In this Supplementary Material, we provide additional details of our method. Section A elaborates on key implementation aspects, including the importance-based rearrangement strategy and the customized improvements to the NSGA-II algorithm. Section B describes the training setup for Curriculum Elastic Adaptation and outlines the hyperparameters used for different datasets. Section C presents additional results, including comparisons under more evaluation metrics and detailed accuracy across multiple MACs levels. Section D presents the architectural configurations of the selected submodels under varying MACs constraints across multiple datasets. Section E discusses some potential applications.

A. Method Details

A.1. Importance Rearrangement

Before constructing the elastic structure, we perform importance-based rearrangement across the expandable dimensions in the Vision Transformer (ViT). This process prioritizes critical units, ensuring they are shared across a larger number of submodels within the nested elastic structure. As a result, important units are sampled more frequently during training, receive more updates, and achieve better robustness and transferability across different submodel configurations.

Embedding Dimension Importance. A Vision Transformer (ViT) consists of a stack of L identical Transformer blocks, where each block includes a Multi-Head Attention (MHA) module and a Feed-Forward Network (MLP), both equipped with residual connections and Layer Normalization (LN).

Given the initial input sequence $\mathbf{x}^{(0)} \in \mathbb{R}^{N\times D_{\text{emb}}}$, the feature propagation across blocks is:

$$\mathbf{x}^{(l+1)} = \text{Block}^{(l)}(\mathbf{x}^{(l)}), \quad l = 0, \dots, L-1. \tag{9}$$

Each block computes:

$$\begin{aligned} \mathbf{y}^{(l)} &= \mathbf{x}^{(l)} + \text{MHA}(\text{LN}(\mathbf{x}^{(l)})), \\ \mathbf{x}^{(l+1)} &= \mathbf{y}^{(l)} + \text{MLP}(\text{LN}(\mathbf{y}^{(l)})). \end{aligned} \tag{10}$$

To assess the importance of each embedding channel, we use the final block output $\mathbf{x}^{(L)}$. For each embedding dimension $i \in \{1, \dots, D_{\text{emb}}\}$, the importance score is computed as:

$$F_{\text{emb}}^{(i)} = \sum_{\mathbf{s}\in\mathcal{X}} \left\| \mathbf{x}^{(L,i)}(\mathbf{s}) \right\|_1, \tag{11}$$

where $\mathcal{X}$ denotes the sampled input set, and $\mathbf{x}^{(L,i)}(\mathbf{s})$ represents the activation of the i-th embedding dimension for sample $\mathbf{s}$.

MLP Hidden Dimension Importance. Each MLP module consists of two linear layers with an intermediate hidden dimension $D_{\text{hid}} = r \times D_{\text{emb}}$, where r is the expansion ratio. Given an input $\mathbf{z} \in \mathbb{R}^{N\times D_{\text{emb}}}$, the MLP computation proceeds as:

$$\mathbf{h} = \mathbf{z}\mathbf{W}_1, \quad \mathbf{z}' = \phi(\mathbf{h})\mathbf{W}_2, \tag{12}$$

where $\mathbf{W}_1 \in \mathbb{R}^{D_{\text{emb}}\times D_{\text{hid}}}$, $\mathbf{W}_2 \in \mathbb{R}^{D_{\text{hid}}\times D_{\text{emb}}}$, and $\phi(\cdot)$ is the activation function (e.g., GELU).

The importance of each hidden neuron $j \in \{1, \dots, D_{\text{hid}}\}$ is measured as:

$$F_{\text{mlp}}^{(j)} = \sum_{\mathbf{s}\in\mathcal{X}} \left\| \phi\left(\mathbf{h}^{(l,j)}(\mathbf{s})\right) \right\|_1, \tag{13}$$

where $\mathbf{h}^{(l,j)}(\mathbf{s})$ denotes the activation of the j-th hidden neuron at layer l for sample $\mathbf{s}$.

