Appendix

In this additional document, we describe more details of SeiT in Appendix A, including the details of token encoding-decoding algorithms (Appendix A.1), the visualization of Emb-noise augmented tokens (Appendix A.2). We also include additional experimental results in Appendix B, including the hyperparameter details (Appendix B.1), the full experimental results of Table 2 (Appendix B.2), the additional results on storage-efficient pretraining (Appendix B.4), exploring other tokenizers (Appendix B.5), and robustness benchmarks (Appendix B.5).

A. More Details for SeiT

A.1. Pseudo-code for Token Encoding-Decoding

Algorithm 1 and Algorithm 2 describe the psuedo-codes for the proposed token encoding and decoding. Here, we assume $\max t_i < 2^{M+1}$ for the simplicity. For example, in our main experiments, each token belongs to 391 classes and we set $M = 8$, hence, $\max t_i = 391 < 2^{8+1} = 512$. If the number of token classes is larger than 2^{M+1}, then our algorithm can be naturally extended by repeating line 6-7 in Algorithm 1. By this simple algorithm, we achieved a nearly optimal compression ratio (1.11 kB vs. 1.08 kB per image) where almost 0.63 smaller than the 16-bit encoding (2.0 kB per image). Note that, we use the native `gzip` library to perform Huffman encoding and decoding for simplicity.

Algorithm 1 An algorithm for token encoding

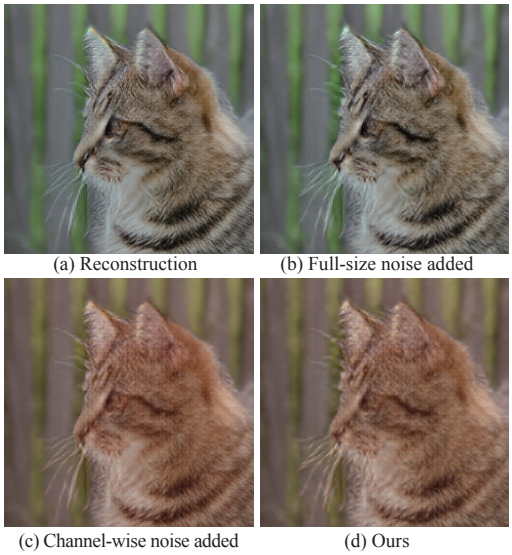
Require: A sequence of tokens $T = [t_1, \ldots, t_N]$, the bits for the storage M

$L_T \leftarrow [\phi]$ ▷ Initialize an empty list for tokens
$L_{\text{idx}} \leftarrow [\phi]$ ▷ Initialize an empty list for start indices
$j \leftarrow 0$
while $i \leq$ N **do**
  if $t_i \geq 2^M$ **then**
    L_T . `append` (2^M)
    L_T . `append` $(t_i - 2^M)$
    $j \leftarrow j + 2$ ▷ Assume $t_i < 2^{M+1}$ for simplicity
  else
    L_T . `append` (t_i)
    $j \leftarrow j + 1$
  end if
  L_{idx} . `append` (j)
  $i \leftarrow i + 1$
end while
Return: `Huffman encoding` (L_T, L_{idx})

Algorithm 2 An algorithm for token decoding

Require: A compressed bytestring $L_T{}'$ from Algorithm 1

$L_T = [t_0, \ldots, t_N], L_{\text{idx}} \leftarrow$ `Huffman deencoding` (L'_T, L'_{idx})
$T \leftarrow [\phi]$
$i \leftarrow 0$
while L_{idx} is not empty **do**
  $j \leftarrow L_{\text{idx}}$. `pop` (0)
  $k \leftarrow i$
  while $k \leq j$ **do**
    if $t_k \geq M$ **then**
      T . `append` $(t_k + t_{k+1})$
      $k \leftarrow k + 2$
    else
      T . `append` (t_k)
      $k \leftarrow k + 1$
    end if
  end while
  $i \leftarrow j$
end while
Return: T



(a) Reconstruction (b) Full-size noise added (c) Channel-wise noise added (d) Ours

Figure A.1. **Emb-Noise visualization.** "Reconstruction" denotes the reconstructed image by the ViT-VQGAN decoder from the extracted tokens. "Full-size noise" is a random noise whose size is equivalent to the embedding vectors.

