Supplementary Material

1. Ablation on Denoising and Caption Filtering

This section aims to provide more insight into the effects of denoising and caption filtering on the final performance. Additionally, we analyze the effects of each enhancement step by visualizing the influence on the vision-language embeddings.

1.1. Effects on Performance

Our enhancement of the original BLIP embeddings involves iterative semantic clustering followed by denoising using a Conditional Random Field (CRF) and majority filter. As shown in Table A2, performing denoising after clustering significantly improves performance, increasing mIoU from 70.4 to 87.1 on PASCAL VOC [1], confirming the effectiveness of this denoising step. Additionally, we analyze the impact of caption filtering by evaluating mIoU performance with this step disabled. The results in Table A2 demonstrate that caption filtering is essential, regardless of the enhancements applied to the original BLIP embeddings. This emphasizes the importance of selecting relevant words from the decoded text.

1.2. Visualization of Embeddings in BBoost

In Fig. 5, we visualize intermediate 32x32 outputs of BBoost to better understand what each individual step of clustering, alignment and denoising achieves. Firstly, Fig. 5 reveals that clustering BLIP embeddings results grouping pixels into semantic areas. With $k = 2$, the image is divided an area which includes the cows (in blue) and another for everything else (green), while for $k = 8$, various other objects - such as the pole in purple - get included. At this stage, the embeddings appear noisy and inconsistent across all k, which is why the combination of all clusters is so important. The alignment output shows that certain objects from a higher k embeddings are now included in those of lower k embeddings, *i.e.* the aligned embedding captures a more extensive list of objects present in the image. Finally, to further reduce the noise in the embedding, we apply CRF and a majority filter. We see that the final embedding consists of three unique semantic areas, which are eventually fed to the text decoder for text generation.

2. Generated/Fixed Vocabulary Similarity

To investigate how similar vocabularies generated with AutoSeg are to the fixed ones, we measure how many of the fixed classes are directly included (with identical wording) in the generated vocabulary. Additionally, we measure the average CLIP cosine similarity between the CLIP embedding of each fixed class to the CLIP embedding of the most similar generated class. Table A1 shows the results. We find

Table A1. **Similarity between generated and fixed vocabularies.** Generated vocabularies include classes that sufficiently overlap with annotated classes.

Dataset	Number of Classes	Generated	% included	Avg. CLIP-sim
VOC	20	938	70.0%	85.3%
PC	459	669	37.7%	91.5%
ADE	847	913	41.3%	92.0%
Cityscapes	20	380	75.0%	85.7%

Table A2. **BBoost Denoising and Caption Filtering Ablation.** Using both clustering and denoising for feature enhancement results in the highest mIoU performance on PASCAL VOC [1], as well as the most compact vocabulary. We use 1 captioning cycle for simplicity.

Enhancement Type	Caption Filtering	Vocabulary Size	mIoU
Clustering	Disabled	4185	3.8
Clustering	Enabled	1273	70.4
Clustering + Denoising	Disabled	4026	3.8
Clustering + Denoising	Enabled	938	**87.1**

Table A3. **Ablation on Mappers.** Compared to other approaches for mapping, LAVE provides more reliable mappings between auto-vocabulary and target vocabulary classes for comparative evaluation that uses IoU. Tested on PASCAL VOC [1].

Mapper	mIoU [1]
Sentence-BERT [4]	73.9
CLIP [3]	77.5
LAVE (Ours)	**87.1**

that there is a higher direct overlap with smaller vocabularies, which can potentially be explained by the specificity of classes in datasets with larger fixed vocabularies. However, when reflecting on the average CLIP cosine similarity, it becomes clear that the generated and fixed vocabulary share high similarity across all datasets. Overall, we conclude that the coverage of the annotated classes of the four datasets is sufficient across the generated vocabularies. Similarly, the generated classes are related to the annotated ones to a great extent.

3. LLM-based Auto-Vocabulary Evaluator

This section provides more insights into the comparison of our LLM-based Auto-Vocabulary Evaluator (LAVE) with other mappers, as well as the pseudocode and text prompts used for each respective dataset. LAVE is key to enabling the evaluation of predicted auto-vocabulary classes on public benchmarks such as PASCAL VOC [1], as it allows us to compare with other auto-vocabulary or open-vocabulary methods which predict a fixed-vocabulary for evaluation.

