6 Appendix

6.1 Full experiment results

In addition to our main paper examples, we test our method across a diverse range of 44 pre-trained models. These models vary in scale, architecture, development year, and purpose. Our method consistently reduces computation significantly in all cases, with minimal or no loss in accuracy.

Pool of models Models in our experiments include:

1. ConvNeXtv2 [66]: *atto*, *femto*, *pico*, *nano*, *tiny*, *base*, *large*, *huge*
2. DaViT [14]: *tiny*, *small*, *base*
3. MaxViT [59]: *tiny*, *small*, *base*, *large*
4. SwinT [41]: *tiny*, *small*, *base*, *large*
5. ResNet [25]: *18*, *34*, *50*, *101*, *152*
6. EfficientNet [57]: *b0*, *b1*, *b2*, *b3*, *b4*
7. EfficientViT [2]: *b0*, *b1*, *b2*, *b3*
8. EfficientViT [40]: *m1*, *m2*, *m3*, *m4*, *m5*
9. EfficientFormerv2 [35]: *s0*, *s1*, *s2*, *l*
10. MobileNetv3 [27]: *small100*, *large100*

All these models are trained in their original ways with the input size of 224×224 for the ImageNet-1k classification task. Some models are pre-trained with larger datasets [66]. The model weights of ResNet is provided by torchvision [44], all other models are provided by timm [65] and their complexities are measured by the profiling tool of DeepSpeed [49]. Each model can employ a smaller one as its saver, as described in Sec. 3.3.

All models were trained using their original procedures with an input size of 224×224 for the ImageNet-1k classification task. Some models are pre-trained on larger datasets [66]. ResNet weights are available from torchvision [44], while weights for other models are accessed via timm [65]. We assessed their complexities using DeepSpeed's profiling tool [49]. According to the methodology described in Sec. 3.3, each model could utilize a smaller one as its saver.

Compression effect Tab. 5 complements Tab. 1, providing extensive experimental results that reinforce our primary findings discussed in Sec. 4.1. Typically, larger models exhibit more substantial compression potential and can maintain their performance even after significant reductions. In contrast, compressing smaller models presents challenges, as their corresponding saver models are relatively larger in proportion. The smallest models, EfficientViT$_{m0}$ and MobileNetv3$_{small100}$, are not used as base models due to the absence of smaller eligible savers. Moreover, for effective zero-loss compression, saver models need to closely approximate the accuracy of their base models; otherwise, a trade-off between performance and compression is inevitable.

