A. Pruning and training details

A.1. Training hyperparameters

In our experiments, we use the same data preprocessing, data augmentation, optimizer setup, and learning rate scheduling scheme as mentioned in Table 9 of the DeiT paper [35], unless otherwise mentioned in the following sections.

A.1.1 Pruning and finetuning

For pruning and finetuning we use the training objective $\mathcal{L} = \alpha\mathcal{L}_{\text{full}} + \mathcal{L}_{\text{CNN}}$ to update the model. We set the balancing factor $\alpha = 1 \cdot 10^5$ and full model distillation temperature $\tau = 20$. For our results reported in Tab. 3 without CNN distillation, we set $\tau = 3$ for the full model distillation objective. The pruning process is performed starting from the pretrained DeiT-Base model, with a fixed learning rate of $0.0002 \times \frac{\text{batchsize}}{512}$. We perform the pruning experiments on the cluster of four NVIDIA V100 32G GPUs, with a batch size of 128 on each GPU. We prune the model continuously until a targeted latency is reached, which is discussed in detail in Appendix A.2. Followed by the iterative pruning we remove the pruned away dimensions of the pruned model to turn it into a small and dense model, and continue to finetune the small model to further recover accuracy. Entire finetuning is performed for 300 epochs with an initial learning rate of $0.0002 \times \frac{\text{batchsize}}{512}$, cosine learning rate scheduling and no learning rate warm up. The finetuning is performed on a cluster of 32 NVIDIA V100 32G GPUs, with a batch size of 144 on each GPU.

A.1.2 Downstream tasks transfer learning

Table 8. Datasets used for downstream task experiments.

Dataset	Train size	Test size	# Classes
CIFAR-10 [21]	50,000	10,000	10
CIFAR-100 [21]	50,000	10,000	100
iNaturalist 2018 [37]	437,513	24,426	8,142
iNaturalist 2019 [37]	265,240	3,003	1,010

The details of the classification datasets used for our downstream task transfer learning experiments are provided in Tab. 8. Similar to the experiment setting of DeiT [35], for downstream task experiments we rescale all the images to 224×224 to ensure we have the same augmentation as the ImageNet training. All models are trained for 300 epochs with a initial learning rate of $0.0005 \times \frac{\text{batchsize}}{512}$, cosine learning rate scheduling and 5 epochs of learning rate warm up. We use batch size 512 for CIFAR-10 and CIFAR-100 models, and batch size 1024 for iNaturalist models.

For Semantic Segmentation, previous work SETR [50] provides an effective downstream model architecture and training pipeline to use ViT models as the backbone model of semantic segmentation tasks [2]. In our experiments we substitute the backbone model with the DeiT/NViT models pretrained on ImageNet. We keep all other downstream architectures and training configurations unchanged. We evaluate the models on the Cityscape dataset [8] and the ADE20K dataset [51]. For the Cityscape dataset, we follow the "SETR_Naive_DeiT_768x768_40k_cityscapes_bs_8" configuration and train on 4 GPUs. For the ADE20K dataset, we follow the "SETR_PUP_DeiT_512x512_160k_ade20k_bs_16" configuration and train on 2 GPUs.

A.1.3 ReViT experiments

For the experiments on ReViT models we use the CNN hard distillation objective as in Eq. (8) as the training objective for all the models. We train Each pair of comparable DeiT and ReViT models with the same set of hyperparameters. In all experiments, we train the model from scratch for 300 epochs with an initial learning rate of $0.0005 \times \frac{\text{batchsize}}{512}$, cosine learning rate scheduling and 5 epochs of learning rate warm up. The models are trained on a cluster of 16 V100 32G GPUs, with a batch size of 48 on each GPU for base models and a batch size of 144 on each GPU for small and tiny models.

