Reproduction of "Self-Supervised Learning of Object Parts for Semantic Segmentation"

Anushka Kore (5910935)[1], Lotte Kremer (4861957)[2],
Pepijn de Kruijff (4962966)[3], Ruben Eland (4868463)[4]

Delft University of Technology, Delft, The Netherlands
[1]A.A.Kore@student.tudelft.nl, [2]L.A.A.Kremer@student.tudelft.nl,
[3]P.G.M.dekruijff@student.tudelft.nl, [4]r.a.eland@student.tudelft.nl

Reproduction code & Blog PDF: https://github.com/reland27/leopart_reproduced
Dataset: https://www.dropbox.com/s/6gd4x0i9ewasymb/voc_data.zip?dl=0
Original paper: https://arxiv.org/pdf/2204.13101.pdf
Code for original paper: https://github.com/MkuuWaUjinga/leopart

1 Introduction

The variedness of object definitions poses a challenge for self-supervised or unsupervised semantic segmentation, hindering the ability to scale beyond object-centric images. This makes unsupervised segmentation difficult due to the lack of principled object definition during training as pointed out by Ziegler and Asano [2]. In the paper, called "Self-Supervised Learning of Object Parts for Semantic Segmentation", Ziegler and Asano propose a novel approach to address this by introducing a dense clustering pretext task to learn semantically rich spatial tokens, focusing on object part learning rather than object priors. They explore the use of Vision Transformers (ViTs) with a new loss function to train self-supervised dense ViT models for unsupervised segmentation.

Ziegler and Asano published multiple results of their model on both fine-tuning loss and fully unsupervised semantic segmentation. In this blog, we will discuss our reproduction of the findings of this paper on the latter. More specifically, our goal in this reproduction project is to reproduce the findings listed in Table 5 of the paper (as shown in Figure 1), which illustrates the contribution by each individual component used in the unsupervised semantic segmentation when applied to the segmentation for the PVOC dataset pre-trained with DINO. Further, we will explain more about the exact method used in the paper for the unsupervised semantic segmentation and our specific reproduction of the results.

	mIoU
K=150	48.8
DINO	4.6
+ Leopart	18.9 (+14.3%)
+ CBFE	36.6 (+17.7%)
+ CD	41.7 (+5.1%)

Table 5. **Component contributions.** We show the gains that each individual component brings for PVOC segmentation and K=21.

Figure 1: Table 5 of the paper showing component contributions [2].

2 Overview

2.1 Outline

The paper aims to design a pretext task to ensure representations of image patches containing the same part of an object are relatively similar. This is done using a clustering pretext task. Two crops of the same image are soft-assigned to clusters, corresponding to object parts. As the crops both contain the same object or a part of this object, the difference between these cluster assignments should be minimal. This is visualized in Figure 2.1. Minimising the difference between these cluster assignments through gradient descent ensures object parts have similar representations for improved object recognition. Additionally, techniques such as Cluster-based Foreground Extraction (CBFE) and Overclustering with Community Detection (CD) are integrated to enhance foreground object extraction and group object parts into coherent communities, respectively, contributing to more robust object categorization and recognition across datasets.

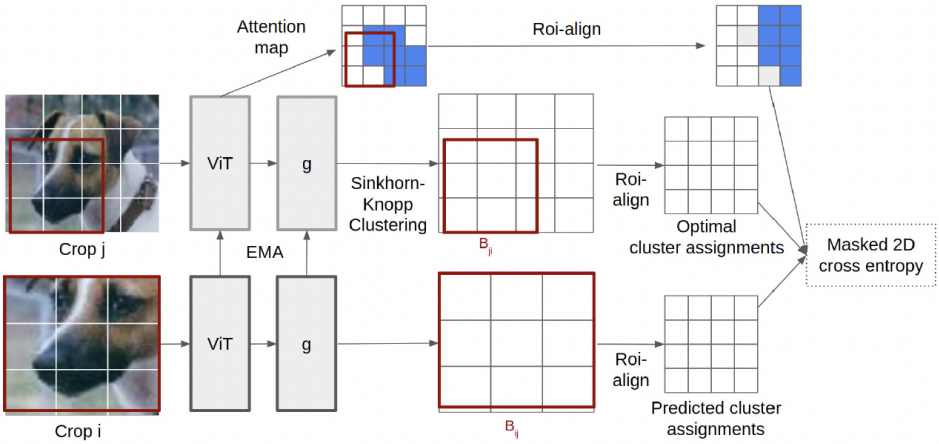


Figure 2: Leopart training process

2.2 Leopart

The input images from the dataset are first cropped according to the multi-crop augmentation strategy first introduced with SwAV. This cropping method transforms input images into either a global or local crop. The global crop is a larger crop than the local crop, which represents only a small part of the image. These patches are subsequently passed through a Vision Transformer, resulting in a vector of spatial tokens. These are the representations that should ultimately be close together for the same object parts. This is evaluated in the experiments. These spatial tokens are subsequently passed through an MLP projection head.