MHA Attention Head Importance. For each block, the MHA module applies self-attention to the input $\mathbf{z}^{(l)}$. For head $m \in \{1, \dots, h^{(l)}\}$, the queries, keys, and values are computed as:

$$\mathbf{q}^{(m)} = \mathbf{z}^{(l)}\mathbf{W}_Q^{(m)}, \quad \mathbf{k}^{(m)} = \mathbf{z}^{(l)}\mathbf{W}_K^{(m)}, \quad \mathbf{v}^{(m)} = \mathbf{z}^{(l)}\mathbf{W}_V^{(m)}, \tag{14}$$

where $\mathbf{W}_Q^{(m)}, \mathbf{W}_K^{(m)}, \mathbf{W}_V^{(m)} \in \mathbb{R}^{D_{\text{emb}}\times d_{\text{head}}}$ are the projection matrices, and d_{head} is the dimension per head.

The attention output for head m is:

$$\mathbf{a}^{(l,m)} = \text{softmax}\left(\frac{\mathbf{q}^{(m)} \left(\mathbf{k}^{(m)}\right)^\top}{\sqrt{d_{\text{head}}}} \right) \mathbf{v}^{(m)}. \tag{15}$$

The full MHA output is obtained by concatenating all heads:

$$\text{MHA}(\mathbf{z}^{(l)}) = \text{Concat}\left[\mathbf{a}^{(l,1)}, \dots, \mathbf{a}^{(l,h^{(l)})}\right] \mathbf{W}_O, \tag{16}$$

where $\mathbf{W}_O \in \mathbb{R}^{h^{(l)} d_{\text{head}}\times D_{\text{emb}}}$ is the output projection matrix.

The importance score of each attention head m is computed as:

$$F_{\text{head}}^{(m)} = \sum_{\mathbf{s}\in\mathcal{X}} \left\| \mathbf{a}^{(l,m)}(\mathbf{s}) \right\|_1, \tag{17}$$

where $\mathbf{a}^{(l,m)}(\mathbf{s})$ denotes the output of head m for input $\mathbf{s}$.

In practice, we use 1024 samples to compute the importance scores. After obtaining all scores, the embedding dimensions, MLP hidden neurons, and MHA heads are individually rearranged in descending order according to their importance.

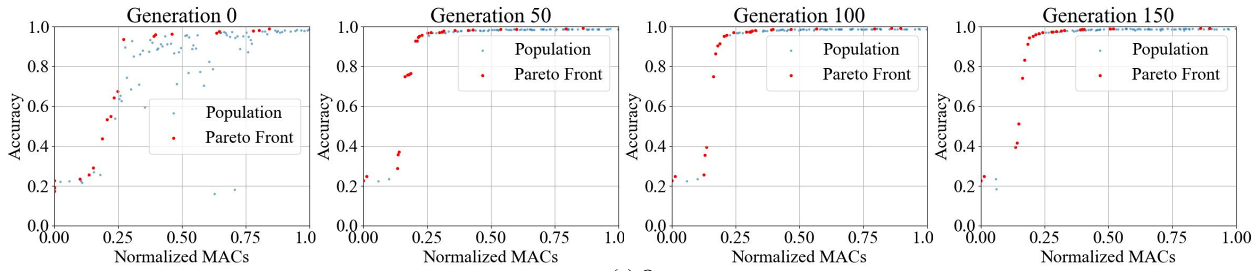
A.2. Details of NSGA-II for Submodel Search

In the main text, we briefly introduced how we apply NSGA-II to identify the Pareto front of submodels balancing complexity and accuracy. Here, we provide a detailed description.

