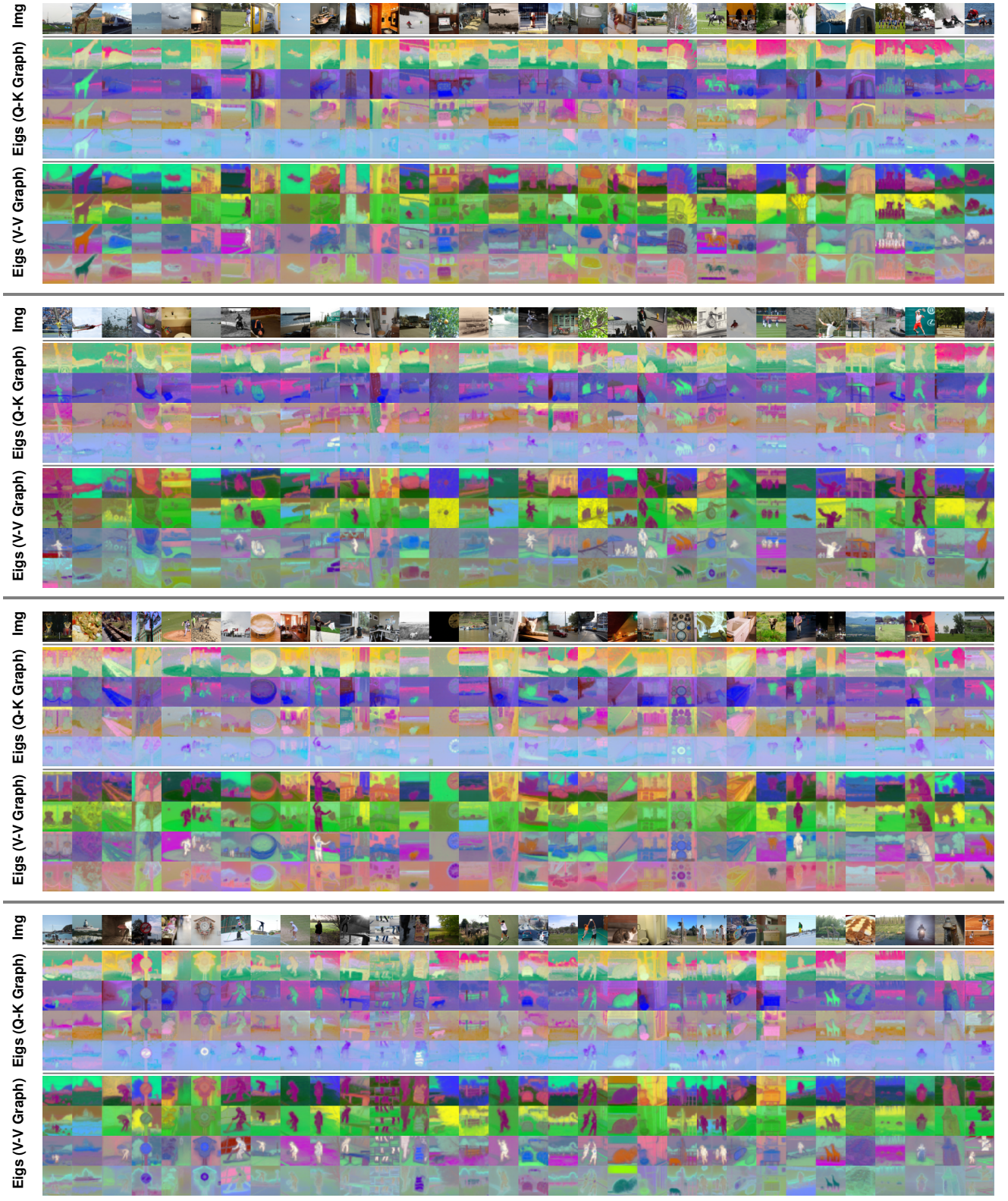

Figure 8. **More examples of extracted eigenvectors on COCO for both graph choices.** We visualize selected components of $\boldsymbol{X}_{\text{ortho}}$, sorted by decreasing eigenvalue. Three eigenvectors at a time are rendered as RGB images.