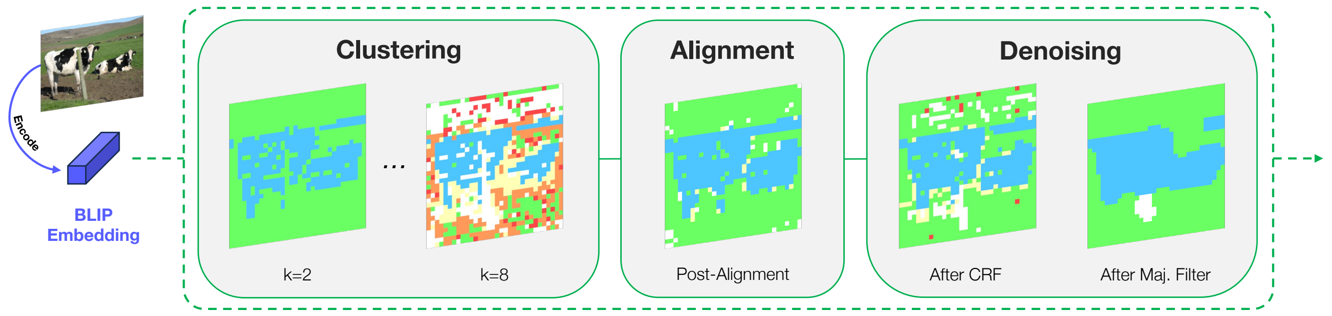


Figure 2. **Visualization of Intermediate BBoost Outputs.** Intermediate BBoost embeddings, visualized as 32x32 grids, show how vision-language features are gradually enhanced into semantically regions that are more coherent, which proved more effective for text generation that preserves locality. After the last step, the embeddings are decoded into text by the text decoder.

3.1. Mapping Comparison

As detailed in the main paper, we initially experimented with Sentence-BERT [4] and CLIP [3] to map auto-vocabulary classes to the fixed annotated vocabulary of the respective dataset. In this approach, we compute the text encoding for each class in the auto-vocabulary as well as the target vocabulary, and select the target class with the highest cosine similarity. However, this strategy proved ineffective, as the mapping often failed for auto-vocabulary classes with clear target matches—for example, mapping *taxi* to *road* instead of *car*. To quantitatively evaluate this issue, we compared the mIoU performance achieved using these mappings against that of LAVE. Intuitively, poorer mappings should lead to lower pixel prediction accuracy and therefore reduced mIoU. Table A3 demonstrates that, using the same auto-vocabulary, LAVE achieves an mIoU that is 13.2 points higher than Sentence-BERT and 9.6 points higher than CLIP. These results highlight LAVE's effectiveness in accurately identifying the correct target class, enabling a more reliable evaluation of AVS methods.

3.2. LLM Mapper Pseudocode

The LLM Mapper is tasked with creating a mapping dictionary which maps from auto-vocabulary classes to fixed-vocabulary classes. The resulting mapping dictionary can be used during inference to map pixel-level class predictions belonging to an automatically generated out-of-vocabulary class to fixed-vocabulary class indices. In Alg. 1, we detail the procedure to create the mapper.

3.3. LLM Mapping Prompts

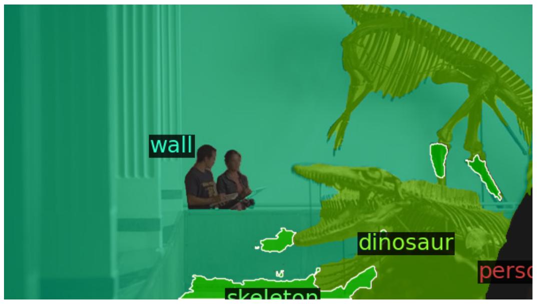
The **generateDialogs()** function in the LLM Mapper takes a prompt template as one of its inputs. This prompt template is dependent on the dataset and is used to generate dialogs to query the LLM with. For PASCAL VOC and Cityscapes, we explicitly specify the list of categories given its short length with 20 classes each. For PASCAL Context and ADE20K, with its 459 and 849 classes respectively, we rely on the LLM's knowledge of the datasets. Explicitly naming each object class from these datasets causes the LLM to run into memory issues when queried with long dialogues. We specify the prompt templates for each respective dataset as follows:

PASCAL VOC and Cityscapes
"To which class in the list `<dataset>` is `<name>` exclusively most similar to? If `<name>` is not similar to any class in the list or if the term describes stuff instead of things, answer with `background`. Reply in single quotation marks with the class name that is part of the list `<dataset>` and do not link it to any other class name which is not part of the given list or `background`." where `<noun>` is the noun to map to the vocabulary and `<dataset>` is the list of classes of PASCAL VOC or Cityscapes.