A.2. Pruning configuration

We use DeiT-Base model with CNN distillation as the starting point of our pruning process, whose pretrained model is available at https://dl.fbaipublicfiles.com/deit/deit_base_distilled_patch16_224-df68dfff.pth. We prune the model in an iterative manner: We compute the moving average of the latency-aware importance score $\mathcal{I}_{\mathcal{S}}^{L}$ for all unpruned dimension groups in each training step of the pruned model. Every 100 steps, we remove a group of dimensions that has the minimum total importance. Removed dimensions will never be reactivated. We prune EMB and MLP in a group size of 16, QK and V in a group size of 8, and H in a group size of 2, so that the input and output dimensions of all the linear projection operations in the model can be divided by 16, thus satisfying the dimension requirement of the Ampere GPU.

The pruning process will terminate once the estimated latency of the model reaches a targeted speedup ratio over that of the DeiT-base model. The pruned model will then be converted into a small dense model and finetuned to further restore the accuracy. The pseudo code of our pruning algorithm is provided in Algorithm 1

[2] Code publicly available at https://github.com/fudan-zvg/SETR.

Table 9. Pruning configurations and remained dimensions for models reported in Table 1. The reported dimensions are averaged across all the blocks.

Model	Target speedup	Pruning steps	Avg. dim remained				
			EMB	H	QK	V	MLP
DeiT-B	N/A	0	768	12	64	64	3072
NViT-B	1.85×	480	496	8.00	35.33	58.67	1917.3
NViT-H	2.00×	524	480	7.33	32.67	56.67	1816.0
NViT-S	2.56×	642	400	5.83	24.00	47.33	1557.3
NViT-T	5.26×	908	224	3.17	14.67	34.00	930.67

Algorithm 1 Hessian-based latency-aware pruning.

```
# Initialization and preparation
Load pretrained DeiT-B model
Profile latency lookup table as in Appedix A.3
# Iterative pruning
while Estimated latency > target do
    for (X, Y) in Train_Loader do
        for All prunable structural group S do
            Compute I_S with (X, Y) following Equation (6)
            Estimate latency improvement for pruning S
            Compute I_S^L following Equation (7)
        Remove the structural group with min_S I_S^L
        Estimate pruned model latency
        Gradient descent on remaining weights
# Finetuning
Finetune pruned model
```

Tab. 9 reports the target speedup ratio we use to achieve NViT-B, NViT-H, NViT-S and NViT-T architectures reported in Tab. 1. The resulted number of pruning steps and the averaged dimension of EMB, H, QK, V and MLP among all the transformer blocks are also provided.

A.3. Latency lookup table profiling detail

We use a latency lookup table to efficiently evaluate the latency of the pruned model given all its EMB, H, QK, V and MLP dimensions. We initialize the lookup table by profiling the latency of a single vision transformer block on a V100 GPU with batch size 576. We evaluate the latency through a grid of:

- EMB: 0, 256, 512, 768 (latency assigned as 0 at zero EMB);
- H: 1, 3, 6, 9, 12;
- QK: 1, 16, 32, 48, 64;
- V: 1, 16, 32, 48, 64;
- MLP: 1, and 128 to 3072 with interval 128;

resulting into 9375 configurations in total. We run each configuration for 100 times and record the median latency value in the lookup table. For a block with arbitrary dimensions, its latency is estimated via a linear interpolation of the lookup table, which we implement with the *RegularGridInterpolator* function from *SciPy* [39, 42]. The estimated latency of the entire model is computed as the sum of the estimated latency of all the blocks, while omitting the latency of the first projection layer and the final classification FC layer.

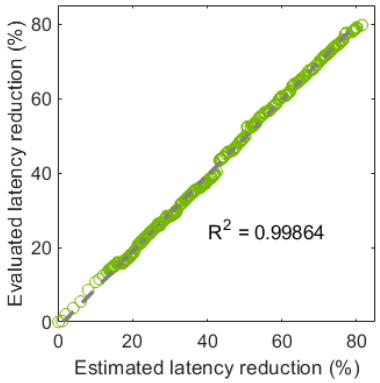

Figure 5. Estimated latency from the lookup table vs. evaluated latency on V100 GPU with batch size 256. Reduction ratio computed with respect to the latency of the full model.