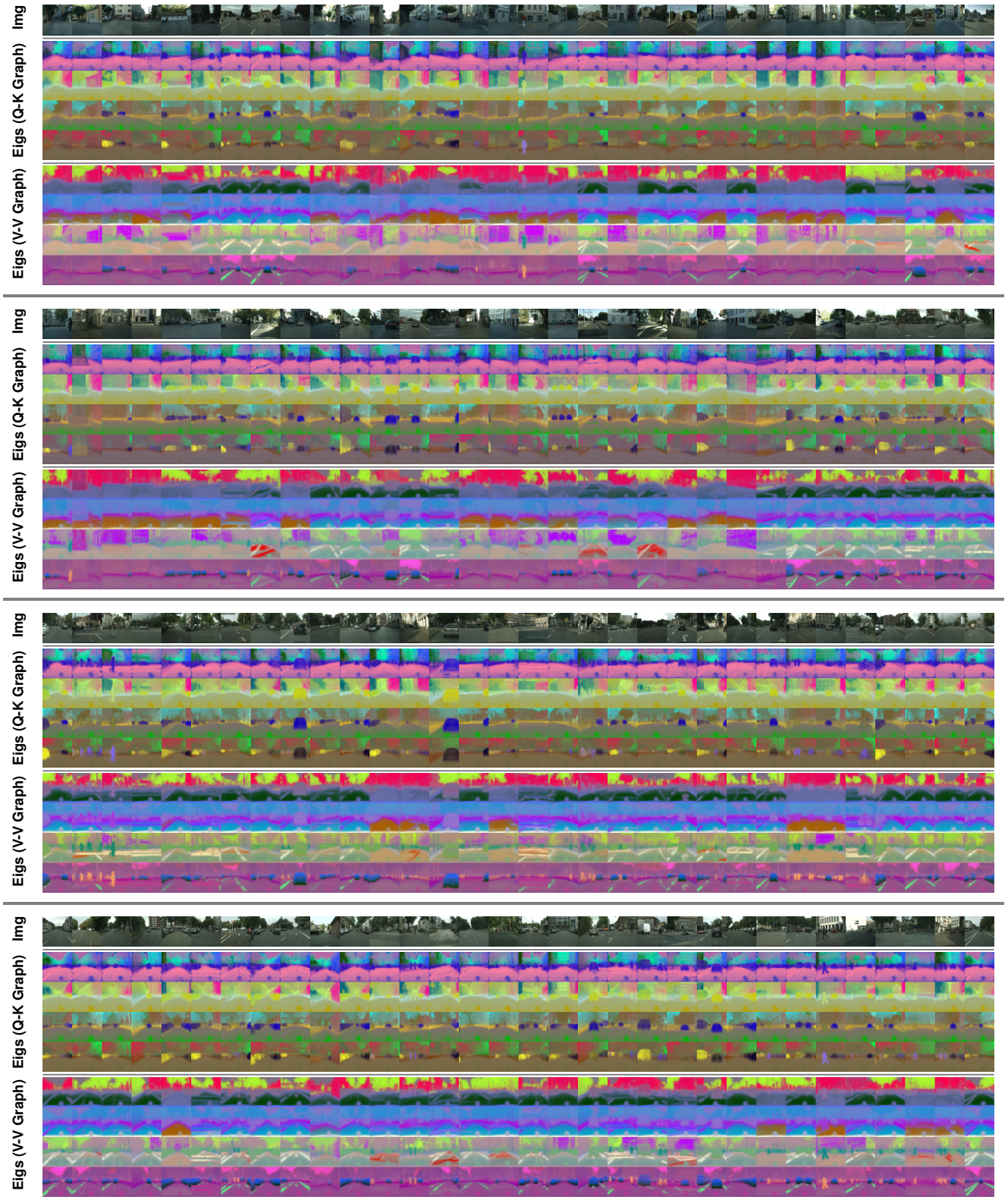

Figure 9. **More examples of extracted eigenvectors on Cityscapes for both graph choices.** We visualize selected components of $\boldsymbol{X}_{\text{ortho}}$, sorted by decreasing eigenvalue. Three eigenvectors at a time are rendered as RGB images.

