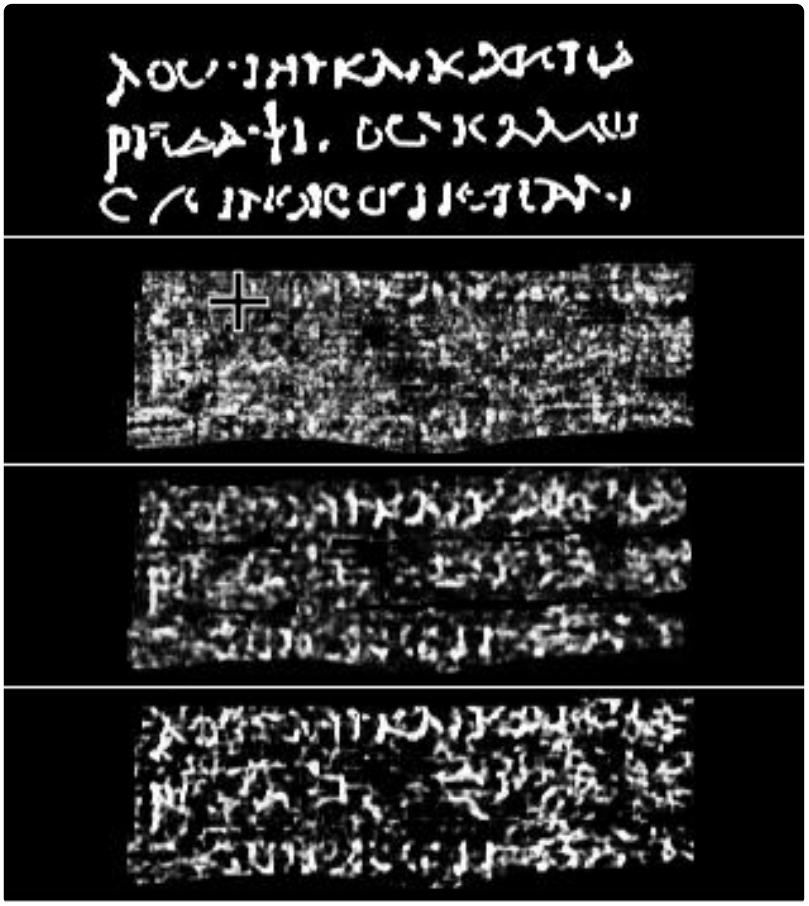
Notes

As a quick summary, what we are tying to do with this method is to:

1. Pretrain on large amounts of unlabeled scroll data (in this case, a whole unlabeled segment for now)
2. Study whether the model has learned good enough features so that, in the best case, we can use them to detect ink without any labels (though this is extremely challenging) or at least be able to detect much better ink by finetuning it to a few annotated labels vs. a non-pretrained model.

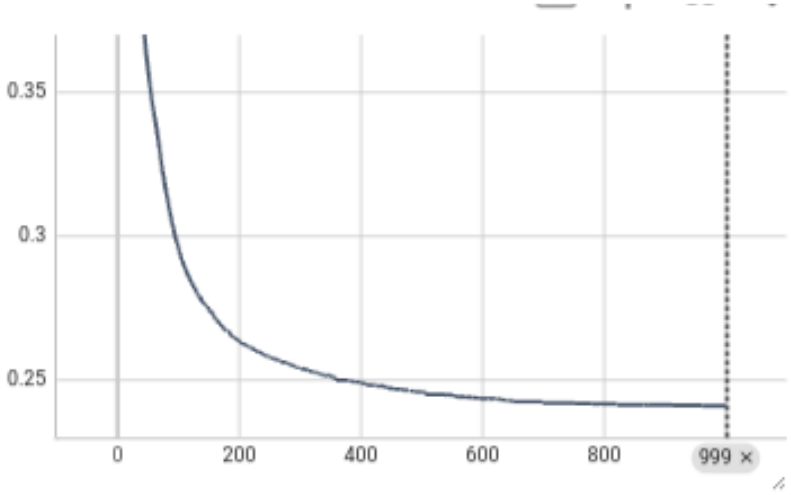
My intention with this document is to show the work I have done until now related to this method. I will not have much time to work on the challenge for a few months, so I thought it was a good idea to share it in case someone finds it useful to improve ink detection.