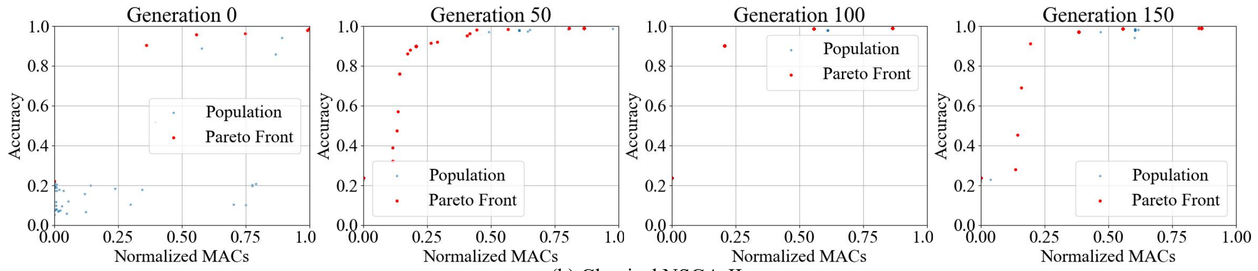
Principle. NSGA-II is a classic multi-objective evolutionary algorithm designed to find Pareto-optimal solutions. It maintains a population of candidates, evolving them through crossover and mutation operations. Selection is based on fast non-dominated sorting, which ranks solutions into Pareto fronts, and crowding distance, which maintains diversity within the population.

Encoding. We encode each submodel configuration as a compact binary sequence.

For the embedding dimension, we reduce the search space by setting $D_{\text{emb}} = k \times d_{\text{head}}$, where k is a scaling factor controlling the embedding size and d_{head} is the dimension per attention head. The value of k is encoded using k binary bits, and during decoding,



(a) Ours



(b) Classical NSGA-II

Figure 7. **Comparison between the proposed customized NSGA-II algorithm (a) and the standard NSGA-II baseline (b).** The customized variant yields a more diverse and well-distributed population across MACs, offering broader coverage and avoiding the redundancy typically exhibited by the standard approach.

the final embedding dimension is obtained by summing the active bits and multiplying by d_{head}.

For the MLP expansion ratio r, we adopt an 8-bit binary encoding, allowing r to vary between 0.5 and 4.0. Similar to the embedding dimension, the final expansion ratio is derived by summing the activated bits.

For the number of attention heads h in the MHA module, we again allocate k binary bits. The number of active heads is determined by summing the corresponding bits during decoding.

Finally, for depth control, we allocate L binary bits for the MHA path and another L bits for the MLP path, where L is the total number of blocks. Each bit specifies whether the corresponding block is retained (1) or skipped (0) during submodel instantiation.

Search Procedure. During the search phase, we use a mini-batch of 1024 samples randomly drawn from the training set to evaluate candidate submodels. However, due to the small batch size and the risk of overfitting on the training set, directly applying standard NSGA-II—based on non-dominated sorting with crowding distance—may result in limited solution diversity and poor coverage of the search space.

To mitigate this, we introduce a partitioned selection strategy. Specifically, we divide the normalized MACs range into 20 intervals and perform independent selection within each interval, ensuring that the overall population achieves broad coverage across different model complexity levels.

Within each partition, a minimum complexity difference between selected individuals is enforced to enhance diversity. During selection, individuals from the current Pareto front are prioritized. For subsequent fronts, crowding distance is recalculated jointly over the candidate set and the already selected individuals. Selection is then performed based on the updated crowding distance ranking, encouraging a more diverse and well-distributed submodel population across the entire search space.

Initialization. At the initialization stage, we construct the population using only two types of individuals: one with all-zero encoding and one with all-one encoding, corresponding to the two extremes of model complexity. New offspring are subsequently generated through crossover and mutation operations. Compared to random initialization, this strategy provides better coverage of the search space boundaries and significantly accelerates convergence during early generations.

Setting. We set the population size to 100, the crossover probability to 0.95, the mutation probability to 0.3, and the number of iterations to 300.

Results. As shown in Figure 7, we compare the performance of the classical NSGA-II and our improved NSGA-II on the SVHN dataset after 0, 50, 100, and 150 iterations, under identical initialization conditions. We visualize both the population distribution and the corresponding Pareto fronts at each stage. Our improved NSGA-II exhibits faster convergence, higher-quality solutions, and greater population diversity. In contrast, the standard NSGA-II suffers from noticeable redundancy, where the population collapses into a few repetitive solutions as the iteration progresses.

The Figure 8 shows the evolution of the Pareto front over generations across nine datasets, demonstrating the effectiveness of our search strategy in progressively identifying high-quality submodels under varying resource constraints.