To show the usefulness of the lookup table, we compare the estimated and evaluated latency of different model architectures in Fig. 5. Each point represent the model achieved from a pruning step towards *NViT-T* configuration (See Appendix A.2). The estimated latency and evaluated latency of ViT demonstrate **strong** linear relationship throughout the pruning process, with $\mathcal{R}^2 = \mathbf{0.99864}$. This enables us to accurately estimate the latency improvement brought by removing each group of dimensions, and to use the estimated speedup of the pruned model as the stopping criteria of the pruning process.

B. Additional ablation studies

B.1. Training objective

As discussed in Sec. 3.3, we propose to use a combination of full model distillation and CNN hard distillation as the final objective of our pruning and finetuning process. Here we ablate the validity of this choice and compare the finetuning performance achieved with removing one or both distillation loss from the objective. Specifically, we consider the following 4 objectives:

- Proposed objective: $\mathcal{L} = \alpha\mathcal{L}_{\text{full}} + \mathcal{L}_{\text{CNN}}$;
- CNN distillation only: $\mathcal{L}_{\text{CNN}}$ as in Eq. (8);
- Full model distillation with cross-entropy: $\mathcal{L}_{\text{full}} + \mathcal{L}_{\text{CE}}\left(\Psi\left(\frac{z_c^s + z_d^s}{2}\right), Y\right)$;
- Cross-entropy only: $\mathcal{L}_{\text{CE}}\left(\Psi\left(\frac{z_c^s + z_d^s}{2}\right), Y\right)$.

We use each of the 4 objectives to finetune the pruned model achieved with NViT-T configuration, and report the final Top-1 accuracy in Tab. 10. The finetuning is performed for 50 epochs, with all other hyperparameters set the same as described in Appendix A.1.1. The proposed objective achieves the best accuracy.

Table 10. NVP-T model finetuning accuracy with different objectives.

Objective	Proposed	CNN	Full model	CE only
Top-1 Acc.	**73.55**	73.40	72.62	72.36

B.2. Pruning individual components

In this section we show the result of pruning EMB, MLP, QK and V component individually. The pruning procedure and objective are almost the same as described in Sec. 3.2, other than here we only enable the importance computation and neuron removal on a single component. The pruning interval of EMB, MLP, QK and V are set to 1000, 50, 200 and 200 respectively, in order to allow the model to be updated for similar amount of steps when pruning different components to the same percentage. 32 neurons are pruned for each pruning step. We stop the pruning process and finetune the model for 50 epochs after the targeted pruned away percentage is reached.

The compression rate and accuracy achieved by pruning each component are discussed in Tab. 11. Under similar pruned away ratio, we can observe that pruning EMB leads to the most significant compression on the parameter and FLOPs count, as well as the largest drop in accuracy. This implies that the embedding dimension leads to the most effective exploration on the compression-accuracy tradeoff, which motivates us to use EMB as the key driving factor in analyzing the parameter redistribution in Sec. 5.1.

Table 11. Iterative pruning single component to targeted percentage.

Component	Pruned away	Para (×)	FLOPs (×)	Top-1 Accuracy
Base	0%	1	1	
EMB	50%	1.98	1.92	79.24
MLP	50%	1.49	1.47	82.13
QK	50%	1.09	1.10	82.98
V	50%	1.09	1.10	82.63
EMB	70%	2.95	2.77	73.15
MLP	75%	1.97	1.91	80.29
QK	75%	1.14	1.16	82.64
V	75%	1.14	1.16	81.51

B.3. Effectiveness of head alignment

We also illustrate the benefit of head alignment, where we explicitly single out the head dimension and align the dimensions of each head in structure pruning. We show the tradeoff curve between latency reduction and the accuracy achieved with or without explicit head alignment in Fig. 6. For models pruned without head alignment, we estimate their latency as if all heads are padded with zeros to have the same QK and V dimensions during inference. Under the same latency target, the accuracy achieved with our proposed head-aligned pruning scheme consistently outperforms that of without head alignment, with up to 0.3% accuracy gain.