Base Model			Saver Model			Required MACs(%) when Acc. Drop ≤				Max Performance	
Name	GFLOPs.	Acc.	Name	GFLOPs.	Acc.	1%	0.5%	0.01%	0%	Avg. GMACs	Acc.
ConvNeXtV2$_{huge}$ [66]	114.92	86.25%	EfficientFormerV2$_{l}$ [35]	2.51	83.53%	13.21 (-88.5%)	19.41 (-83.1%)	31.17 (-72.9%)	67.34 (-41.4%)	70.70 (-38.5%)	86.25% (+0.00%)
MaxViT$_{large}$ [59]	42.80	84.82%	EfficientFormerV2$_{l}$ [35]	2.51	83.53%	3.46 (-91.9%)	5.48 (-87.2%)	7.81 (-81.8%)	9.04 (-78.9%)	11.24 (-73.7%)	84.92% (+0.09%)
ConvNeXtV2$_{large}$ [66]	34.36	85.76%	EfficientFormerV2$_{l}$ [35]	2.51	83.53%	5.16 (-85.0%)	7.15 (-79.2%)	11.00 (-68.0%)	15.15 (-55.9%)	17.49 (-49.1%)	85.78% (+0.02%)
Swin$_{large}$ [41]	34.02	86.24%	EfficientFormerV2$_{l}$ [35]	2.51	83.53%	5.56 (-83.7%)	7.22 (-78.8%)	10.95 (-67.8%)	14.46 (-57.5%)	18.70 (-45.0%)	86.26% (+0.02%)
MaxViT$_{base}$ [59]	23.39	84.80%	EfficientFormerV2$_{l}$ [35]	2.51	83.53%	2.95 (-87.4%)	4.13 (-82.3%)	5.33 (-77.2%)	6.74 (-71.2%)	7.28 (-68.9%)	84.86% (+0.06%)
ConvNeXtV2$_{base}$ [66]	15.35	84.87%	EfficientFormerV2$_{s2}$ [35]	1.22	82.15%	2.74 (-82.1%)	3.57 (-76.8%)	4.85 (-68.4%)	5.44 (-64.6%)	6.35 (-58.6%)	84.92% (+0.05%)
DaViT$_{base}$ [14]	15.22	84.48%	EfficientFormerV2$_{s2}$ [35]	1.22	82.15%	2.40 (-84.3%)	3.16 (-79.2%)	4.40 (-71.1%)	5.19 (-65.9%)	6.50 (-57.3%)	84.53% (+0.04%)
Swin$_{base}$ [41]	15.13	85.14%	EfficientFormerV2$_{s2}$ [35]	1.22	82.15%	2.81 (-81.4%)	3.70 (-75.6%)	4.79 (-68.3%)	9.06 (-40.1%)	9.40 (-37.9%)	85.15% (+0.00%)
ResNet$_{152}$ [25]	11.51	78.24%	EfficientViT$_{m2}$ [40]	0.20	70.78%	4.17 (-63.8%)	4.91 (-57.3%)	6.30 (-45.3%)	8.79 (-23.7%)	9.92 (-13.8%)	78.27% (+0.03%)
MaxViT$_{small}$ [59]	11.31	84.33%	EfficientFormerV2$_{s2}$ [35]	1.22	82.15%	2.01 (-82.2%)	2.51 (-77.8%)	3.53 (-68.8%)	3.86 (-65.9%)	4.29 (-62.1%)	84.42% (+0.08%)
DaViT$_{small}$ [14]	8.58	84.00%	EfficientFormerV2$_{s2}$ [35]	1.22	82.15%	1.65 (-80.8%)	2.06 (-76.0%)	2.92 (-66.0%)	3.25 (-62.1%)	3.54 (-58.7%)	84.06% (+0.05%)
Swin$_{small}$ [41]	8.51	83.25%	EfficientNet$_{b3}$ [57]	0.96	78.63%	2.58 (-69.6%)	3.11 (-63.5%)	4.09 (-51.9%)	4.41 (-48.2%)	5.67 (-33.4%)	83.34% (+0.09%)
ResNet$_{101}$ [25]	7.80	77.26%	EfficientViT$_{b0}$ [2]	0.10	71.35%	2.10 (-73.1%)	2.53 (-67.5%)	2.93 (-62.5%)	3.46 (-55.7%)	4.43 (-43.3%)	77.38% (+0.12%)
MaxViT$_{tiny}$ [59]	5.37	83.36%	EfficientFormerV2$_{s2}$ [35]	1.22	82.15%	1.29 (-76.0%)	1.52 (-71.7%)	1.82 (-66.0%)	1.96 (-63.5%)	2.57 (-52.2%)	83.58% (+0.22%)