A.2. ViT-VQGAN decoded images for Emb-Noise

Fig. A.1 and Fig. A.2 show the visualization examples of the Emb-Noise augmented tokens and the tokens without augmentation. We use the ViT-VQGAN decoder for visualization. We observe that our Emb-Noise can make meaningful distortions on the decoded images.

Figure A.2. **Channel-wise modification visualization.** We present ViT-VQGAN decoded images obtained by adding a constant to each of the 32 channels in codebook vectors.

B. Additional Experimental Results

B.1. Hyperparameter details

Table B.1 shows the full list of hyperparameters used in our experiments. All hyperparameters are for the ViT-B backbones. In the table, Token IN-1k corresponds to SeiT (ImageNet-1k) in Table 2, Token IN-21k PT corresponds to token pre-training in Table 3, and Token FT and Image FT correspond to token and image fine-tuning in Table 3, respectively. For other backbones and datasets, we only adjust the learning rate as the maximum learning rate showing a stable convergence (*e.g.*, We use 0.15 for ResNet and ViT-S uses the same learning rate as ViT-B).

B.2. The full experimental results

We report the full experimental results in Table B.5. Details are the same as Table 2.

B.3. Other datasets

The performances of SeiT on various datasets are reported in Table B.2. We tokenize the datasets and then fine-tune a model (ViT-B) with the tokenized dataset, using token-trained model weights. The token-trained models weights, Token (IN-1k) and Token (IN-21k, IN-1k) achieve top-1 accuracies of 74.0%, 81.1% on ImageNet-1k, respectively. The pixel counterpart is fine-tuned on pixel target datasets from the pixel pre-trained model; Pixel (IN-1k), showing 81.8% top-1 accuracy on IN-1k. We followed the pixel-training recipes of DeiT [65]. Although we do not modify the training recipe for tokens, the results verify the possibility of SeiT on those datasets.

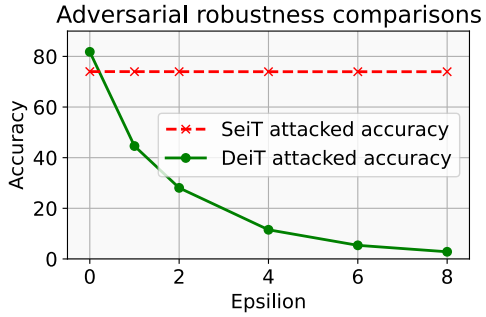


Figure B.1. **Adversarial robustness of DeiT and SeiT by varying ε.** $\varepsilon = 0$ denotes the clean accuracy.

B.4. Three-stage storage-efficient pre-training

Following BeiT v2 [48], we extend our storage-efficient pre-training in three stages, namely, 21k token pre-training $\rightarrow$ 1k token pre-training $\rightarrow$ 1k image fine-tuning. For simplicity, we directly fine-tune the "21k token pre-trained and 1k token fine-tuned model" (*i.e.*, 81.1% model in Table 3) on the image pixels with the same optimization hyperparameter of the image fine-tuned model. As a result, we have 82.8% top-1 accuracy, slightly better than the original two-staged training strategy (+0.2% than 82.6%).