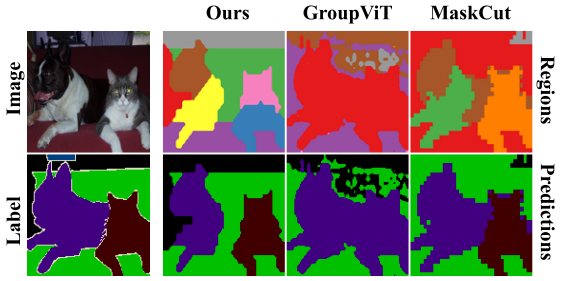

Figure 10. **Examples of different segmentation methods on PASCAL VOC [21].** All methods besides GroupViT [73] use DINO [8] features or attention. Ours can generate diversified regions while maintaining accurate object borders. In contrast, GroupViT [8] tends to generate a noisy boundary, while MaskCut [70] can miss subtle boundaries.

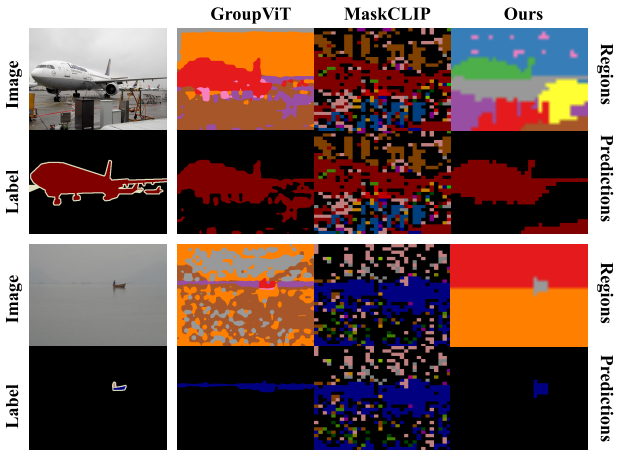

Figure 11. **Examples for different methods on zero-shot semantic segmentation.** Notice the tendency of GroupViT [73] and MaskCLIP [77] to break up objects, and the eagerness of MaskCLIP to cover the image. On the airplane image we perform slightly worse than GroupViT but our regions have more spatially coherent structures. On the boat image our method has better performance and can even separate water and sky, though the gap between their pixel values is almost imperceptible.

C.3. Evaluation

Unsupervised semantic segmentation. We consider $\boldsymbol{X}_{\text{ortho}}$ as features for our method. We also compare with several baseline methods by collecting backbone features from a number of different models: STEGO, DINO and Stable Diffusion. For Stable Diffusion we choose the most semantic features in the model, as measured by semantic correspondence performance in prior work [63]. For DINO we take features at the last layer, like prior work [24, 70, 71]. For STEGO, we use output just before the linear head that projects to the number of clusters.

After obtaining features, we cluster with $K = 27$, the number of ground truth categories in both datasets, for K-Means over $\boldsymbol{X}_{\text{ortho}}$. We report results with mIoU and compare to other methods in Table 3.

X-Y coordinate regression. After extracting features, we use a random sample of 80% of the features to learn a linear regression model onto the X-Y coordinates of a 32 x 32 grid, and check performance on the remaining 20%. For methods where the features are of a different resolution, we resize bilinearly.

D. More Applications of Per-Image Regions

D.1. Adapting CLIP for Open-Vocabulary Semantic Segmentation

As a more interesting case-study than oracle decoding, we assess our regions for zero-shot semantic segmentation on PASCAL VOC [21]. In order to form class decisions, we follow insights from GroupViT [73] and MaskCLIP [77]. First we compute regions on top of CLIP ViT-B/16, then we take the final value vectors from the last attention layer as pixel-wise features, similar to MaskCLIP. We compute region-wise features by averaging pixel-wise features over the regions they correspond to, then compute cosine similarities between these region-wise features and the text embeddings of CLIP, where per-class text embeddings are computed by an average over many different prompts like *"a photo of a {class name}, a picture of a {class name}, ..."*, as is done in GroupViT. Finally we threshold these similarities by a fixed number (0.7), and set all regions to their most similar class, where regions with no similarity greater than the threshold are assigned background. We compare to MaskCLIP [77], a training-free approach, as well as GroupViT [73], which proposes modifications to the original CLIP architecture in order to better suit segmentation.