Previous experiments showed that pretraining has a positive effect both on convergence and results, but were far from desirable. For example, I did the experiments with a UNet with a 2D Resnet-50 backbone on segment `20231210121321` and found the following results:

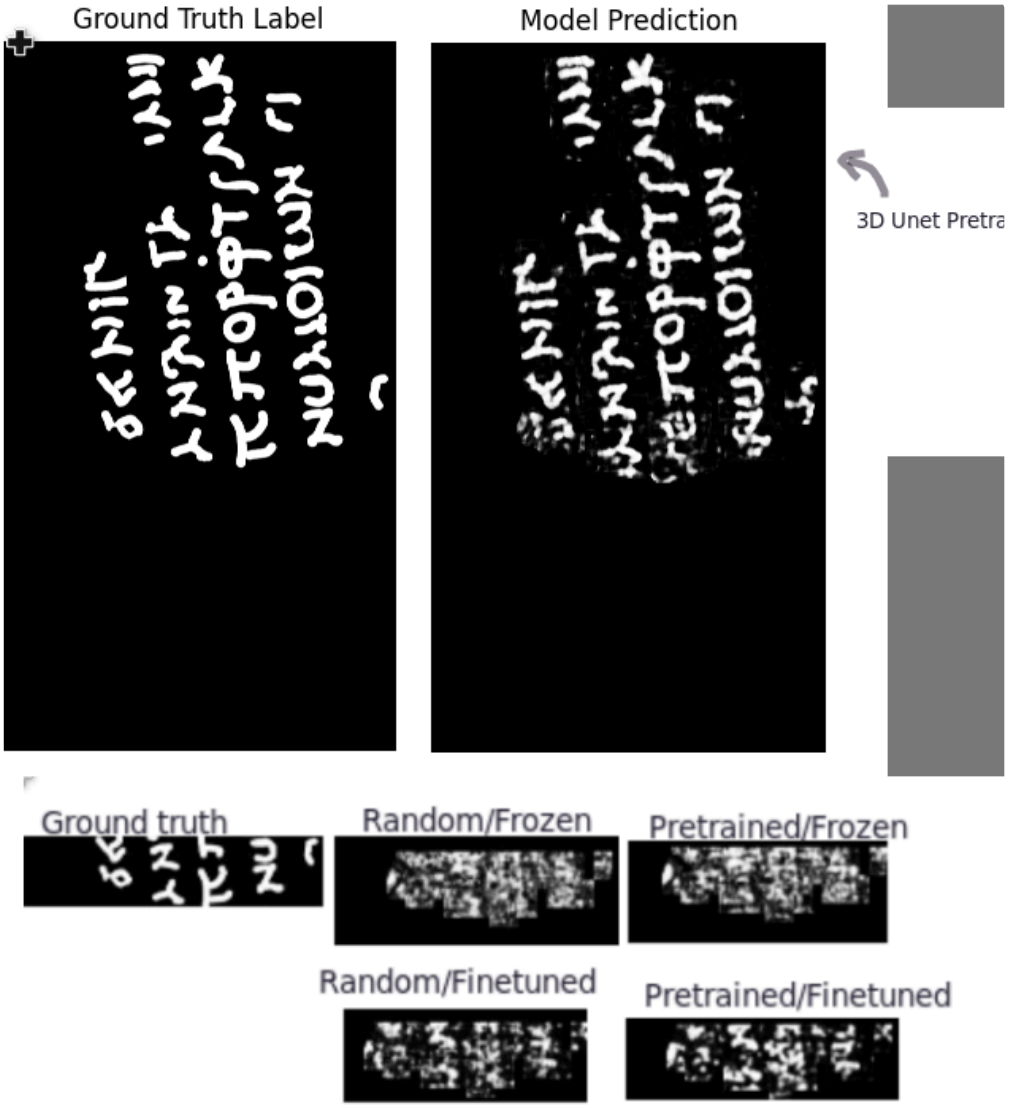


From top to bottom, the images show (i) the ground truth (ii) pretraining, then freezing the encoder and finetuning (ii) vanilla supervised training and (iii) pretraining and not freezing the encoder during finetuning.

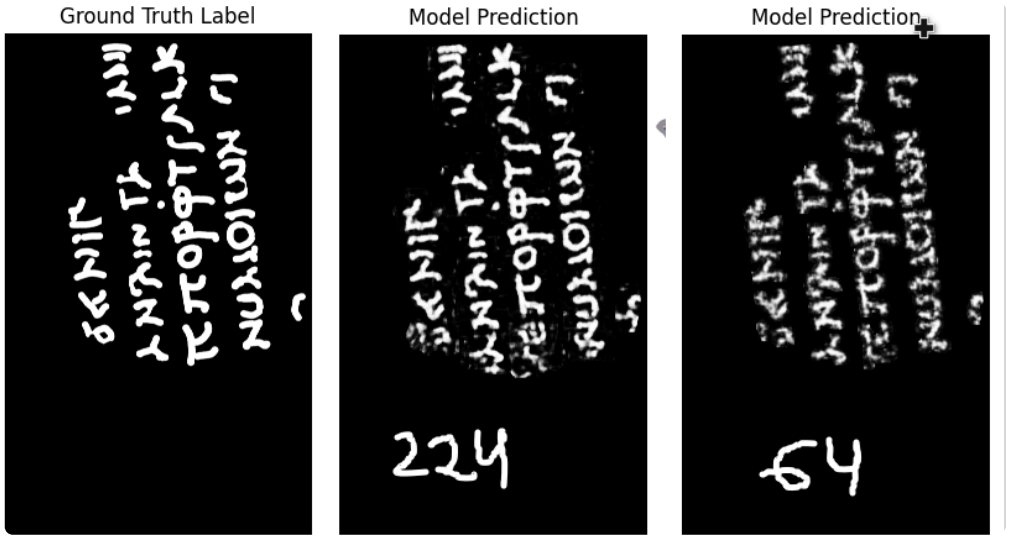
My first guess was that the results were poor because I was using a 2d ResNet as a backbone. Therefore, I implemented SparK to work with 3d resnets. The figure below shows the loss term during pretraining a 3D resnet on segment `20230827161847` (if I remember correctly, pretraining was performed furing ~2 days with a 4090):