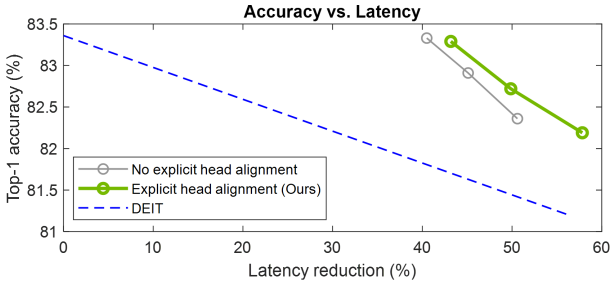

Figure 6. Comparing the parameter reduction-accuracy tradeoff and latency reduction-accuracy tradeoff of different pruning schemes. Latency estimated on RTX 2080 GPU. Model size compression rate and latency reduction rate are computed based on that of the DeiT-Base model respectively.

B.4. Effectiveness of Hessian importance score

In our pruning method we claim that utilizing a Hessian-based importance score is the key factor to allow global structural pruning in the ViT models. Here we perform perform an ablation study on pruning with the magnitude-based criteria, where the group with the smallest L2 norm will be pruned in each step. We prune the model to match the latency of DeiT-S, and compare with our NViT-S performance.

All the other hyperparameters are set the same. Results are shown in Tab. 12. It can be seen that magnitude-based pruning struggles to reach the latency target with a larger number of steps, while the pruned model accuracy is much worse. Looking at the remained dimension of the magnitude-based pruning unveils that most of the structural components are either unpruned or all pruned away, which infers magnitude-based criteria is incomparable across different structural components and different layers, thus unsuitable for global pruning.

Table 12. Comparing magnitude-based pruning vs proposed NVP. The pruned model accuracy before finetuning is reported.

Method	Pruning steps	Para (×)	FLOPs (×)	Top-1 Accuracy
Magnitude	968	4.14	4.26	33.79
NViT-S	642	4.18	4.24	76.59

B.5. Correlation between Hessian importance score and real loss difference

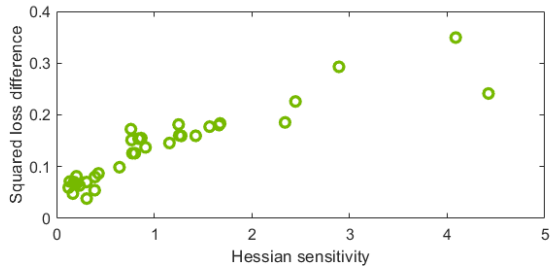

Figure 7. Hessian importance score vs. squared loss difference.

In this section we verify the theoretical result derived in Sec. 3.2.1, on estimating the loss difference induced by pruning with the proposed Hessian importance score. We evaluate the squared model loss increase for performing a single structural pruning step on different structural components of the DeiT-B model, and plot it with the corresponding importance score computed for the pruned structure following the derivation in Eq. (6). All the loss differences and Hessian importance score are estimated on the same batch of 64 training images. As shown in Fig. 7, we observe strong positive correlation between the estimated sensitivity and the real loss difference.

B.6. Effectiveness of latency-aware regularization

In Tab. 13 we show the result of pruning without latency regularization, i.e. set $\eta = 0$ in the importance score formulated in Eq. (7), and compare with our NViT results. Both models are pruned to match DeiT-S latency. We can see from the result that pruning with latency-aware regularization can help reaching the target latency quicker, while achieving higher accuracy under the latency budget. To better understand the difference in the achieved architecture, we also show the average dimension across all the blocks after pruning. It can be seen that model pruned with latency regularization tends to have more dimensions on MLP and less on MSA (QK and V), which is in line with our observation made in Sec. 5.1 on designing more efficient ViT architecture, where reducing dimensions related to the attention (H, QK, V) while increasing MLP dimension may lead to more accurate model under similar latency.

B.7. Performance on low-end GPUs

As one of the main motivation for pruning is to enable model deployment on low-end devices with cheaper cost and lower energy consumption. To this end we further examine the latency of running the pruned NViT models on NVIDIA Jetson NANO, a commonly used low-end GPU for embedded system. Here we utilize a batch size of 64 for ImageNet inference.