The embeddings of each crop are soft assigned to a cluster, representing different object parts. Similarly to SwAV, the spatial tokens are assigned such that the clusters have equal probability mass assigned to them. This prevents collapse, as distance can be minimized by assigning all tokens to the same cluster. The assignment of spatial tokens given this constraint can be implemented using the Sinkhorn-Knopp algorithm.

The difference between the cluster assignments for the spatial tokens from the global crop and local crop should be minimal, as they represent the same object parts. The loss is defined as the sum of cross-entropy loss between the soft assignments of the clusters of the local view and those of the global view for all pairs of crops. The spatial tokens of the global view, because they are constructed based on a larger part of the image, provide a target that the clustering prediction for the local view should be close to. The formal loss is defined as follows:

$L(x_{t_1}, \ldots, x_{t_V}) = \sum_{j=0}^{v_g} \sum_{i=0}^{V} [l(x_{t_i}, x_{t_j})]$

This equation sums the pairwise loss between the pairs of global and local crops. The pairwise loss is defined as:

$$l(x_{t_i}, x_{t_j}) = H\left(\left[s_\tau\left(\alpha_{B_j,i}(g(\Phi(x_{t_i}))^T\right), \alpha_{B_j,i}(Q_{t_j})\right]\right)$$

where H is the cross entropy, S_τ softmax and α_B alignment operator. The alignment operator aligns previously flattened spatial tokens of one crop with the other at the intersection of these crops. This ensures the correct spatial token cluster assignments are compared. In the paper, the RoI-align method is used.

2.3 CBFE

Cluster-based Foreground Extraction or CBFE is a technique used to extract foreground objects from images based on clustering of learned embeddings. In the context of this paper, CBFE plays a crucial role in identifying and segmenting foreground objects without the need for explicit supervision.
The method assumes that clusters in the learned embedding space correspond to object parts, allowing for the extraction of foreground objects by assigning each cluster ID to either foreground object (fg) or background (bg). ViT's merged attention maps are used as a noisy foreground hint to guide the extraction process. These attention maps provide a rough indication of regions of interest in the image.
By processing the attention maps to focus on foreground regions, a binary mask is created by averaging the attention heads, applying Gaussian filtering, and setting a threshold to classify a cluster as foreground. Then by using the training data, clusters are ranked by pixel-wise precision with the binary mask, and a threshold is determined to classify a cluster as foreground or background. The obtained classification is then applied to the patch-level clustering to generate a foreground mask, which helps in segmenting and highlighting foreground objects in the image.
By utilizing CBFE, the paper demonstrates improved performance in fully unsupervised semantic segmentation tasks, showcasing the effectiveness of this technique in extracting foreground objects and enhancing the overall segmentation quality.

2.4 CD

Overclustering with community detection (CD) was utilised to enhance the segmentation results by grouping clusters that represent object parts into coherent communities. This technique leverages network science principles to identify clusters that frequently co-occur together, thereby aiding in the discovery of object categories without the need for additional supervision.
Initially, clusters were overclustered based on the assumption that clusters correspond to object parts. Higher clustering granularities were explored to improve segmentation results, drawing on the idea that grouping clusters can lead to better object categorization.
Next, an undirected and weighted co-occurrence network was constructed, where nodes represent clusters, and edges indicate the likelihood of co-occurrence between clusters based on their presence in images. The conditional co-occurrence probability between clusters i and j was calculated over all images in the dataset. This probability reflected how likely it is for two clusters to appear together in an image.
The Infomap algorithm was then utilized for community detection in the constructed network. This algorithm identifies communities of clusters that frequently co-occur, helping to group related object parts into coherent categories.
Through the integration of CD, improved segmentation performance was achieved by grouping clusters that were semantically related, leading to more accurate and coherent object organization.

2.5 Experiments

The paper uses DINO to initialise the Vision Transformer weights. The model is then trained for 50 epochs with a batch size of 50.