Method	Segmentation-specific?	Model	mIoU
GroupViT [73]	Yes	modified ViT-S/16	0.53
MaskCLIP [77]	No	ViT-B/16	0.25
Ours	No	ViT-B/16	0.50

Table 7. **Zero-shot segmentation on PASCAL VOC [21].** Our method is stronger than the MaskCLIP baseline, and competitive with GroupViT, whose architecture is tailored to segmentation.

We see in Table 7 that, even without a segmentation-specific training objective, we can achieve competitive performance on PASCAL VOC [21], and our region-extraction pipeline aids in segmentation on top of CLIP [51]. We emphasize that this is possible *without any segmentation-specific objectives or additional training*.

Our regions are often contiguous and large in size, while GroupViT's regions contain holes. As a result, the errors that CLIP makes in localizing certain classes may be magnified by our regions. This can be seen in per-class IoU

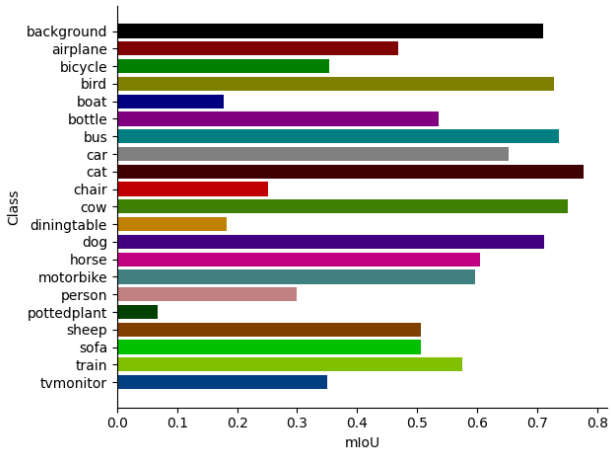

Figure 12. **Per-class mIoUs on PASCAL VOC.** Errors are pronounced in a few particular classes, like "boat", "potted plant", and "dining table," which are primarily due to localization issues with CLIP.

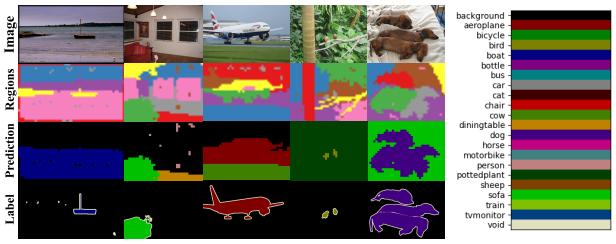

Figure 13. **Examples of segmentation failures.**From the regions we see that most object are correctly segmented and classified, but CLIP fails on the background. From left to right, water is classified as "boat," hardwood floor is classified as "dining table," runway grass is classified as "cow," forest foliage is classified as "potted plant," and bedding is classified as "sofa." This persists across threshold values, as the CLIP similarities are very high. Refining CLIP's localization ability can close much of the gap to oracle decoding.

scores in Figure 12, and examples of CLIP's failure to localize in Figure 13. Crucially, it appears that CLIP does a poor job localizing particular classes, associating "boat" to any water or beach in the image, "potted plant" and "cow" to ground cover, and "person" to all sorts of human-built objects. Fixing these localization errors in CLIP is out of the scope of our contributions, but could yield improvements to match segmentation-specific methods.

D.2. Unsupervised Instance Segmentation

As an additional proof-of-concept, we run experiments on a more difficult task, unsupervised instance segmentation, which requires simultaneously generating object proposals and segmenting salient objects. To benchmark our method, we use the standard COCO 2017 [40] validation split, and follow prior work [70] to report results on both instance segmentation and object detection metrics in a class-agnostic setting. Due to the difficulty of generating instance proposals in a diverse image distribution, recent attempts [69, 71] design heuristic decoding strategies based on the structure of a particular model's features, *e.g.,* the final layer of DINO [8], in order to generate region proposals.

However, we hypothesize that, if the features are informative enough, a simple clustering strategy and generic scoring function should suffice for high-quality instance segmentation. In our implementation, we use K-Means to generate region proposals, and silhouette scores to rank those proposals.