ConvNeXtV2$_{tiny}$ [66]	4.46	82.94%	EfficientNet$_{b2}$ [57]	0.66	77.89%	1.65 (-62.9%)	1.98 (-55.6%)	2.54 (-42.9%)	3.50 (-21.4%)	4.54 (+2.0%)	82.96% (+0.01%)
DaViT$_{tiny}$ [14]	4.41	82.71%	EfficientFormerV2$_{s2}$ [35]	1.22	82.15%	-	1.23 (-72.1%)	1.37 (-68.9%)	1.42 (-67.9%)	2.06 (-53.3%)	83.23% (+0.51%)
Swin$_{tiny}$ [41]	4.35	80.89%	EfficientNet$_{b0}$ [57]	0.39	77.70%	0.91 (-79.0%)	1.15 (-73.5%)	1.42 (-67.4%)	1.54 (-64.5%)	2.07 (-52.3%)	81.09% (+0.19%)
ResNet$_{50}$ [25]	4.09	75.85%	EfficientViT$_{b0}$ [2]	0.10	71.35%	0.93 (-77.2%)	1.11 (-72.9%)	1.31 (-67.9%)	1.36 (-66.7%)	1.87 (-54.2%)	76.15% (+0.30%)
EfficientViT$_{b3}$ [2]	3.88	83.14%	EfficientNet$_{b0}$ [57]	0.39	77.70%	1.33 (-65.6%)	1.59 (-59.0%)	2.08 (-46.4%)	3.17 (-18.4%)	3.89 (+0.3%)	83.16% (+0.01%)
ResNet$_{34}$ [25]	3.66	73.2%	MobileNetV3$_{small100}$ [27]	0.06	67.65%	1.11 (-69.7%)	1.35 (-63.0%)	1.57 (-57.3%)	1.63 (-55.6%)	2.01 (-45.1%)	73.36% (+0.16%)
EfficientFormerV2$_{l}$ [35]	2.51	83.53%	MobileNetV3$_{large100}$ [27]	0.22	75.78%	0.99 (-60.6%)	1.17 (-53.4%)	1.49 (-40.6%)	2.51 (+0.1%)	2.51 (+0.1%)	83.53% (+0.00%)
ConvNeXtV2$_{nano}$ [66]	2.45	81.87%	MobileNetV3$_{large100}$ [27]	0.22	75.78%	0.88 (-63.9%)	1.04 (-57.5%)	1.34 (-45.2%)	1.76 (-28.2%)	2.45 (+0.3%)	81.87% (+0.00%)
ResNet$_{18}$ [25]	1.81	69.53%	MobileNetV3$_{small100}$ [27]	0.06	67.65%	0.19 (-89.4%)	0.28 (-84.3%)	0.37 (-79.5%)	0.41 (-77.2%)	0.84 (-54.0%)	70.31% (+0.78%)
EfficientViT$_{b2}$ [2]	1.55	81.91%	MobileNetV3$_{large100}$ [27]	0.22	75.78%	0.64 (-58.9%)	0.74 (-52.5%)	0.95 (-38.6%)	1.65 (+6.2%)	1.65 (+6.2%)	81.91% (+0.00%)
EfficientNet$_{b4}$ [57]	1.50	79.40%	MobileNetV3$_{large100}$ [27]	0.22	75.78%	0.46 (-69.4%)	0.54 (-63.8%)	0.64 (-57.7%)	0.66 (-56.1%)	0.78 (-48.4%)	79.66% (+0.26%)
ConvNeXtV2$_{pico}$ [66]	1.37	80.31%	MobileNetV3$_{large100}$ [27]	0.22	75.78%	0.52 (-61.9%)	0.61 (-55.4%)	0.71 (-48.0%)	0.81 (-40.6%)	1.05 (-23.1%)	80.38% (+0.07%)
EfficientFormerV2$_{s2}$ [35]	1.22	82.15%	MobileNetV3$_{large100}$ [27]	0.22	75.78%	0.57 (-53.2%)	0.64 (-47.2%)	0.82 (-33.0%)	1.02 (-16.2%)	1.05 (-13.8%)	82.17% (+0.01%)
EfficientNet$_{b3}$ [57]	0.96	78.63%	MobileNetV3$_{large100}$ [27]	0.22	75.78%	0.33 (-66.0%)	0.38 (-61.0%)	0.43 (-55.4%)	0.45 (-53.2%)	0.55 (-42.4%)	79.01% (+0.38%)