For evaluation, the Vision Transformers' spatial tokens are evaluated. The MLP projection head used during training is discarded. The model is evaluated by using a linear classifier or over-clustering. The over-clustering evaluation method works by running K-means on the spatial tokens. A method called pixel-wise precision is used to match these clusters to the ground truth. This method involves finding the best match for each cluster by considering how many pixels of a cluster correctly belong to a class. The Hungarian matching algorithm is used to solve the assignment of clusters to ground truths in a way that maximizes the accuracy.

The performance is measured as mean Intersection over Union (mIoU). It is calculated as the area of overlap between the predicted segmentation and the ground truth segmentation, divided by the area of the union between these two. This is averaged over all classes in the dataset.

3 Dataset Description

Pascal Visual Object Classes (PVOC) is a dataset originally released in 2005, but is has had regular updates since. It provides standardised image data sets for class recognition and segmentation. Each image has pixel-level segmentation annotations and class annotations.

The data repository is downloaded from the GitHub page of the paper. It consists of 2913 images in JPG format. The repository consists of a folder with the images, a folder with the segmentation classes, containing PNG masks where each class is coloured with a different colour, and a segmentationClassAug folder, containing the black and white outlines of classes. A final folder called provides the class labels.

4 Methods

4.1 Setting up the environment

The environment provided in GitHub [1] has several dependencies on packages that are only available in Linux. Therefore, when we tried to build the environment in Windows, we got errors that conda could not find these Linux packages. As none of the group members ever programmed before in Linux, we decided to set up the environment through the Windows application 'Windows Subsystem for Linux' (WSL). This application runs a Linux kernel inside a Virtual Machine and we could build the environment in WSL without errors. We then used Visual Studio Code (VS Code) on Windows to edit the code given in GitHub and run it in the WSL. We ran on the HP ZBook G9 Mobile Workstation and used its NVIDIA RTX A1000 GPU.

4.2 Understanding the code

In the repository, the complete environment is given for all of the different experiments in the paper. The README file explains how these Python scripts can be used and provides links for downloading the data. In the case of unsupervised semantic segmentation, the experiments were done with the PVOC dataset, as mentioned before. To get the correct outputs, it is also important to use the checkpoint for the trained model and masks provided in the README. The setup for those files is well explained in the file.

Since the README does not provide specific instructions for Table 5, we have added the exact lines and configurations to run below. To reproduce the first two lines of Table 5 in the paper, the DINO and Leopart scores, the file *sup_overcluster.py* has to be run with the correct configurations. The CBFE and CD scores can be reproduced by running *fully_unsup_seg.py* with the correct configurations. See Table 1 for an overview of what to run in order to reproduce the scores from Table 5. Make sure to provide the right paths for the checkpoints (our results are based on the vits16 checkpoint), masks (the pre-trained attention maps from DINO used as ground truth for the foreground clustering) and data (PVOC dataset), in order to overcome errors. The number of clusters is $K = 21$.

Python file	Configuration	Method
sup_overcluster.py	–config_path experiments/overcluster/configs/pascal/k=21/dino.yml	DINO
sup_overcluster.py	–config_path experiments/overcluster/configs/pascal/k=21/leopart-vits16.yml	+ Leopart
fully_unsup_seg.py	–ckpt_path {vit-small-ckpt} –experiment_name vits16 –best_k 149 –best_mt 2 –best_wt 0.09 –evaluate_cbfe True	+ CBFE (+CD)
fully_unsup_seg.py	–ckpt_path {vit-small-ckpt} –experiment_name vits16 –best_k 149 –best_mt 2 –best_wt 0.09	+ CD

Table 1: Files and configurations to reproduce Table 5

The 'vits' in the configurations is an abbreviation of 'Vision Transformer - small'. The GitHub provides several pre-trained checkpoints with vitb8 (ViT - base, patch size = 8) and vits16 (ViT - small, patch size = 16). The vitb8 checkpoints are more complex: they have 85M params versus 21M for the vits16. As we already ran into memory issues with the vits16, we decided to do all reproducing with this checkpoint and we did not look into the more complex model vitb8. You can find more information on the memory errors (and how to solve them) in the Appendix.

The masks that are provided are pre-trained attention masks from DINO. They are used as the ground truth in both training and evaluation with the help of the computed mIoU score, which is further explained in the results section. To get more insight into the data and the masks we have provided the following figures showing the true image, the downloaded mask and the combination of image and mask in Figure 4, 3, 6 and 5.