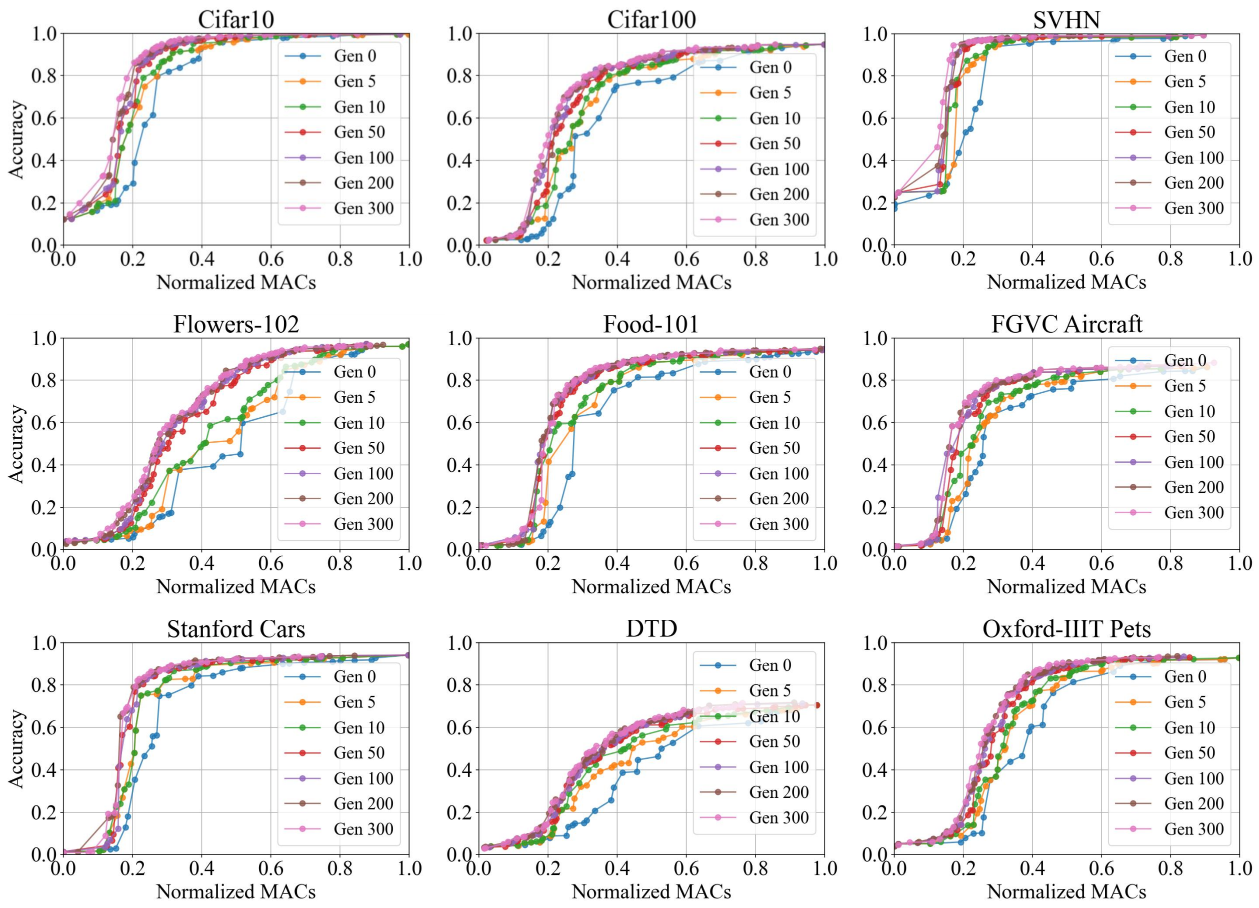


Figure 8. **Evolution of the Pareto front over generations on nine image classification datasets.**

B. Training Hyperparameter Settings

B.1. Curriculum Elastic Adaptation Training Setting

We conduct our experiments using ViT-Base as the backbone. The training configuration for the Curriculum Elastic Adaptation stage is detailed below.