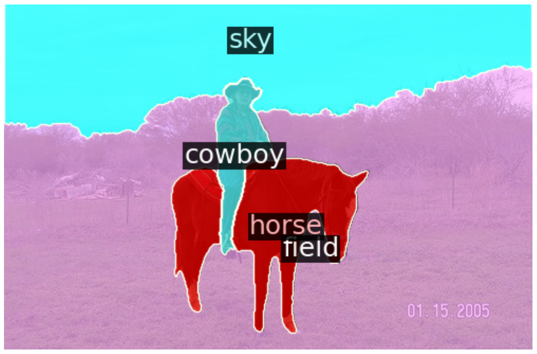
PASCAL Context and ADE20K
"To which class in the `<dataset>` dataset is `<name>` exclusively most similar to? If `<name>` is not similar to any class in `<dataset>` or if the term describes stuff instead of things, answer with `background`. Reply in single quotation marks with the class name that is part of `<dataset>` and do not link it to any other class name which is not part of the given list or `background`." where `<noun>` is the noun to map to the vocabulary and `<dataset>` is either 'PASCAL-Context 459' or 'ADE20K'.

4. Additional Qualitative Results

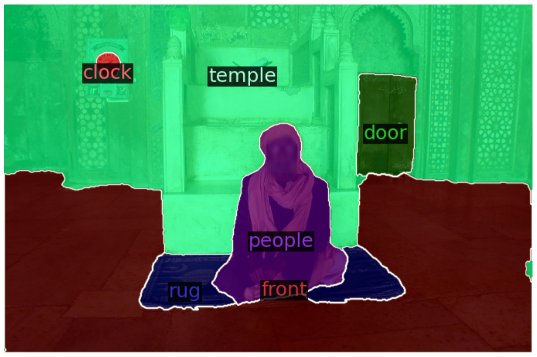
In Fig. 3 and 4, we provide additional qualitative results of AutoSeg and discuss observed capabilities or limitations in the caption of each figure. Our main insights are that 1) AutoSeg effectively labels unseen categories; 2) AutoSeg uses semantically more precise classes for classification; 3) generated classes that are redundant could end up being segmented.



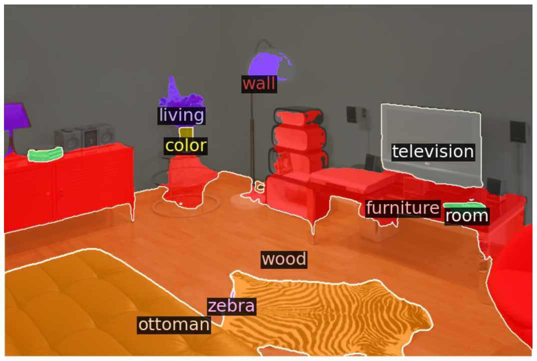
(a) **Success Case.** The rare and out-of-vocabulary class name *dinosaur* is generated automatically and segmented.



(b) **Success Case.** AutoSeg is able to accurately classify and segment classes not present in the dataset, such as the *cowboy*.

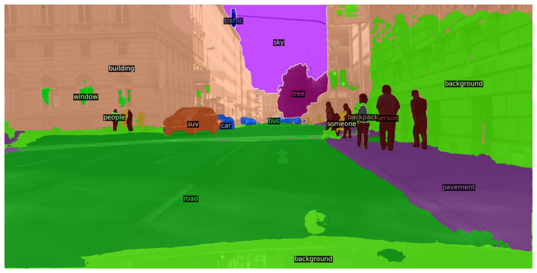


(c) **Failure Case.** AutoSeg occasionally struggles with indoor scenes from specific buildings; for example, in this case, a *mosque* is mistakenly identified as a temple.

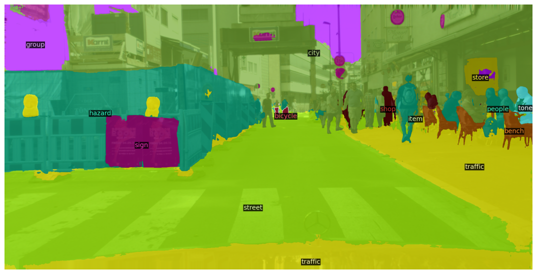


(d) **Failure Case.** A *rug* is mistaken for a *zebra*, potentially due to *rug* not being generated as a class on itself.