ConvNeXtV2$_{femto}$ [66]	0.78	78.49%	MobileNetV3$_{large100}$ [27]	0.22	75.78%	0.31 (-60.2%)	0.36 (-53.4%)	0.42 (-46.7%)	0.43 (-44.6%)	0.51 (-34.9%)	78.76% (+0.27%)
EfficientNet$_{b2}$ [57]	0.66	77.89%	MobileNetV3$_{large100}$ [27]	0.22	75.78%	0.27 (-59.4%)	0.30 (-53.9%)	0.34 (-47.8%)	0.36 (-45.5%)	0.46 (-30.9%)	78.29% (+0.39%)
EfficientFormerV2$_{s1}$ [35]	0.63	79.69%	MobileNetV3$_{large100}$ [27]	0.22	75.78%	0.34 (-46.7%)	0.38 (-39.9%)	0.43 (-31.4%)	0.50 (-21.4%)	0.58 (-8.6%)	79.76% (+0.06%)
EfficientNet$_{b1}$ [57]	0.57	77.57%	EfficientViT$_{b0}$ [2]	0.10	71.35%	0.26 (-54.9%)	0.28 (-50.2%)	0.33 (-42.4%)	0.35 (-38.8%)	0.38 (-33.3%)	77.64% (+0.06%)
ConvNeXtV2$_{atto}$ [66]	0.55	76.64%	MobileNetV3$_{large100}$ [27]	0.22	75.78%	-	0.23 (-57.6%)	0.26 (-52.8%)	0.26 (-51.7%)	0.39 (-29.6%)	77.33% (+0.68%)
EfficientViT$_{m5}$ [40]	0.52	77.08%	MobileNetV3$_{large100}$ [27]	0.22	75.78%	0.23 (-55.7%)	0.25 (-51.2%)	0.28 (-44.9%)	0.29 (-43.0%)	0.38 (-27.1%)	77.55% (+0.47%)
EfficientViT$_{b1}$ [2]	0.51	79.12%	MobileNetV3$_{large100}$ [27]	0.22	75.78%	0.30 (-40.8%)	0.34 (-33.4%)	0.37 (-26.8%)	0.40 (-22.6%)	0.45 (-11.5%)	79.20% (+0.08%)
EfficientNet$_{b0}$ [57]	0.39	77.70%	EfficientViT$_{b0}$ [2]	0.10	71.35%	0.21 (-45.6%)	0.23 (-39.5%)	0.29 (-25.2%)	0.33 (-14.6%)	0.36 (-7.2%)	77.71% (+0.01%)
EfficientFormerV2$_{s0}$ [35]	0.38	76.24%	EfficientViT$_{b0}$ [2]	0.10	71.35%	0.18 (-51.0%)	0.20 (-46.0%)	0.22 (-41.3%)	0.23 (-38.8%)	0.26 (-30.6%)	76.38% (+0.13%)
EfficientViT$_{m4}$ [40]	0.30	74.32%	MobileNetV3$_{small100}$ [27]	0.06	67.65%	0.16 (-47.3%)	0.18 (-40.3%)	0.21 (-29.2%)	0.23 (-21.6%)	0.26 (-12.9%)	74.34% (+0.02%)
EfficientViT$_{m3}$ [40]	0.26	73.38%	MobileNetV3$_{small100}$ [27]	0.06	67.65%	0.14 (-46.2%)	0.16 (-39.7%)	0.18 (-30.8%)	0.19 (-26.6%)	0.21 (-18.3%)	73.45% (+0.06%)
MobileNetV3$_{large100}$ [27]	0.22	75.78%	MobileNetV3$_{small100}$ [27]	0.06	67.65%	0.14 (-33.6%)	0.16 (-26.5%)	0.19 (-12.2%)	0.22 (+2.8%)	0.22 (+3.4%)	75.78% (+0.00%)
EfficientViT$_{m2}$ [40]	0.20	70.78%	MobileNetV3$_{small100}$ [27]	0.06	67.65%	0.09 (-54.5%)	0.10 (-48.3%)	0.12 (-40.9%)	0.12 (-38.8%)	0.15 (-22.6%)	71.13% (+0.35%)
EfficientViT$_{m1}$ [40]	0.16	68.32%	MobileNetV3$_{small100}$ [27]	0.06	67.65%	-	0.06 (-64.1%)	0.06 (-60.7%)	0.07 (-59.6%)	0.11 (-31.9%)	69.41% (+1.09%)
EfficientViT$_{b0}$ [2]	0.10	71.35%	MobileNetV3$_{small100}$ [27]	0.06	67.65%	0.08 (-16.5%)	0.09 (-8.7%)	0.10 (-0.6%)	0.10 (+3.7%)	0.11 (+11.5%)	71.46% (+0.10%)

