

Segment This Thing: Foveated Tokenization for Efficient Point-Prompted Segmentation

Supplementary Material

6. Training Details

6.1. MAE Pre-training

We pre-trained our foveated image encoders using MAE pre-training for 500K iterations. We use an AdamW optimizer with a learning rate of 2^{-13} and weight decay of 0.001. There is a 10K step linear warm-up of the learning rate, after which it is held constant. Instead of periodically dropping the learning rate, we double the batch size every 100K iterations. The initial batch size is 1024. We use the standard masking ratio of 0.75 and do not apply loss to the patches provided to the MAE as input. Fixations are sampled uniformly with a margin of 256 pixels.

6.2. Segmentation Training

We trained our full STT models for an additional 250K iterations. We again use AdamW with a weight decay of 0.001, this time with a learning rate of 2^{-16} , with a 5K step linear warm-up. The batch size starts at 2048 and doubles after 50K steps. We sample up to 16 fixations per image, and again use a sampling margin of 256 pixels. We use loss weights of 20.0 for the focal loss, 1.0 for the dice loss, and 0.01 for the IoU prediction loss.

7. Foveation Pattern

As described in Section 3.1, our foveation patterns consist of a series of nested rings of patches with a dense grid in the center. Here we give a more formal definition of the parameterization of such a pattern.

A pattern with N layers must specify a stride $s_i \in \mathbb{Z}^+$ and a grid size $g_i \in \mathbb{Z}^+$ for each level $i \in [0, N)$. Generally, $s_0 = 1$ such that the dense grid in the center is full resolution. We require $s_i > s_{i-1}$ for all i , i.e. the layers are defined in order of increasing stride. Layer i then defines a grid of patches with a bounding size of $g_i s_i p$, where p is the patch size (in our experiments $p = 16$ pixels). In order to ensure nesting, we further require $g_i s_i > g_{i-1} s_{i-1}$ (the grids get larger from layer to layer), and

$$s_i \mid \frac{g_i s_i - g_{i-1} s_{i-1}}{2},$$

i.e. the difference in sizes between successive grids is an even multiple of the stride of the higher level such that the lower level grid can be centered and surrounded by patches at the next level, leaving no gaps or overlap.

Successive grids redundantly cover some of the same pixels. The patch strides are not constrained to be integer multiples of each other, so retaining redundant patches

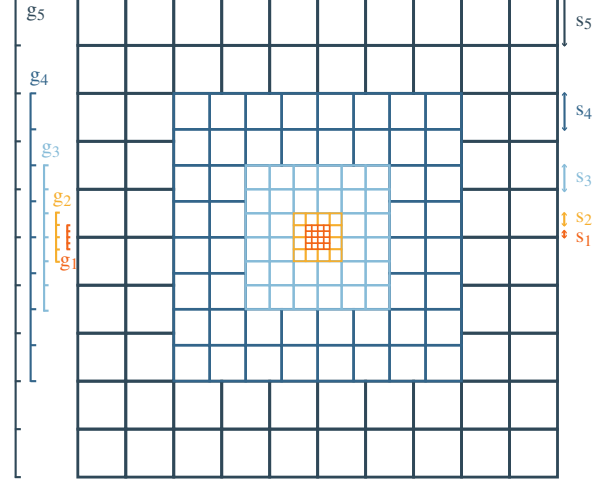


Figure 10. The foveated tokenization pattern as used in our experiments with the Segment This Thing model. Each square in the image indicates the size and location the receptive field of a patch. The patch sizes are all integer multiples of the smallest patch size, such that every patch can be downsampled to the same size using a simple box filter. Patches are colored by level with grid sizes indicated at left and strides at right.

would increase the number of pixels required to represent the image and increase bandwidth requirements. We therefore drop redundant tokens for efficiency, at each level introducing only those patches that cover regions of the image that are not already covered by lower levels.

Level (i)	Stride (s_i)	Grid size (g_i)
1	1	4
2	2	4
3	4	6
4	6	8
5	8	10

Table 3. The precise definition of our foveation pattern. The interpretation of the parameters is given in Section 7. The pattern is depicted in Figure 10.