Methods	DeiT IN-1k [65]	Token IN-1k	Token IN-21k PT	Token IN-1k FT	Image IN-1k FT
Epochs	300	300	270	100	100
Batch size	1024	1024	2048	4096	512
Optimizer	AdamW	AdamW	AdamW	AdamW	AdamW
Learning rate	0.0005 x $\frac{bs}{512}$	0.00075 x $\frac{bs}{512}$	0.0015	0.00001	0.0005
Learning rate decay	cosine	cosine	cosine	✘	cosine
Weight decay	0.05	0.1	0.02	0.1	0.05
Warmup epochs	5	5	5	5	5
Label smoothing	0.1	0.1	0.1	0.1	0.1
Dropout	✘	✘	✘	✘	✘
Stoch. Depth	0.1	0.1	0.1	0.15	0.1
Gradient Clip	✘	✘	✘	✘	✘
Cutmix prob.	1	1	1	1	1
Mixup prob.	0.8	0	0	0	0.8
RandAug	9 / 0.5	-	-	-	9 / 0.5
Repeated Aug	✔	-	-	-	✘
Erasing prob.	0.25	-	-	-	0
EDA prob.	-	0.25 (RS) / 0.25 (SR)	0	0	-
Emb-Noise prob.	-	0.5	0.5	0.5	-

Table B.1. **Hyperparamters for SeiT and DeiT-B.** All hyperparameters are for the ViT-B backbone. DeiT IN-1k is the same as the original DeiT paper (baseline).

Pre-trained on	Flowers	Cars	iNat18	iNat19
Pixel (IN-1k)	98.0	91.8	73.0	77.7
Token (IN-1k)	93.5	79.7	43.1	50.1
Token (IN-21k, IN-1k)	98.7	84.5	50.1	58.3

Table B.2. **Other datasets.** We report the top-1 accuracies on diverse fine-grained datasets achieved by SeiT. We tested SeiT on the Flowers [46], StanfordCars [36], iNaturalist (iNat)-18 [28] and iNat-19 [27] datasets.

B.5. Exploring other tokenizers

In this subsection, we explore other tokenizers rather than ViT-VQGAN [72], *e.g.*, VQGAN [21]. We employ four VQGAN models from the official repository[3], ImageNet-trained VQGAN with patch size 16 and vocabulary size 1024, ImageNet-trained VQGAN with patch size 16 and vocabulary size 16384, OpenImages [37]-trained VQGAN with patch size 8 and vocabulary size 256, and OpenImages [37]-trained VQGAN with patch size 8 and vocabulary size 8192. Here, the last VQGAN model is trained with the Gumbel softmax [30, 41] quantization, instead of the original vector quantization by VQ-VAE [67]. Here, we slightly change our Stem-Adopter from 4×4 Conv with stride 2 to 2×2 Conv with stride 1 for tokenizers with patch size 16.

In Table B.3, we report the ViT-S (SeiT) top-1 accuracy on the ImageNet-100 benchmark by varying the choice of tokenizers. We also report the reported ImageNet-1k validation FID score of each tokenizer. In the table, we observe that the top-1 accuracy of SeiT follows the generation quality (FID) if we use the same quantization method (*e.g.*, vector quantization). The ViT-VQGAN shows the best FID (1.28) as well as the best ImageNet performance with SeiT (77.3). While the Gumbel quantized VQGAN achieves the best performance, in practice, we use ViT-VQGAN due to two reasons. First, the storage efficiency: 2886 valid codes need 1.5 times more storage than 391 valid codes. Second, Although the OpenImages [37]-trained VQGAN shows better quality, it needs to be trained on a large-scale external dataset. We did not use the OpenImages-trained VQGAN for a fair comparison with other ImageNet-1k-only training methods.

B.6. Robustness benchmarks

We compare ViT-S models trained on ImageNet-1k with different training strategies using robustness benchmarks. We employ three scenarios: (1) noise and blur scenario (2) domain shift scenario (3) adversarial attack scenario. For the first scenario, we add Gaussian noise and Gaussian blur to the validation images. We use ImageNet-R [26] and Sketch-ImageNet [69] for testing the robustness against domain shifts. Finally, we use a weak version of AutoAttak [17] for measuring adversarial robustness.