Given that it showed signs of convergence, I repeated the experiments on the same segment, and this time we could see noticeable improvements:



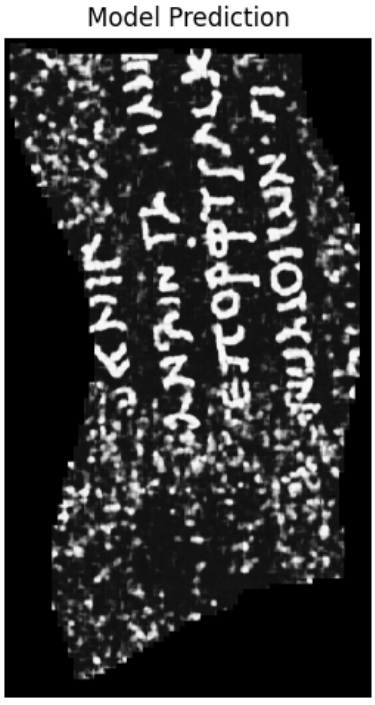
The difference between using a 3D UNet vs. the 2D one is quite noticeable, now we are able to detect clear letters (see the lambda). IThen, I thought that including the other tricks used in the gp (mainly soft-labeling) would improve the results even more. I also did another extra experiment: inferencing with different window sizes: the Vesuvius team recommends that we don't use windows larger than 64x64 to mitigate hallucination risks. Now, I assume that we can train with larger windows (e.g. 224) and perform inference with 64x64 windows.