Table 5: Compression effect while TinySaver applied to all models in the pool

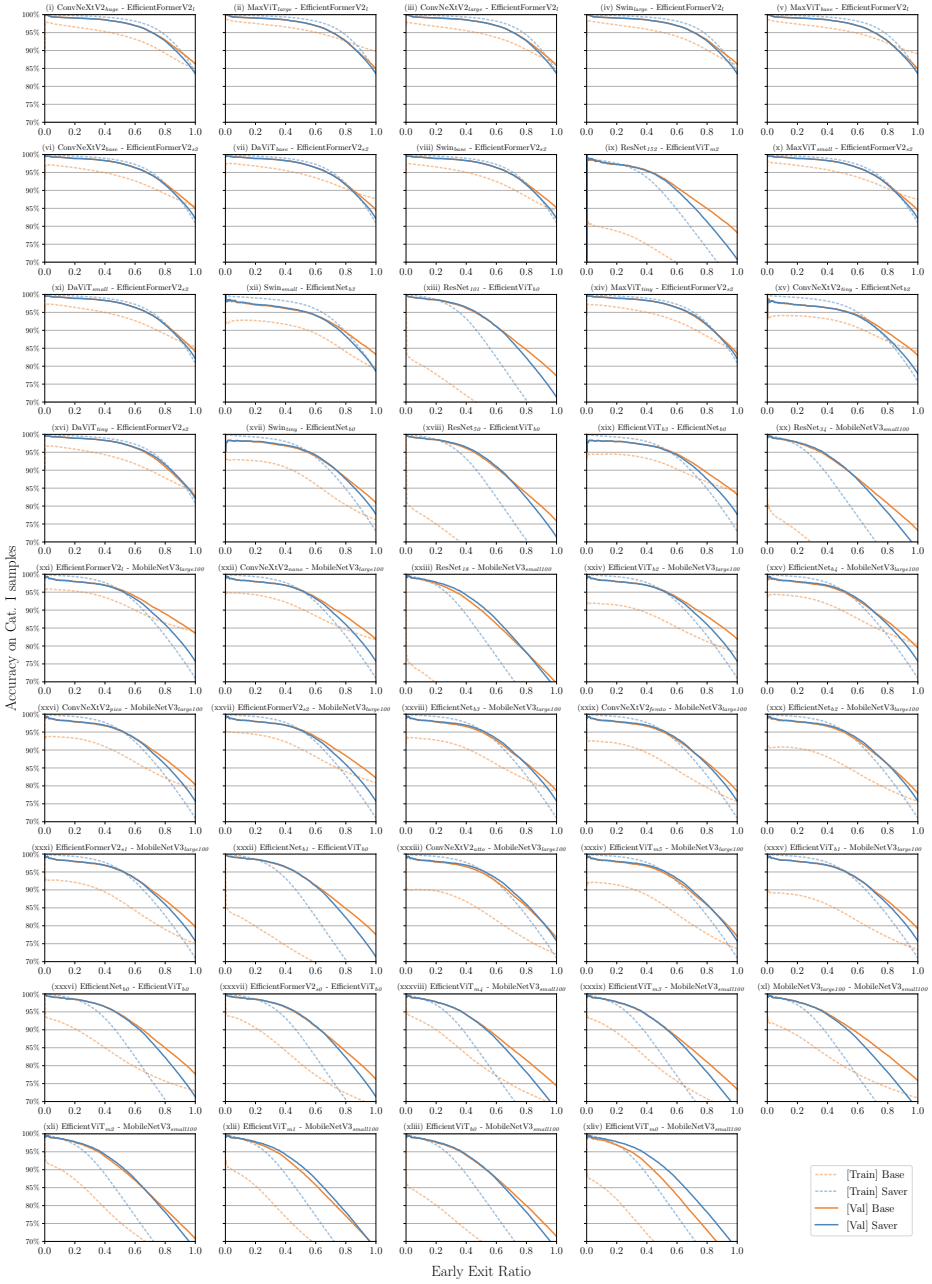

Fig. 6: Accuracy vs early exit ratio on the ImageNet-1k training/validation set. The intersection of solid lines denotes the ratio where the saver and base model have the equivalent accuracy on the validation set. The intersection of dashed lines are for the training set. Every plot is labelled as **Base - Saver**.

Performance and early exit ratio Fig. 6 expands on Fig. 3, illustrating how accuracy varies as more samples are processed by the saver model. In all test cases, the saver model initially excels on the training set. Cat. I samples are filtered based on saver's confidence and the saver can perform exceptionally well with high-confidence samples. While base models struggle with these same samples, leading to poorer performance when Cat. I is small. Additionally, data augmentation complicates early performance for base models. However, as Cat. I expands to include more samples, the situation changes. Finally, if we set saver's threshold as 0 and let all samples be Cat. I, the base will apparently outperform the saver.