As the original DeiT is trained on strong augmentation, such as RandAugment or 3-Augment, we also compare our method with "weak augmented" ViT-S, where it only employs resized random crop (RRC) and CutMix [73]. Our assumption is that because the pixel-trained models are sensitive to imperceptible details, they will be less robust than our approach in noise or adversarial attack scenarios. However, on the other hand, because our method relies on the encoding power of the pre-trained tokenizer, if the employed tokenizer is not a robust feature extractor, our method could

[3] https://github.com/CompVis/taming-transformers

Tokenizer	Training dataset	Quantiztation	Voca size (# of valid voca)	PS	FID	ViT-S (SeiT) Acc
VQGAN	ImageNet	Vector quantization	1024 (454)	16	7.94	75.3
VQGAN	ImageNet	Vector quantization	16384 (971)	16	4.98	76.9
VQGAN	OpenImages	Gumbel quantization	8192 (2886)	8	1.49	79.1
VQGAN	OpenImages	Vector quantization	256 (256)	8	1.49	**81.8**
ViT-VQGAN	ImageNet	Vector quantization	8192 (391)	8	1.28	77.3

Table B.3. **Exploring other tokenizers.** Various ViT-S (SeiT) results on the ImageNet-100 benchmark are shown. We compare various VQGAN tokenizers with ViT-VQGAN by varying the quantization methods (Gumbel softmax vs. vector quantization) the vocabulary size, the valid vocabulary size (the number of classes actually used for the ImageNet-1k training dataset), and the patch size (PS).

Model	Data format	Clean	Gauss. Noise	Gauss. Blur	ImageNet-R	Sketch
ViT-S (DeiT)	Pixels	79.9	75.1 (6.0%)	73.4 (8.1%)	28.8 (63.9%)	29.9 (62.6%)
ViT-S (Weak Aug)	Pixels	78.0	64.7 **(17.1%)**	66.8 (14.4%)	20.8 (73.4%)	18.1 (76.8%)
ViT-S (SeiT, ours)	Tokens	74.0	60.8 (17.3%)	65.3 **(11.2%)**	26.0 **(64.6%)**	23.0 **(68.7%)**

Table B.4. **Robustness evaluation.** We show the clean and robust accuracies against corruptions and domain shifts of each model trained on ImageNet-1k. The performance drops are put in parentheses (lower is better) for robust accuracies.

be more vulnerable than pixel-trained counterparts.

Table B.4 shows the results of the first and the second scenarios. Here, we observe two important findings. First, when we use the same augmentations with the same strength (ViT-S Weak Aug vs. ViT-S SeiT), SeiT shows smaller performance drops on both noise scenarios and domain shift scenarios. On the other hand, when we use strong pixel-level augmentations, the pixel-trained counterpart outperforms our approach. It implies that the key to the input pixel robustness depends on the pixel-level augmentations with severe distortions as observed by previous studies [13, 62]. However, because our method uses only tokens, not pixels directly, investigating how to explore pixel-level distortion augmentations on the token level will be an open question and an interesting future research direction.

We also compare the adversarial robustness of DeiT-S and SeiT-S. We employ the APGD (a step size-free version of PGD attack [42]) with cross-entropy loss and DLR loss, following AutoAttack [17]. Because SeiT employs discrete non-differentiable representations in the computational graph, we employ the straight-through estimator (STE) [9] to estimate the non-differentiable gradients, following Athalye *et al*. [5]. We also evaluate the non-quantized version of the quantizer (*i.e*., omitting the vector quantization process, but using the extracted feature by the encoder directly to the ViT input), but we empirically observe that attacking the non-quantized version cannot drop the performance at all. Instead, we use the STE, also used during the training as well as the previous extensive robustness study [5]. We compare the attacked accuracies of DeiT and SeiT by varying ε (a control parameter for the attack intensity) from 0 to 8 in Fig. B.1. We observe that SeiT shows almost neglectable performance drops even under the strongest attack (showing 73.95 for $\varepsilon = 8$ where 73.98 for $\varepsilon = 0$), where DeiT shows 2.8% top-1 accuracy.