The model works with smaller window sizes. In fact, it's even less noisy!

Note: At this stage, I noticed a flaw in my evaluation. I only take crops containing ink, including in the inference. This means that I am filtering information that I am not going to have when detecting new ink. As we only perform inference in segments with ink, this is going to provide a lot of information and a clearer model. I should validate with all the crops present in a single mask, or at least do this when performing inference! From now on, the inference will be performed across all the segment, including non-ink regions.

I also included soft labeling, a techniques used in the GP prize that improved the results.

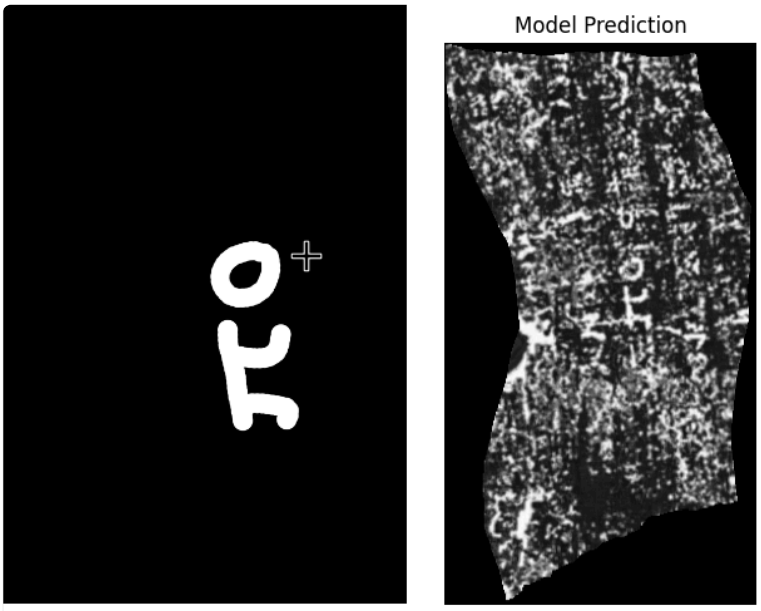


Inferencing the whole segment yields the result above. We can notice more noise but we can clearly read letters. Another cool thing is that I can actually see an epsilon where

the validation label marks a pi (see the bottom row, lambda epsilon nu)! This might be an hallucination or an incorrect ground truth.

Now, what I wanted to test is the following:

1. Pretrain on a whole segment (we've already done this).
2. Start with just one/two annotated letters.
3. Fine-tune on these two letters
4. Check if the model is able to infer more letters
5. Refine them manually.
6. Repeat the process with the extra labels
   Essentially, what I wanted to check is if we could use pretraining to quickstart the iterative labeling process, as Youssef did to obtain the GP model.