We start by generating initial region proposals by clustering with K-Means on top of the dense features we extract, with K ranging from 2 to 10. To further expand our pool of proposals, we use agglomerative clustering to hierarchically merge spatially adjacent regions with ward linkage.

Naively, we can treat each instance proposal as a binary clustering problem with the foreground and background each as their own cluster, and directly use silhouette scores to rank proposals. However, instances usually take up a relatively small portion of an image making the binary clustering extremely imbalanced, which significantly harms the scores and ranking.

To this end, instead of treating the complement of foreground masks as background, we subsample the background pixels to create a balanced subset by only preserving the background pixels that are close to the foreground pixels in feature space. We also adopt standard post-processing steps to remove duplicate and extreme-sized segments before producing the final output. Finally, since the silhouette score is in the range $[-1, 1]$, we can use 0 as a threshold to remove low-quality proposals.

We follow the above procedure on top of the features produced by optimizing Eqn. 4 over Stable Diffusion's attention layers. We report our results and compare to the current state-of-the-art region proposal methods in Table 8.

Due to the approximation error in binarizing the affinity matrix for clustering, both TokenCut and MaskCut have trouble yielding diversified samples. By contrast, our learned features contain richer information that allows us to adopt a generic instance grouping pipeline without any post-processing on the features. As we see in Table 8, this leads us to generate high-quality diversified proposals with better recall in both instance segmentation and object detection metrics, while maintaining comparable precision to prior methods. Qualitative results are available in Figure 14.

E. Code Sources

All experiments are implemented in Python with PyTorch [50]. For Stable Diffusion [53], we use Hugging-

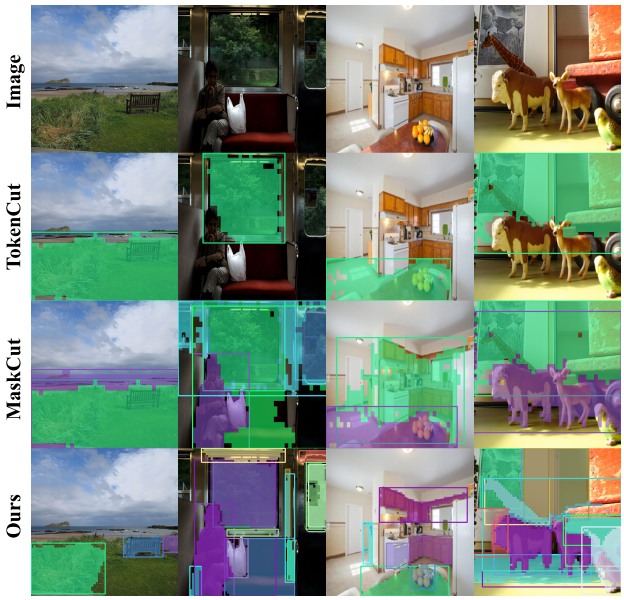

Figure 14. **Examples of different methods on instance segmentation.** As described by the original authors, TokenCut [71] can only generate a single object proposal, and MaskCut [70] is limited as well. Our method shows better localization results and scales to many instances.

Method	#Masks	AP^{box}_{50}	AP^{box}	AR^{box}_{100}	AP^{mask}_{50}	AP_{mask}	AR^{mask}_{100}
TokenCut [71]	1	5.2	2.6	5.0	4.9	2.0	4.4
TokenCut [71]	3	4.7	1.7	8.1	3.6	1.2	6.9
MaskCut [70]	3	6.0	2.9	8.1	4.9	2.2	6.9
Ours	13	4.0	1.9	**11.2**	4.0	1.5	**8.2**

Table 8. **Results of instance segmentation on COCO-val-2017 [40].** Our learned pixel-wise features, with a simple and generic instance segmentation decoding pipeline, significantly outperform baselines in recall, in both object detection and instance segmentation. On the other hand, despite generating many more proposals per image, our method still maintains comparable precision.

Face Diffusers [68]. For baselines, we use official numbers, implementations, and model weights, except in the case of MaskCLIP [77], where we reimplement the method due to difficulty in obtaining satisfactory performance.