The total foveation pattern size in pixels is $g_{N-1} s_{N-1} p$ pixels. The total token count can be computed as:

$$\sum_{i=0}^{N-1} g_i^2 - \sum_{i=1}^{N-1} \frac{s_{i-1} g_{i-1}^2}{s_i} \quad (2)$$

where the first term adds all the grid sizes and the second ac-

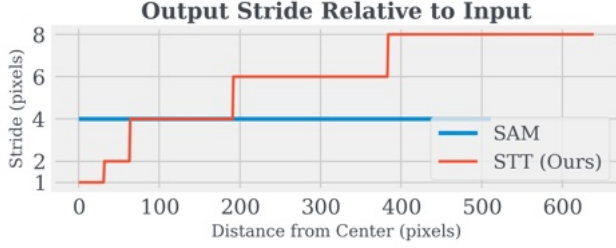


Figure 11. The stride of the segmentation map outputs relative to the input size. SAM outputs segmentation maps at a flat resolution that is one quarter of the input resolution.

counts for the removal of redundant tokens. We trained our model with a five-layer foveation pattern, with parameters listed in Table 3. The pattern and the geometric interpretation of its parameters are visualized in Figure 10.

In Figure 11 we show the input and output stride of the pattern as a function of horizontal or vertical distance from the center. We also plot the output stride of SAM, indicating which regions of our output segmentation maps have lower, higher, or equivalent resolution.

8. MAE Pre-training

To evaluate the effectiveness of MAE pretraining on foveated tokenizations, we trained size B models for 100K iterations with various initial weights: random initialization, a series of MAE checkpoints, and pre-trained publicly available ViT weights. We plot the training loss curves in Figure 12. We see that longer pre-training results in gains that persist through fine-tuning, with eventual diminishing returns. The ViT weights were trained with standard patch tokenization on the regular grid and is initially worse than random initialization. The network is eventually able to repurpose these weights, but after 100K steps of training even a small amount of foveated MAE pre-training yields better results.

9. Varying the Token Count

We also ran a small experiment to evaluate the effect of the foveation pattern on segmentation accuracy. Our foveation patterns exists in a high-dimensional design space, and each new pattern requires its own MAE pre-training. We thus focused on the token count, performing a somewhat abbreviated training run (300K steps pre-training, 100K steps training) for one pattern with fewer tokens and one pattern with more. The patch size and overall receptive field were kept fixed, and all models are size L. The segmentation accuracy is presented in Table 4. As expected, performance increases with increased token count.

Token Count	Cityscapes	EgoHOS	VISOR
100 (-42%)	0.364	0.551	0.531
172	0.390	0.594	0.568
268 (+56%)	0.403	0.614	0.590

Table 4. Segmentation accuracy of models with more or less tokens than the baseline 172-token model. Note that results for the baseline differ from table 5 due to the abbreviated training schedule.

10. Full Evaluation Results

We list the full evaluation results in tabular form in Table 5.

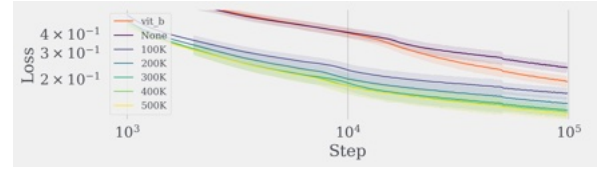


Figure 12. Training Loss by Initialization (Log/log scale)

11. STT Performance Analysis

In this section we describe two investigations into the performance of STT relative to the baselines to deepen the understanding of the model.

11.1. Breakdown by Distance from Prompt

To gain further insight into the relative performance of SAM and STT we measured the pixel-wise precision, recall, and overall accuracy as a function of distance from the prompt on a subset of three evaluation datasets. The results are plotted in Figure 13. Both precision and recall decrease for both models with increased distance as expected. The particular shape of the curve varies significantly by dataset but some trends hold across all three.

The first trend to note is the distinctive "swoosh" shape of the accuracy plots — accuracy is high in the vicinity of the prompt, drops to its lowest value for both models with a somewhat increased distance, then begins to asymptotically approach 1.0. The cause can be clearly seen in the plot of the positive label rate by distance. The nadir of each accuracy curve reliably occurs at the point the positive rate crosses 0.5. We note that even in the datasets where SAM is generally more accurate than STT, there is at least a narrow region centered on the prompt with a radius of 4-8 pixels in which STT is more accurate. This is perhaps due to the higher resolution in this region as shown in Figure 11 and described in Section 3.2.2.