On the validation set, the significant initial advantage of the saver model decreases, but it remains more accurate until a later intersection point. This consistent trend across all plots demonstrates the reliability of models when they are highly confident, which is the source of compression effect.

Nonetheless, the base model's underperformance on the training set results in a relative overconfidence in the saver model. Although the validation set is not directly impacted, this discrepancy could adversely affect the accuracy of saver selection, particularly in multi-exit systems like ESN, which rely on exit policies derived from training data. Introducing more data augmentation on the training set may help reduce the discrepancy between training and validation performances.

6.2 Speed test

In our experiments, we use original pre-trained models, which are implemented differently by each author. Their heterogeneity makes them unevenly supported on different platforms. Therefore, it's difficult to compare FLOPs reduction and actual speedups directly. However, for a given deployment scenario, we can replace FLOPs with measured latency while identifying the best saver model. The speed test on a PC with i9-11900F and one RTX 3090 is below. We ran inference on IN-1k val set, and recorded the model throughput. All TinySaver enabled models were configured to have no accuracy loss. Tab. 6 shows the results. Though models exhibit varing efficiency on different deployment scenarios, we can still observe that TinySaver can significantly improve the throughput of the base model.

6.3 Exit Sequence Network (ESN)

The design of the ESN comprises three components: (1) Embedding units that transform features at various scales into a format compatible with the subsequent sequential model. (2) A sequential feature bus designed to aggregate features from all preceding steps and transmit them to subsequent stages. (3) Head units, responsible for outputting results like regular attached exits.

To ensure efficiency, the overall scale of the ESN should be kept moderate, thereby minimizing the introduction of overhead. While over-simplified networks

Base Model	Saver Original→Ours Avg. throughput img/s			
	bs=1,CPU	bs=1	bs=128	bs=512
ConvNextv2_h	Swin_b 1.9→6.4	ConvNextv2_l 55.0→98.5	Swin_b 96.9→307.8	OOM
Swin_b	$\text{EfficientFormerV2}_l$ 9.9→17.7	ConvNextv2_b 97.2→117.5	DaViT_t 487.0→659.7	DaViT_t 496.3→688.8
EfficientNet_{b2}	ConvNextv2_a 51.9→81.4	ConvNextv2_a 150.9→229.4	EfficientViT_{m5} 2172.1→2649.7	EfficientViT_{m5} 2237.5→3344.4
$\text{EfficientFormerV2}_{s2}$	ConvNextv2_n 34.0→42.0	ConvNextv2_n 83.8→222.3	EfficientViT_{b2} 562.1→1339.9	EfficientViT_{b2} 570.8→1542.8

Table 6: Average throughput of models with TinySaver enabled on different deployment scenarios

might not be able to keep its functions. Careful tuning should be required for the best practice of ESN. We illustrate the detail of ESN in Fig. 7

Feature embedding In our approach, both saver and base models are considered well-trained and remain unchanged during ESN training. Their features, treated as a sequence of tokens, need embedding prior to joining the attention layer. In our ESN design, we employ small networks each containing a global average pooling layer and a linear layer, to embed intermediate backbone features.

Sequential feature bus The primary objective of this part is to ensure the delivery of the most informative feature for each head unit. The sequential execution is mandatory, aligning with the step-by-step nature of the dynamic inference process. For this purpose, we have chosen to implement a transformer decoder network recognized for its effectiveness across a broad of applications. Meanwhile, the first step, i.e. features from the saver model, are critical in our system. The transformer allows for direct referencing of these features without requiring intermediary states. This is another reason to select transformers. Contrasting with the commonly used Feature Pyramid Network (FPN) [37], our Exit Sequence Network (ESN) does not accept backward connections from deeper layers. This is required by EE to inference progressively in practice. Therefore, our implementation uses mask to maintain causality.

Exiting heads Exiting heads are responsible for generating the output. We also brings the saver model's output, making $\tilde{\boldsymbol{y}}_n = \tilde{\boldsymbol{y}}_S + \Delta\tilde{\boldsymbol{y}}_n$. The weight of ESN heads are initialized as zero and only need to learn $\Delta\tilde{\boldsymbol{y}}_n$. Therefore, the intermediate outputs combine the information from the saver and base model and their performance is lower bounded by the saver model.