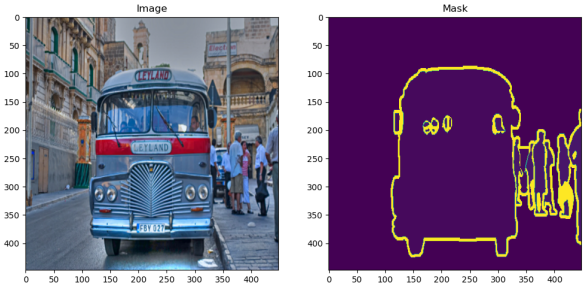


Figure 3: Example 1 - Image next to its mask

Figure 4: Example 1 - Image and mask in one figure

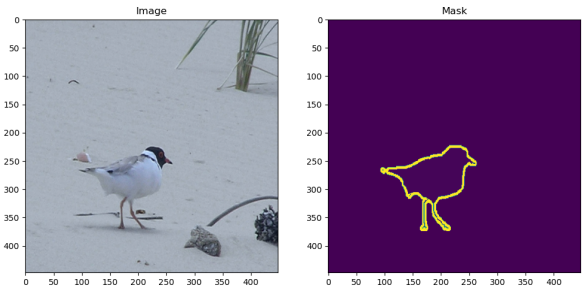


Figure 5: Example 2 - Image next to its mask

Figure 6: Example 2 - Image and mask in one figure

4.3 Troubleshooting/Changing the code

We will highlight one small mistake in the code here that everybody will encounter. Read the appendix for additional errors you may encounter when you use a similar setup to ours: running locally with WSL and VS code. Moreover, we made a few changes in the code to be able to plot the images and masks while running.

Errors in sup_overcluster.py

When running *sup_overcluster.py* with the provided configurations, the following error occurred:

File "/home/reland/leopart-main/experiments/utils.py", line 422, in get_backbone_weights url, loader, weight_transform = arch_to_args[f'archpatch_size-method"] KeyError: 'vit16-ours'

After some research, we found out the function *arch_to_args* from *utils.py* requests inputs 'vit-base16-ours', 'vit-base8-ours' or 'vit-small16-ours'. In our case, the error can thus be solved by changing the arch in the configuration files from 'vit' to: 'vit-small'.

Plotting images and masks

We updated *fully_unsup_seg.py* to be able to plot the input images and their ground truth masks while running. The downloaded masks are DINO ViT-S/16 attention maps, used as the true foreground clustering of the image. The plotting is implemented in the function *start_unsup_seg*. The images and masks can be created as separate figures next to each other, or one figure can be plotted where the mask overlays its image. See Appendix 6 for the code. Being on a time limit and focusing on the numeric results, we have not implemented a plot of the created masks of the different components. A great overview of this can be seen in Figure 3 of the paper [2].

5 Results

We ran each method 20 times to get a representative score while preventing excessive run times. See Table 2 for the mean of the obtained scores. For more information about which lines were run, look again to Table 1. The numbers represent the mIoU, which stands for the mean Intersection over Union. It represents the overlap between the trained masks and given (ground-truth) masks. To highlight the specific improvement by adding respectively Leopart, CBFE and CD, the improvement is in between brackets.

	mIoU
DINO	4.5
+ Leopart	18.0 (+ 13.5%)
+ CBFE	35.6 (+ 17.6%)
+ CD	35.2 (- 0.4%)

Table 2: Reproduction of Table 5

5.1 Comparison

When comparing those results with the original results in Table 1, it can clearly be seen that the fourth line is the only one that seems far out of range. For the first three lines, the mIoU differs at most 1 %

from the original scores. The fourth line which adds the CD, shows a small decrease in performance in our reproduction. In the original paper, the CD increased 5.1 %. Although the small differences in the first three lines could be due to different settings, for example, the number of samples used to compute the score, the fourth line seems too far off. We believe it could have something to do with the use of Infomap. In the most recent versions of both Anaconda and Miniconda, our machine was not able to recognize the Infomap package. Although we were able to solve this error by copying a specific version of *libstdc++.so.6* in the virtual environment settings, it could be that Infomap still does not work as intended. Read more about the Infomap import error in the Appendix.

We did not reproduce the mIoU for the supervised clustering with $K = 150$ which was only used as an upper bound indication in Table 5, since our goal was to focus on the unsupervised results.