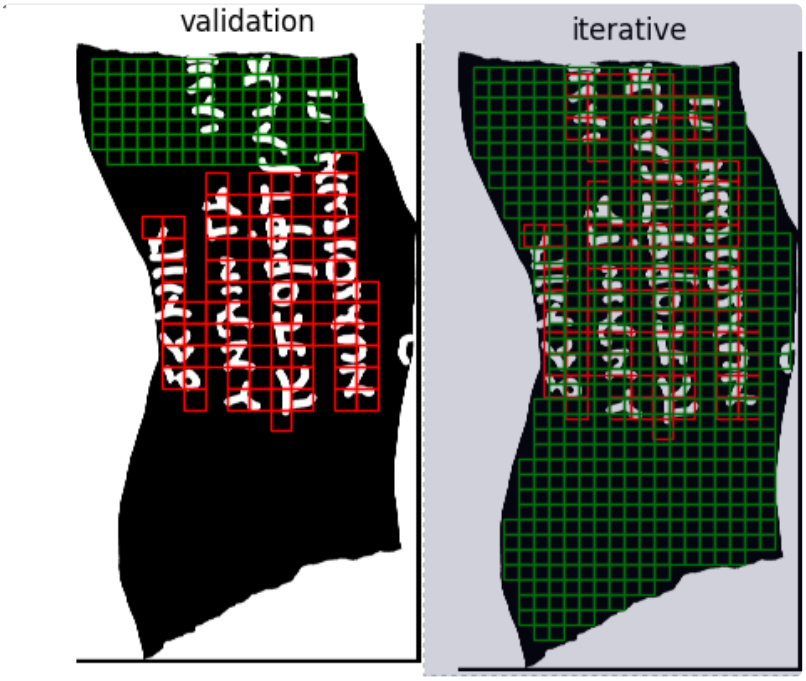


A quick experiment reveals that we could indeed start from one/two letters, or a small annotation, and keep iteratively refining the labels until we get the whole segment! It would be interesting to keep iterating to check if we recover the same label, without looking at the "ground truth". I should also try freezing the encoder.

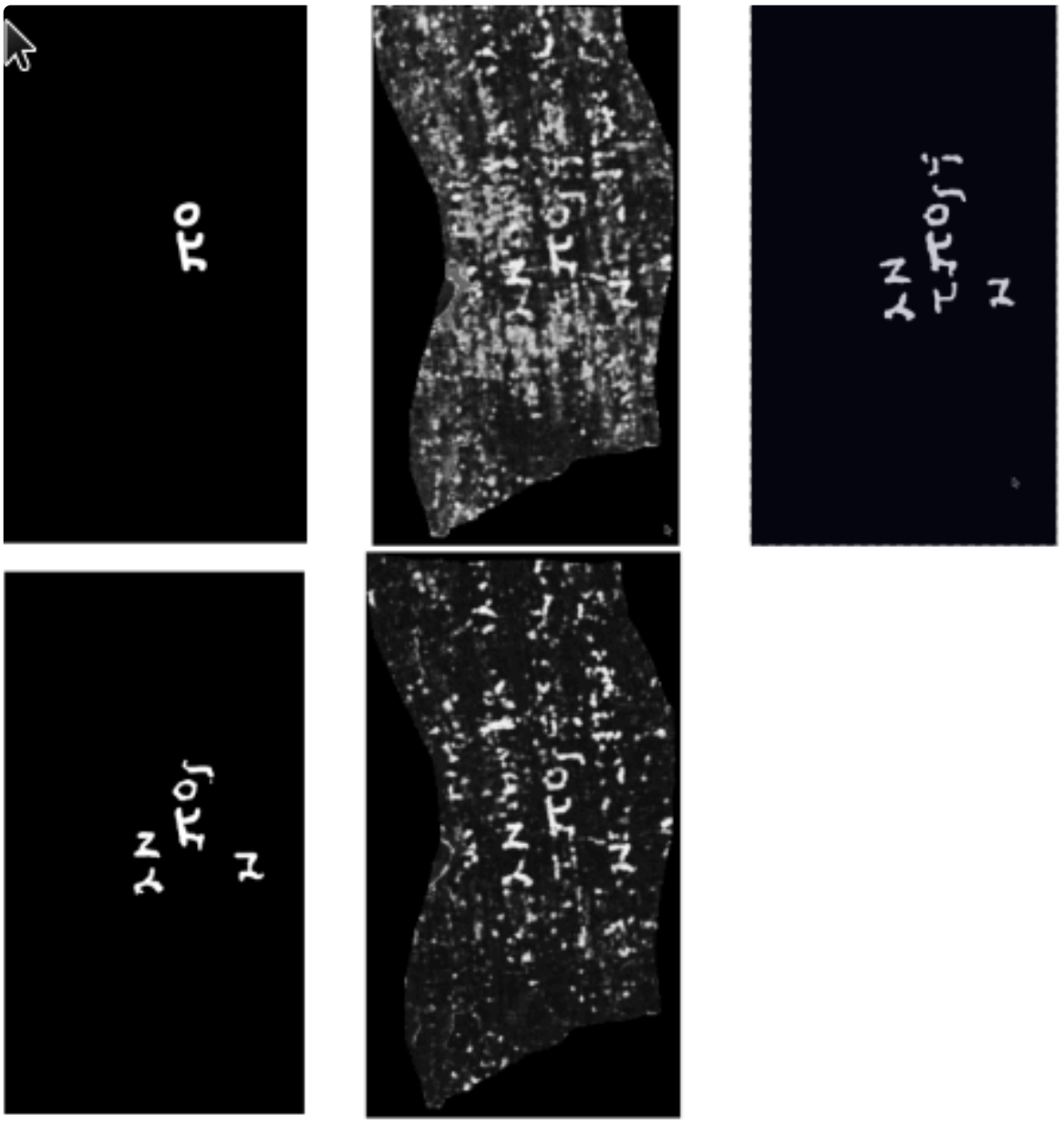
At this stage, I did a big refactor to properly establish two different training schemes:

- **Supervised:** Train on a fraction of regions with ink, evaluate on another region with/without ink
- **Iterative**: Train on all the ink crops provided, evaluate on whole segment

The schemes can be better visualized in the figure below (red is train green is val).



After this, I did a quick iterative labeling test and it looks like it works! The figure should be read left-to-right, top-to-bottom: First, we only train the (pretrained) model with a pi and an omicron. This model yields hints of new letters (such as lambda, tau, etc.). Then, I manually refine the hints of letters, re-train the model and obtain more letters! This is essentially the iterative ink label approach presented by Youssef and its team.



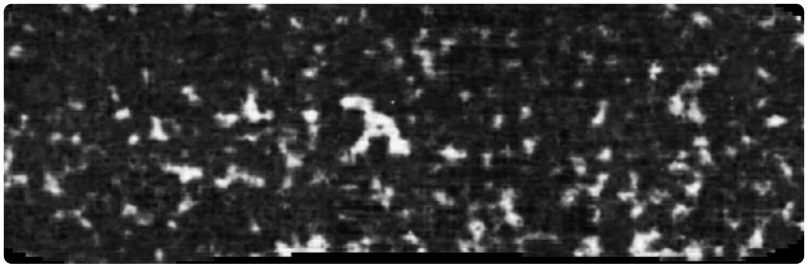
At this stage, I decided to go for the first letters prize by applying this approach in Scroll 4. Concretely, I wanted to to this on one of the segments that had a small amount of inklabels annotated by polytrope.

Which segment from Scroll 4 to choose?

- `20231210132040`: Has an area of 10cm2 (`12122x8790`). It has more hand-labeled annotations but less letters are clear. The other segment has been discussed more, specially the "KA" letters. As a comparison, the Scroll 1 segment where we have experimented (`20230827161847`) has an area of 20.83cm2
- `20240304144031`: Has a (larger) area of `30563x14377`. It has a smaller number of annotations but it has been discussed more.

For now, I decided to try experimenting with the first one.

Then, as shown above, I pretrained for ~6 days on a 4090 and tried to repeat the iterative labeling approach by finetuning the model on just one letter. However, the results are quite bad:



Unfortunately, these last weeks have been busy and I will have less time to work on this for a few months, so I thought about sharing the work until now. Even though it is preliminary, maybe this approach can be used to detect new letters on other scrolls. Also, this method could be used to improve the ink labels that we already have.