Finally, we note an interesting trend in the precision and recall curves. As distance increases, STT tends towards higher precision and lower recall compared to SAM. This

Method		Accuracy (mIoU)								
		ADE20K	Cityscapes	EgoHOS	NDD20	PPDLS	TimberSeg	VISOR	ZeroWaste	WoodScape
SAM	H	0.543	0.393	0.582	0.826	0.762	0.674	0.604	0.629	0.301
	L	0.537	0.392	0.601	0.817	0.764	0.644	0.606	0.634	0.296
	B	0.547	0.384	0.620	0.798	0.771	0.524	0.599	0.607	0.300
MobileSAM		0.471	0.302	0.557	0.733	0.640	0.338	0.549	0.579	0.234
EfficientSAM	S	0.553	0.405	0.632	0.771	0.678	0.507	0.618	0.628	0.323
	Ti	0.544	0.378	0.615	0.786	0.747	0.473	0.572	0.598	0.320
STT (Ours)	H	0.552	0.410	0.620	0.754	0.730	0.434	0.596	0.620	0.308
	L	0.553	0.412	0.607	0.719	0.758	0.421	0.582	0.595	0.310
	B	0.541	0.393	0.596	0.732	0.735	0.398	0.571	0.583	0.291

Table 5. A full numerical listing of all segmentation accuracy results.

could be a function of the inductive bias in the network caused by the centering of the input on the prompt. STT apparently estimates positive labels for fewer distant pixels, sometimes missing parts of the segment which results in lower recall. On the other hand, when it does estimate positive labels for pixels far from the prompt, it is correct more often than SAM.

11.2. Alternative Evaluation Modes

When evaluating SAM and EfficientSAM, we followed the standard protocol. Both methods accept the full frame as input and then rescale it to 1024×1024 . However, STT requires a 1280×1280 crop centered on the prompt, and we do not rescale the input. If the ground truth segment extends beyond the crop boundaries, STT therefore simply pays the penalty for failing to include those pixels.

This does mean that STT and the baselines receive input images with differing receptive fields. To determine the impact of this, we gave EfficientSAM the same 1280×1280 crops used by STT as input and re-evaluated both methods only on this region (ignoring any labels outside the crop). The results are given in Table 6. We note that both methods see an increase in performance, due to the restricted evaluation domain and, in the case of EfficientSAM, a potential increase in the input resolution after rescaling.

We further evaluate EfficientSAM on *foveated* versions of the same 1280×1280 crop. This is done by passing the crop through the foveation process and then restructuring them into an image as done for visualization (c.f. Section 3.1.1). Evaluating in this mode, both methods receive exactly the same input and EfficientSAM also benefits from the same reduced bandwidth requirements as STT. However, as can be seen in Table 6, the performance of EfficientSAM drops significantly when operating on foveated inputs, indicating that STT can process such reduced-bandwidth data both more efficiently and more effectively.

Model	Eval Mode		Accuracy (mIoU)		
	Cropped	Foveated	Cityscapes	EgoHOS	VISOR
EfficientSAM	✓		0.444	0.640	0.619
	✓	✓	0.410	0.560	0.540
STT (Ours)	✓	✓	0.403	0.604	0.576

Table 6. Evaluating EfficientSAM (size S) and STT (size L) in different evaluation modes.

12. Computing FLOPs

We follow Kaplan *et al.* in computing FLOP counts for transformer architectures (c.f. [19], Table 1). Specifically, we omit non-linearities, biases, normalizations, and other such operations with negligible contributions relative to the FLOPS counts incurred by the remaining operations. However, Kaplan *et al.* focus on language models and give counts per token. We are interested instead in the cost per image. The expressions we use to calculate FLOPS are given below. Note that these expressions yield FLOPS counts for a single layer or application of the indicated function, and thus omit the multiplier by layer count given in [19]. We also include expressions for linear and convolutional layers as these are used to compute the FLOPS in the mask decoder.

Here d_{model} is the hidden dimension of the transformer model d_{attn} is the total dimension over which attention is computed, i.e. the sum of the dimension of each head. These values are typically the same but need not be. d_{ff} is the inverse bottleneck dimension, which is typically set to $4d_{\text{model}}$. Note that we differentiate between the number of keys and the number of values to include cross-attention, as used for example in the two-way transformer in the mask decoder of all models considered. In the case