6 Conclusion

In general, it can be concluded that Table 5 in the paper [2] can be reproduced with only a few small changes in the repository given. However, the time necessary to make the repository run on a new machine was significant. The community detection which was added to make the best version of unsupervised semantic clustering, is not working as expected in our reproduction. The improvement shown in the original paper is not reflected in our results in Table 2. Whether this is due to mistakes in the version of Infomap used in the environment or flakes in the algorithm itself, should be investigated more. For new reproduction studies, we also recommend testing with different datasets to see whether the improvement for each part is still as significant as it is for this dataset.

Acknowledgements

We want to express our sincere gratitude to our TA, Alexandru Bobe for his guidance and assistance during the course of this project. Additionally, we extend our thanks to our external supervisor, Sayak Mukherjee for his valuable feedback and input.

Contributions

This project was a collaborative effort where contributions were made by all team members, with some tasks overlapping and each member contributing equally.

Anushka I worked on how to best implement the given code on our system and implement it on the PVOC dataset. By brainstorming and working with Ruben, I successfully configured the environment in WSL, understood the various dependencies, troubleshooted the many errors encountered along the way and finally got the code running on Visual Studio. I also contributed to the writing of this blog.

Lotte I mainly worked on understanding the code after we were able to run the basics of the repository in WSL. This also included some troubleshooting and fixing the import error with Infomap. In the last week, the focus was on writing the blog.

Pepijn First I attempted to run the code on Mac OS, which turned out not to be possible. Afterwards, while the other group members were working on WSL, I attempted to get the code to run on the Google Colab environment. When the code ran I investigated which parts of the code to disable to reproduce Table 5. The last week was spent on the blog.

Ruben As Anushka mentioned, we discovered together how to set up the WSL & VS Code and build the environment there. I also found out which files to run in order to reproduce Table 5 and went through the troubleshooting to get the right results. I also implemented the Python script which plots the images & masks. Finally, I helped write the blog.

References

[1] Adrian Ziegler and Yuki M Asano. Leopart repository. https://github.com/MkuuWaUjinga/leopart, 2022. Accessed: 14-03-2024.

[2] Adrian Ziegler and Yuki M Asano. Self-supervised learning of object parts for semantic segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14502–14511, 2022.

Appendix

A1: Code for plotting images and their masks

```
    #Creating plots of images and masks

    imgloader = data_module.val_dataloader() #Initialize dataloader

    #COMMENT OUT IF UNWANTED
    #Plotting images and masks as separate figures
    for batch in imgloader:
        imgs, masks = batch
        imgs = imgs.numpy() #Convert to Numpy for plotting
        masks = masks.numpy()

    for img, mask in zip(imgs, masks):
        img = np.transpose(img, (1, 2, 0)) #Convert image to (H, W, C) format
        mask = mask.squeeze() #Remove unnecessary dimensions from mask

        fig, ax = plt.subplots(1, 2, figsize=(12, 6))

        ax[0].imshow(img)
        ax[0].set_title("Image")

        ax[1].imshow(mask)
        ax[1].set_title("Mask")

        plt.show()

    #COMMENT OUT IF UNWANTED
    #Plotting image & mask in one figure
    for batch in imgloader:
        imgs, masks = batch
        imgs = imgs.numpy() #Convert to Numpy for plotting
        masks = masks.numpy()

    for img, mask in zip(imgs, masks):
         img = np.transpose(img, (1, 2, 0))  #Convert image to (H, W, C) format
        mask = mask.squeeze()  #Remove unnecessary dimensions from mask

        fig, ax = plt.subplots(figsize=(6, 6))

        ax.imshow(img)
        ax.imshow(mask, cmap='jet', alpha=0.5)  #Overlay the mask with lowered
                                                    transparency
        ax.axis('off') #Hide axis ticks

        plt.show()
```

A2: Troubleshooting

Infomap import error The following error was obtained when trying to import Infomap from infomap:

```
(obdet) akore@LAPTOP-RR6ABD63:~/leopart-main$ python experiments/fully_unsup_seg/fully_unsup_seg.py --ckp_path{checkpoints/leopart_vits16.ckpt} --experimant_name vits16 --best_mt 2 --best_wt 0.09
Traceback (most recent call last):
  File "/home/akore/leopart-main/experiments/fully_unsup_seg/fully_unsup_seg.py", line 16, in <module>
    from infomap import Infomap
  File "/home/akore/miniconda3/envs/obdet/lib/python3.9/site-packages/infomap.py", line 38, in <module>
    import _infomap
ImportError: /home/akore/miniconda3/envs/obdet/lib/python3.9/site-packages/_infomap.cpython-39-x86_64-linux-gnu.so: undefined symbol: _ZSt28__throw_bad_array_new_lengthv
(obdet) akore@LAPTOP-RR6ABD63:~/leopart-main$
```

We tried reinstalling a lot of different versions of infomap, but each version kept on giving the same import error. We were really stuck on this error for a long time and it was quite remarkable that we got this error as we ran our code with exactly the same packages and builds as the authors of the paper.