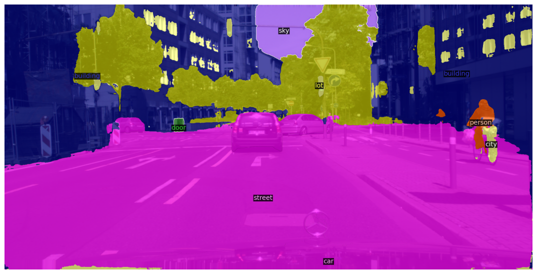
Figure 3. **Additional Qualitative Results**. We discuss a selection of success and failure cases on PASCAL and ADE20K above.



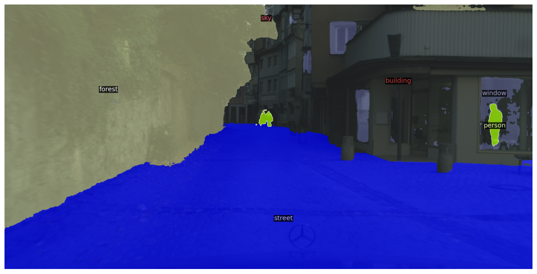
(a) **Success Case.** While annotations in Cityscapes consider all car types to be equal, AutoSeg often generates more specific descriptions, such as the *SUV* in this image.



(b) **Success Case.** AutoSeg is able to provide class names which can be crucial for real-life scenarios, such as *hazard* in this example. The standard 20 classes in Cityscapes do not account for such descriptions.



(c) **Failure Case.** Occasionally AutoSeg generates general class names, such as *street*, which dominate the search space of the segmentor and cause it to attend more on that class. This may result in oversegmentation and other class names, such as *car*, to be ignored. Such general classes, however, could be filtered out.



(d) **Failure Case.** A wall with vegetation is mistaken for a forest, likely due to overclustering in the clustering process.

Figure 4. **Additional Qualitative Results** We discuss a selection of success cases and failure cases on Cityscapes above.

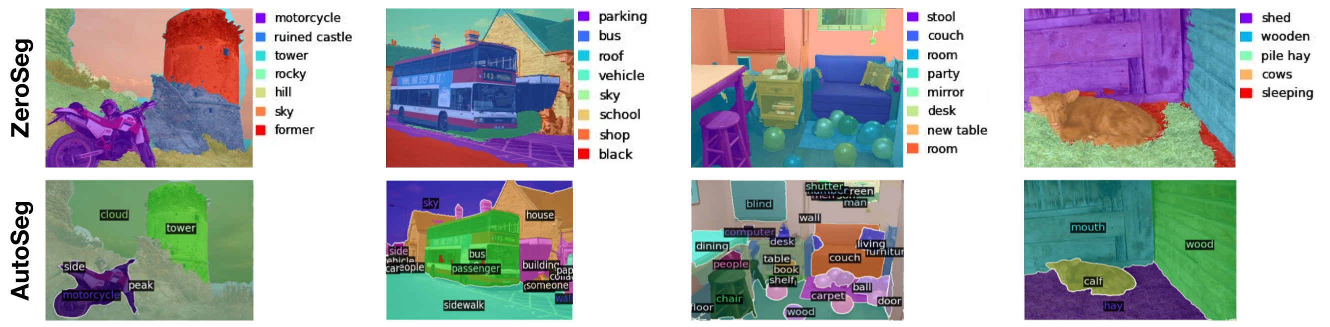


Figure 5. **Qualitative Comparisons.** We provide qualitative comparisons between AutoSeg (bottom row) with ZeroSeg (top row). Our main take-away is that AutoSeg has sharper object boundaries and often generates target classes that are closer to the object's actual label. However, ZeroSeg introduces interesting contextual semantics, such as *former*, *party* or *sleeping*. Depending on the use case, this might be valuable information to predict.

Algorithm 1 LLM Mapper

```
Require: nouns, dataset_vocabulary, llm_batch_size,
    prompt_template, LLM

  all_responses, map_dict, skipped ← initialize()
  for batch in split(nouns, llm_batch_size) do
      dialogs ← generateDialogs(batch, prompt_template)
      batch_responses ← LLM(dialogs)
      store(all_responses, (batch_responses, batch))
  end for

  for (batch_response, batch) in all_responses do
      for (response, noun) in (batch_response, batch) do
          answer ← parseResponse(response)
          common ← intersect(answer, dataset_vocabulary)
          updateDict(map_dict, noun, common)
          updateSkipped(skipped, noun, common)
      end for
  end for

  for key in map_dict do
      if key is in vocabulary then
          map_dict[key] ← key
      end if
      if key is in skipped then
          skipped ← removeItem(skipped, key)
      end if
  end for

  for skipped_key in skipped do
      map_dict[skipped_key] ← "background"
  end for

  return map_dict
```