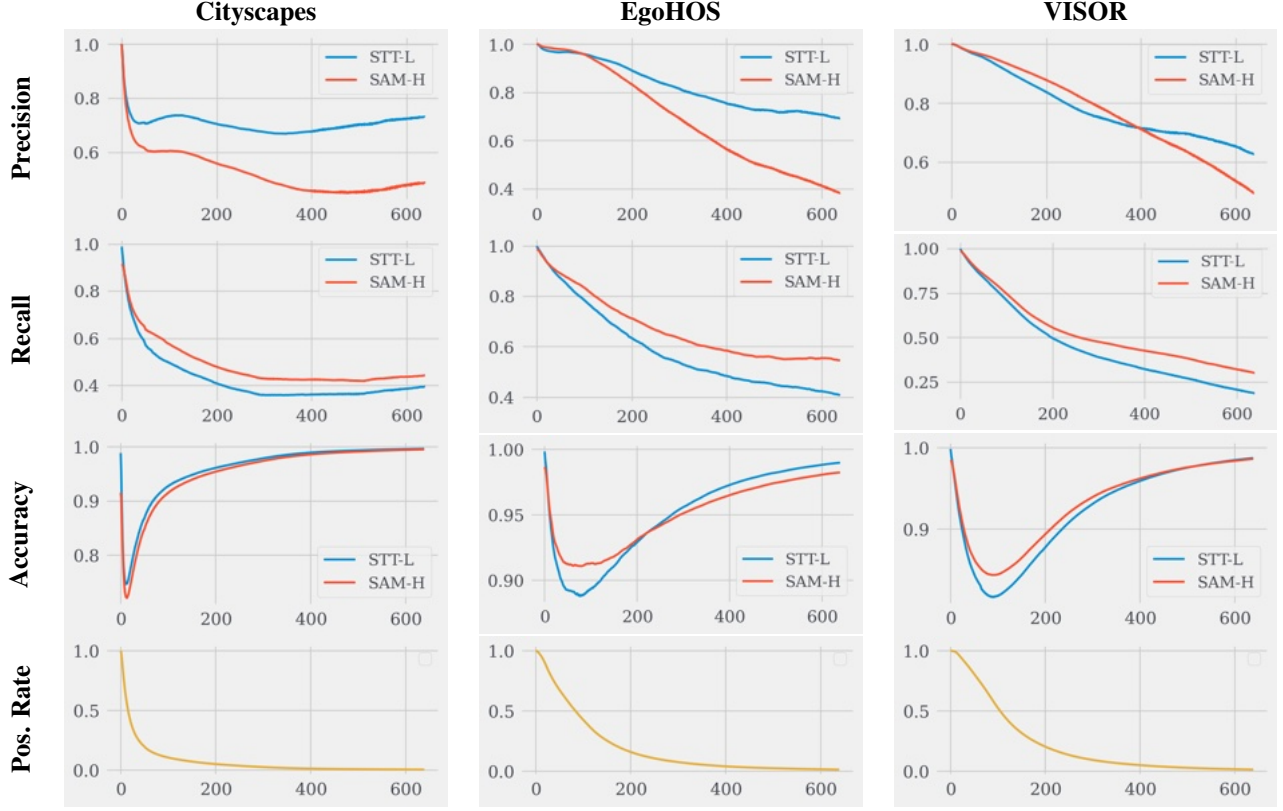


Figure 13. The top three rows show the pixel-wise precision, recall, and accuracy of SAM and STT, respectively, as a function of the distance of a pixel from the prompt. The last row shows the fraction of pixels with a positive label as a function of distance from the prompt.

Operation	FLOPS
Attention: QKV	$2d_{\text{model}}d_{\text{attn}}(n_{\text{query}} + 2n_{\text{key}})$
Attention: QK Logits	$2n_{\text{query}}n_{\text{key}}d_{\text{attn}}$
Attention: Softmax	$3n_{\text{query}}n_{\text{key}}n_{\text{head}}$
Attention: Reduce V	$2n_{\text{query}}n_{\text{key}}d_{\text{attn}}$
Attention: Project V	$2n_{\text{query}}d_{\text{model}}d_{\text{attn}}$
Feedforward	$4n_{\text{query}}d_{\text{model}}d_{\text{ff}}$
Linear	$2n_{\text{vals}}d_{\text{input}}d_{\text{output}}$
Convolution	$2d_{\text{input}}d_{\text{output}}w_{\text{out}}h_{\text{out}}w_{\text{kernel}}h_{\text{kernel}}$

pute the logits and omit all other terms.

of self-attention, $n_{\text{key}} = n_{\text{query}}$.

We also have to treat the windowed attention layers in the SAM image encoder specially. Given a function $flops(d_{\text{model}}, n_{\text{tokens}}, n_{\text{heads}})$ that returns the FLOPS count for a transformer encoder layer (with the default settings of d_{attn} and d_{ff}), the global attention layers in the SAM ViT encoder require about $flops(d_{\text{model}}, s^2, n_{\text{heads}})$ for a token map of size $s \times s$. Given a window size w , the local windowed attention layers require about $\lceil \frac{s}{w} \rceil^2 flops(d_{\text{model}}, w^2, n_{\text{heads}})$ FLOPS. For mask decoders, we include the cost of the MLPs, the deconvolutions, and the dot product used to com-