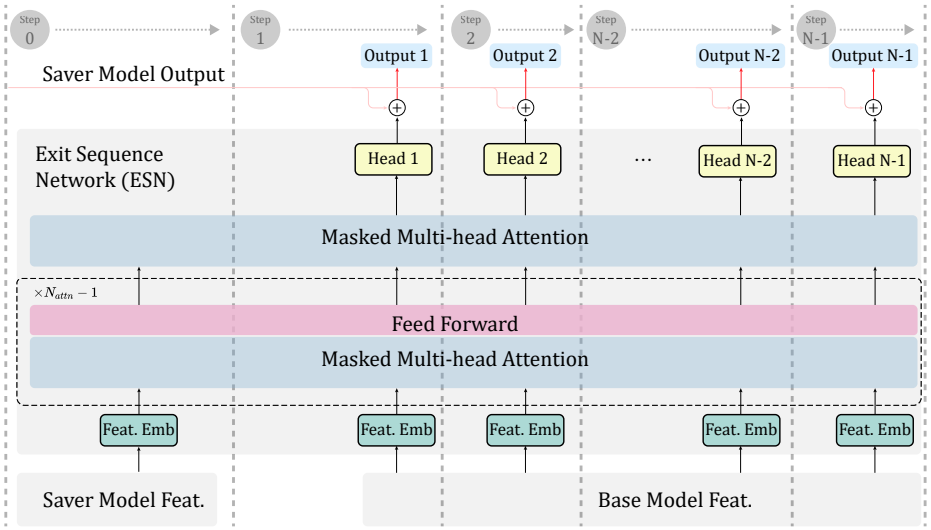

Fig. 7: Detailed ESN

6.4 Configuration and training details of ESN

Fig. 7 shows detailed structure of our prototype design of the ESN. We extract intermediate features from each residual block's output, integrating them with saver features into the ESN of multi-head causal attention layers. Subsequently, single-layer classifiers at each exit generate predictions, all supervised by the loss function detailed in Sec. 6.4. Tab. 7 includes the hyper-parameter of training ESN. We run a basic grid search of suitable attention layer numbers and dims.

Fig. 7 illustrates the detailed structure of our ESN prototype. We extract intermediate features from the output of each backbone segment and combine them with features from the saver into the ESN, which includes multi-head causal attention layers. Then, single-layer classifiers at each exit are responsible for generating predictions, trained by the loss function detailed in Sec. 6.4. Additionally, Tab. 7 lists the hyperparameters used for training the ESN, where we conducted a basic grid search to identify optimal numbers of attention layers and their dimensions.

Loss function In traditional Early Exit (EE) models, the training usually employs the original loss function or knowledge distillation. However, in our approach, a significant portion of inputs is processed solely by the saver model, meaning intermediate exits do not influence these particular inputs. Additionally, lightweight networks have limited capacity for knowledge representation, suggesting the utility of reducing the burden on such networks from samples unlikely to reach intermediate exits. Consequently, we adjust the weight of samples in the loss function based on the saver model's confidence levels. The loss

config	value
optimizer	AdamW
learning rate	6.25×10^{-4}
learning rate schedule	cosine decay
warmup epochs	20
augmentation	AutoAugment [9]
mixup	0.8
cutmix	1.0
training epochs	10
num attn heads	8
num attn layers	{0, 1, 2, 3}
dim	{8, 32, 128, 512}

Table 7: Configurations for training ESN

function applied at the n^{th} exit is outlined in Eq. (11), reflecting this adaptation to reduce the load on intermediate exits while maintaining effective learning.

$$L_n = \sum_{m=0}^{M} 2(1 - \underset{i}{max}(\tilde{\boldsymbol{y}}_S^{(m,i)}))\mathcal{L}(\tilde{\boldsymbol{y}}_n^{(m)}) \tag{11}$$

Where $max_i(\tilde{\boldsymbol{y}}_S^{(m,i)})$ is the confidence of the saver model. $\mathcal{L}(\tilde{\boldsymbol{y}}_n^{(m)})$ is the regular loss function per sample and we use cross entropy in the experiment of this paper.