For base model, we note that DeiT-B cannot fit into the memory of the device, preventing it from being compiled onto the NANO device. Whereas our pruned NViT-B model can run with a decent speed, reaching 83.3% Top-1 acc. NViT-T matches the speed of DeiT-T, and the speedup over NViT-B is consistent to our measurement on V100 reported in Tab. 1 ($2.8\times$ on NANO vs. $2.7\times$ on V100). This further demonstrates that for low-end devices NViT enables the originally prohibitive high-performance model to run, while the speedup achieved on high-end devices can be retained.

C. Additional parameter redistribution analysis

C.1. Attention head diversity

As we observe in Fig. 4 and mentioned in Sec. 5.1, the pruned models tend to preserve *more* dimensions in the transformer blocks towards the *middle* layers, while having *less* dimensions towards the *two ends* of the model. Here We explore an intuitive analysis on why this trend occurs by observing the diversity of features captured in each transformer blocks. Given the attention computation serves important functionality in ViT models, here we use the diversity of the attention score learned by each head as an example. Specifically, we take a random batch of 100 ImageNet validation set images, pass them through the pretrained DeiT-Base model and our NViT-B model, and record the averaged attention score $softmax\left(\frac{q_h k_h^T}{\sqrt{d_h}}\right)$ of all the images computed in each head h. We then compute the pair-wise cosine distance of the attention score from each head as a measure of diversity, and visualize the results in Fig. 8.

In DeiT-B model, we can observe that in earlier blocks like block 2 and later blocks like block 11, there are clear patches of darker blue indicating a group of heads having attention scores similar to each other. While for blocks in the

Table 13. Comparing pruning results with (η =5e-4) or without ($\eta = 0$) latency-aware regularization. The pruned model accuracy before finetuning is reported. The reported dimensions are averaged across all the blocks.

					Avg. dim remained				
η	Pruning steps	Para (×)	FLOPs (×)	Acc.	EMB	H	QK	V	MLP
0.	657	4.11	4.17	74.80	416	5.7	25.3	49.3	1510.7
5e-4 (NViT-S)	642	4.18	4.24	76.59	400	5.8	24.0	47.3	1557.3

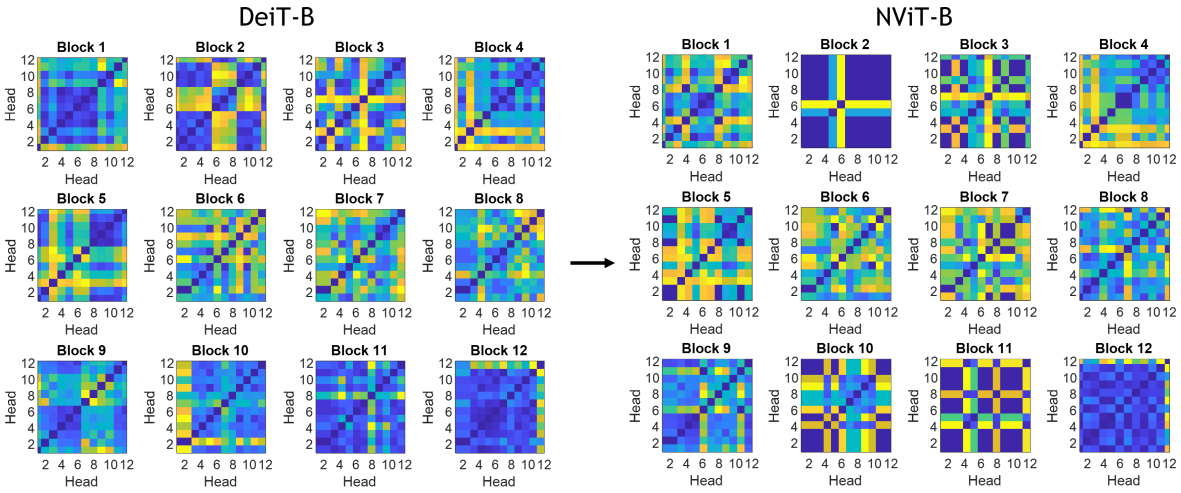

Figure 8. Pair-wise cosine distance between all heads' attention score in each transformer block. Blue indicates a smaller distance while yellow indicates a larger one. The dark blue blocks in NViT-B figures corresponds to the heads being pruned away, which have all-zero attention scores thus zero cosine distance in between.