The model is trained for a total of 100 epochs, with elasticity expansion scheduled at epochs $\{10, 15, 20, 25, 30, 35\}$. At each expansion step, the sampling ranges of the submodel hyperparameters are progressively broadened to increase the model's elasticity.

Initially, the upper bounds are set as:

$$R_{\max} = 4, \quad H_{\max} = 12, \quad E_{\max} = 768, \quad n_{\max} = 0,$$

where $R_{\max}$ denotes the maximum MLP expansion ratio, $H_{\max}$ maximum number of attention heads, $E_{\max}$ the maximum embedding dimension, and $n_{\max}$ controls the maximum number of layers that can be skipped.

The decrements applied at each expansion step are configured as:

$$\Delta_R = \{1, 0.5, 0\}, \quad \Delta_H = 1, \quad \Delta_E = 64, \quad \Delta_n = 1,$$

where Δ_n is applied only during the first two expansion steps to gradually introduce layer skipping.

By the end of the Curriculum Elastic Adaptation stage, the sampling ranges are expanded to:

$$R \in [0.5, 4], \quad H \in [6, 12], \quad E \in [384, 768].$$

B.2. Learning Rate and Training Epochs

For all nine image classification datasets, we trained the model for 100 epochs in both Stage 1 and Stage 2. Except for FGVC [35] and Stanford Cars [25], we used a learning rate that decays from 1e-5 to 1e-7 with linear warm-up and cosine decay in both stages. For FGVC [35] and Stanford Cars [25], we adopted a higher initial learning rate: 1e-4 to 1e-6 in Stage 1, and 1e-5 to 1e-7 in Stage 2.

On segmentation benchmarks ADE20K [61], we trained for 25 epochs per stage using a learning rate schedule from 1e-5 to 1e-7 throughout.

For more difficult datasets Imagenet-1K [8], Kvasir [40] and UCMerced [53], we also followed a 100+100 epoch training schedule, using a learning rate from 1e-5 to 1e-7 in both stages.

All learning rates mentioned above refer to the ViT backbone. The learning rate for the router was set to 1,000 times that of the backbone.

For all baselines, we ensured that the total number of training epochs was kept consistent with our method.

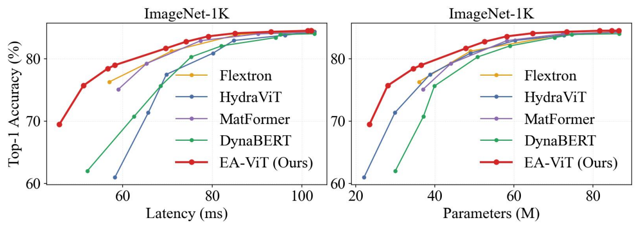


Figure 9. **Comparison of Top-1 accuracy under different latency and parameter count constraints on the ImageNet-1K dataset.** Our method consistently outperforms existing approaches, demonstrating superior trade-offs not only in computation (MACs), but also in latency and model size, which are critical for real-world deployment.

C. More Results

C.1. Comparision wtih other metrics

While the main text focuses on the MACs–accuracy trade-off, we further assess our method using additional metrics, including latency and parameter count. As shown in Figure 9, we compare our model with existing approaches on the ImageNet-1K dataset. The results show that our method consistently achieves higher accuracy under varying latency and parameter constraints. This demonstrates that our approach not only achieves a superior trade-off in terms of computational cost, but also performs competitively across other key deployment-related metrics.

C.2. Comparision under more MACs

As shown in Table 4, we compare our method with previous approaches under five MACs levels: 5, 8, 11, 14, and 17 GMACs. Our method consistently achieves superior performance across most settings, with particularly pronounced advantages under lower computational budgets.

D. Visualization of Submodel Architectures

As shown in Figure 10, we visualize the architectural configurations of submodels on nine datasets under normalized MACs constraints of 0.3 and 0.7. Specifically, we present the per-layer MLP expansion ratios and the number of attention heads in the MHA modules, revealing clear differences across datasets and computational budgets.