In the end, we were able to resolve the error by manually copying the C++ library specified in the error with the following Linux command:

cp /usr/lib/x86_64-linux-gnu/libstdc++.so.6 /home/

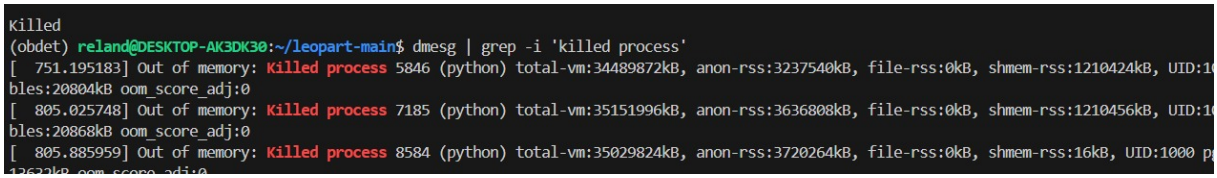
We then moved this library manually into /anaconda3/envs/obdet/lib and overwrote the existing file there. The infomap import error was then resolved.

Memory issues

Throughout this reproduction project, oftentimes processes would stop and the message 'Killed' would be displayed in the Terminal. We found out that with the following command you can find out why the Linux kernel killed the process:

dmesg — grep -i 'killed process'

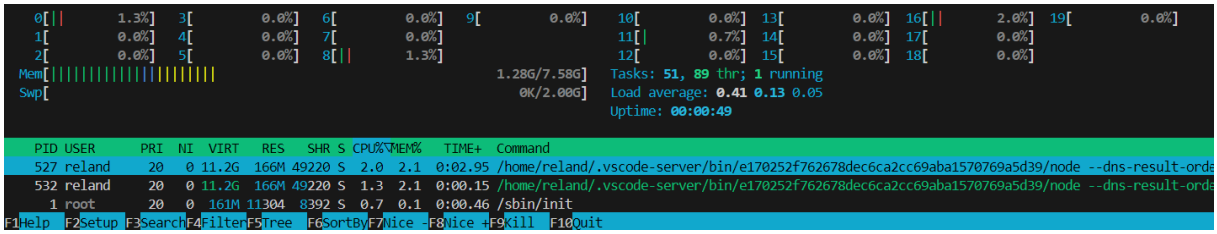
In our case, the response always was a shortage of memory:



After some research, we found out you can accurately keep track of your memory usage with htop and in Linux you can manually enable more memory. By this, we overcame the memory issues. You can install htop with the following command:

sudo apt-get install htop

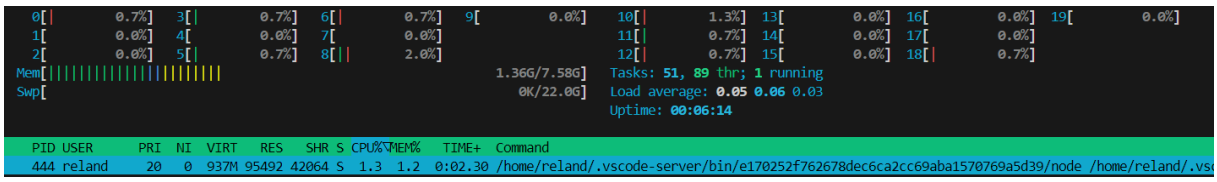
Then running 'htop' gives the following overview:



As you can see, the swap memory by default only is 2GB. We added another 20 GB with the following commands:

- sudo fallocate -l 20G /swapfile #Create a 20GB swap file
- sudo chmod 600 /swapfile #Secure the swap file by restricting access
- sudo mkswap /swapfile #Mark the file as swap space
- sudo swapon /swapfile #Enable the swap

Then running 'htop' again, we now have 22GB swap memory:



Note: 'sudo swapon /swapfile' has to be run again each time after shutting down the WSL. We additionally found out while using htop that decreasing the batch_size and num_workers also decreased the amount of memory required. In the end, the extra memory, running *fully_unsup_seg.py* with batch_size = 5 (was 15) and num_workers = 4 (was 12) and running *sup_overcluster.py* with num_workers = 4 (was 12) completely freed us of the memory issues.