middle such as block 5-8, almost all pairs of heads appear to be fairly diverse. Such difference in diversity leads to different behavior in the pruning process, where less heads are preserved in earlier and later blocks while more are preserved in the middle. Note that all remaining heads in NViT-B model appears to be diverse with each other, showing a more efficient utilization of the model capacity. Interestingly, this less-more-less trend of dimensional change across different transformer is not observed in previous works compressing BERT model for NLP tasks [25, 26, 40]. The learning dynamic of ViT model leading to this trend is worth investigating in the future work.

C.2. Parameter redistribution on SWIN

We have shown the effectiveness of the proposed pruning method on pruning SWIN-Transformer stages. In this section, we examine the effectiveness of the discovered parameter redistribution rule of DeiT on the Swin-Transformer model. Though SWIN follows a multi-stage design that is different from DeiT, within each stage all the transformer blocks have the same dimension, which gives us the potential of exploring better dimension redistribution rules. Here we take SWIN-T model, with 2-2-6-2 transformer blocks in stage 0-3 respectively. As the redistribution rule treats the first/last block and intermediate blocks differently, the rule mainly takes effect on stage 2 with 6 blocks. The parameter redistribution is performed following exactly the same ReViT rule as reported in Tab. 6. Specifically, the dimensions of each transformer block in the redistributed SWIN-ReViT-T is reported in Tab. 14.

Table 14. Redistributed SWIN-ReViT-T model Stage-2 dimensions.

Block	1	2	3	4	5	6
EMB	384	384	384	384	384	384
Head	10	4	8	8	4	10
QK/Head	32	16	32	32	16	32
V/Head	64	64	64	64	64	64
MLP	1152	1152	2304	2304	1152	1152

We train the SWIN-ReViT-T model on ImageNet following the same training scheme described in the official GitHub repo [3]. The model statistics and training performance of the resulted SWIN-ReViT-T is compared with the original SWIN-T in Tab. 15.

[3] https://github.com/microsoft/Swin-Transformer

Table 15. Comparing the efficiency and accuracy of SWIN-ReViT-T vs. SWIN-T on ImageNet. The throughput is evaluated with a single TITAN RTX GPU.

Model	Parameters	FLOPs	Throughput	Top-1 Accuracy
SWIN-T	29M	4.5G	546.37 img/s	81.3%
SWIN-ReViT-T	**28M**	**4.4G**	**574.25** img/s	81.3%

The redistributed SWIN-ReViT-T model achieves the same Top-1 accuracy as the original model with 1.1x speedup. This indicates that the redistribution rule derived on DeiT can also be transferred to other ViT variants to achieve efficiency improvements.

C.3. The significance of ReViT-S performance gain

Table 16. Repeated experiments of ReViT-S and DeiT-S training.

Model	Ckpt 1	Ckpt 2	Ckpt 3	Ckpt 4	Ckpt 5	Mean	STD
DeiT-S	80.96	80.93	80.95	81.01	80.92	80.954	0.035
ReViT-S	81.17	81.19	81.17	81.20	81.22	81.190	0.021

As we report the accuracy improvement brought by ReViT-S over DeiT-S in Tab. 7, here we verify the significance of this improvement via repeated experiments. Specifically, we report the Top-1 accuracy of 5 checkpoints for training ReViT-S and DeiT-S from scratch on ImageNet in Tab. 16. Note that the averaged 0.23% Top-1 accuracy gain of ReViT-S over DeiT-S is 10 times the standard derivation of repeated experiment results, showing the improvement is truly significant.