However, we should be careful to interpret Fig. B.1; it could be due to a strong obfuscated gradient effect [5] that cannot be detected by a naive straight-through estimator. Moreover, our method could be vulnerable to the codebook attack by changing the token indices directly, not by perturbing the pixels. However, as an efficient and natural adversarial attack on discrete domains is still an open problem [76] (*e.g*., altering indices as imperceptible to humans but sensitive to machines — only a small index change can make a huge semantic gap, such as replacing "huge" in the previous sentence to "neglectable"), we leave the investigation of advanced adversarial attack methods for SeiT beyond straight-through estimator as future work.

Method		Storage size	# of images	Top1 Acc.
Full-pixels	100%	140 GB	1.28 M	81.8
Uniform random sampling	20%	27.2 GB	0.26 M	59.8
	30%	41.0 GB	0.38 M	69.3
	40%	54.6 GB	0.51 M	74.0
	50%	68.4 GB	0.64 M	76.0
	60%	82.0 GB	0.77 M	77.8
	70%	95.7 GB	0.90 M	78.2
	80%	109.3 GB	1.02 M	79.4
	90%	123.1 GB	1.15 M	81.1
C-score based sampling	20%	26.3 GB	0.26 M	65.1
	30%	39.8 GB	0.38 M	69.4
	40%	53.3 GB	0.51 M	73.3
	50%	66.9 GB	0.64 M	76.9
	60%	80.6 GB	0.77 M	77.5
	70%	94.3 GB	0.90 M	79.2
	80%	108.1 GB	1.02 M	80.4
	90%	121.8 GB	1.15 M	80.9
Adjusting image reolution	10%	5.3 GB	1.28 M	63.3
	20%	9.6 GB	1.28 M	75.2
	30%	16 GB	1.28 M	78.6
	40%	24 GB	1.28 M	79.4
	50%	34 GB	1.28 M	80.9
	60%	46 GB	1.28 M	80.8
	70%	60 GB	1.28 M	81.6
	80%	75 GB	1.28 M	81.6
	90%	93 GB	1.28 M	80.8
Adjusting JPEG quality factor	1	9.3 GB	1.28 M	67.8
	5	11 GB	1.28 M	74.6
	10	14 GB	1.28 M	78.1
	25	23 GB	1.28 M	80.7
	50	34 GB	1.28 M	81.1
	75	50 GB	1.28 M	81.5
	85	66 GB	1.28 M	81.3
	90	79 GB	1.28 M	80.9
	95	113 GB	1.28 M	81.6
SeiT (ImageNet-1k, ours)		1.36 GB	1.28 M	74.0
SeiT (ImageNet-1k-5M, ours)		7.49 GB	5.83 M	78.6
SeiT (ImageNet-1k, OpenImages-VQGAN)		1.36 GB	1.28 M	78.4
SeiT (ImageNet-1k-5M, OpenImages-VQGAN)		7.49 GB	5.83 M	78.7
SeiT (IN-21k tokens $\rightarrow$ 1k tokens, ours)		16 GB	12.4 M	81.1
SeiT (IN-21k tokens $\rightarrow$ 1k tokens, OpenImages-VQGAN)		16 GB	12.4 M	82.3
SeiT (IN-21k tokens $\rightarrow$ 1k pixels, ours)		154 GB	12.4 M	82.6
SeiT (IN-21k tokens $\rightarrow$ 1k tokens $\rightarrow$ 1k pixels, ours)		156 GB	12.4 M	82.8

Table B.5. **The full main results.** The full results of Fig. 1, Table 2 and Table 3. The results in the rows denoted to OpenImages-VQGAN are obtained by utilizing OpenImages-trained VQGAN with patch size 8 and vocabulary size 256 as the tokenizer.