In addition, we illustrate how the router's output evolves as the normalized MACs constraint varies in Figure 11. For each dataset, we report the average number of attention heads, MLP expansion ratios, embedding dimensions, and the number of MHA and MLP layers selected by the router. These results provide insight into how the model adjusts architectural complexity in response to resource constraints and dataset characteristics.

E. Potential Applications

Although this work focuses on Vision Transformers and primarily targets image classification and segmentation tasks, the proposed EA-ViT framework can be seamlessly applied to other Transformer-based architectures and extended to broader domains such as image and video editing [32, 33, 49]. Furthermore, incorporating more deployment-efficient datasets [50–52] may further enhance both efficiency and performance in real-world scenarios.

Table 4. **Comparison of Top-1 accuracy across nine classification benchmarks under five GMACs constraints.** The best results are highlighted in **bold**, and the second-best are underlined.

Method	Dataset								
	Cifar 10[26]	Cifar 100 [26]	SVHN[11]	Flowers [38]	Food101 [1]	Aircraft [35]	Cars [25]	DTD [6]	Pets [39]
5 GMACs Budget									
DynaBERT[20]	–	–	–	–	–	–	–	–	–
MatFormer[9]	–	–	–	–	–	–	–	–	–
HydraViT[12]	91.99	73.26	96.26	12.09	73.92	70.66	78.98	21.31	31.36
Flextron[3]	–	–	–	–	–	–	–	–	–
EA-ViT (ours)	**95.81**	**80.49**	**97.49**	**64.85**	**83.32**	**79.06**	**86.91**	**59.99**	**78.95**
8 GMACs Budget									
DynaBERT[20]	94.24	78.30	96.58	24.53	82.24	71.19	80.19	36.88	57.05
MatFormer[9]	96.17	86.18	97.45	69.00	86.31	76.13	88.10	49.25	84.53
HydraViT[12]	96.11	85.39	96.98	29.27	85.49	75.95	85.07	43.53	75.32
Flextron[3]	97.11	85.95	97.61	73.80	87.79	79.36	88.62	61.21	86.35
EA-ViT (ours)	**97.98**	**88.20**	**97.63**	**85.39**	**90.05**	**83.37**	**90.09**	**67.56**	**90.57**
11 GMACs Budget									
DynaBERT[20]	97.53	88.29	96.96	62.46	88.61	74.61	83.47	58.18	83.04
MatFormer[9]	98.08	90.29	97.65	88.00	90.52	78.93	90.49	64.86	91.64
HydraViT[12]	98.08	89.89	97.24	70.46	89.58	77.94	87.18	63.38	89.12
Flextron[3]	98.38	90.34	**97.72**	82.51	90.85	79.86	90.24	69.54	**92.85**
EA-ViT (ours)	**98.53**	**90.64**	97.70	**93.45**	**91.20**	**84.78**	**91.28**	**71.97**	92.49
14 GMACs Budget									
DynaBERT[20]	98.17	91.07	97.42	75.70	90.85	75.93	87.78	65.70	89.85
MatFormer[9]	98.76	92.14	97.77	94.61	92.21	80.11	91.37	71.86	93.37
HydraViT[12]	98.64	91.98	97.62	85.46	91.67	79.05	89.19	69.81	92.29
Flextron[3]	98.71	91.88	97.71	84.98	91.94	80.05	91.09	72.68	**93.87**
EA-ViT (ours)	**98.96**	**92.62**	**97.81**	**96.13**	**92.73**	**85.47**	**91.95**	**74.94**	93.55
17 GMACs Budget									
DynaBERT[20]	98.89	92.95	97.76	83.05	92.77	79.53	90.77	73.05	93.53
MatFormer[9]	99.03	92.91	**97.87**	**96.96**	93.07	80.84	91.76	**75.29**	94.06
HydraViT[12]	98.91	93.13	97.83	90.75	92.98	79.21	89.99	73.72	93.68
Flextron[3]	98.79	92.61	97.77	85.97	93.02	80.52	91.23	73.56	94.16
EA-ViT (ours)	**99.09**	**93.22**	97.84	95.79	**93.13**	**85.48**	**92.17**	74.93	**94.32**

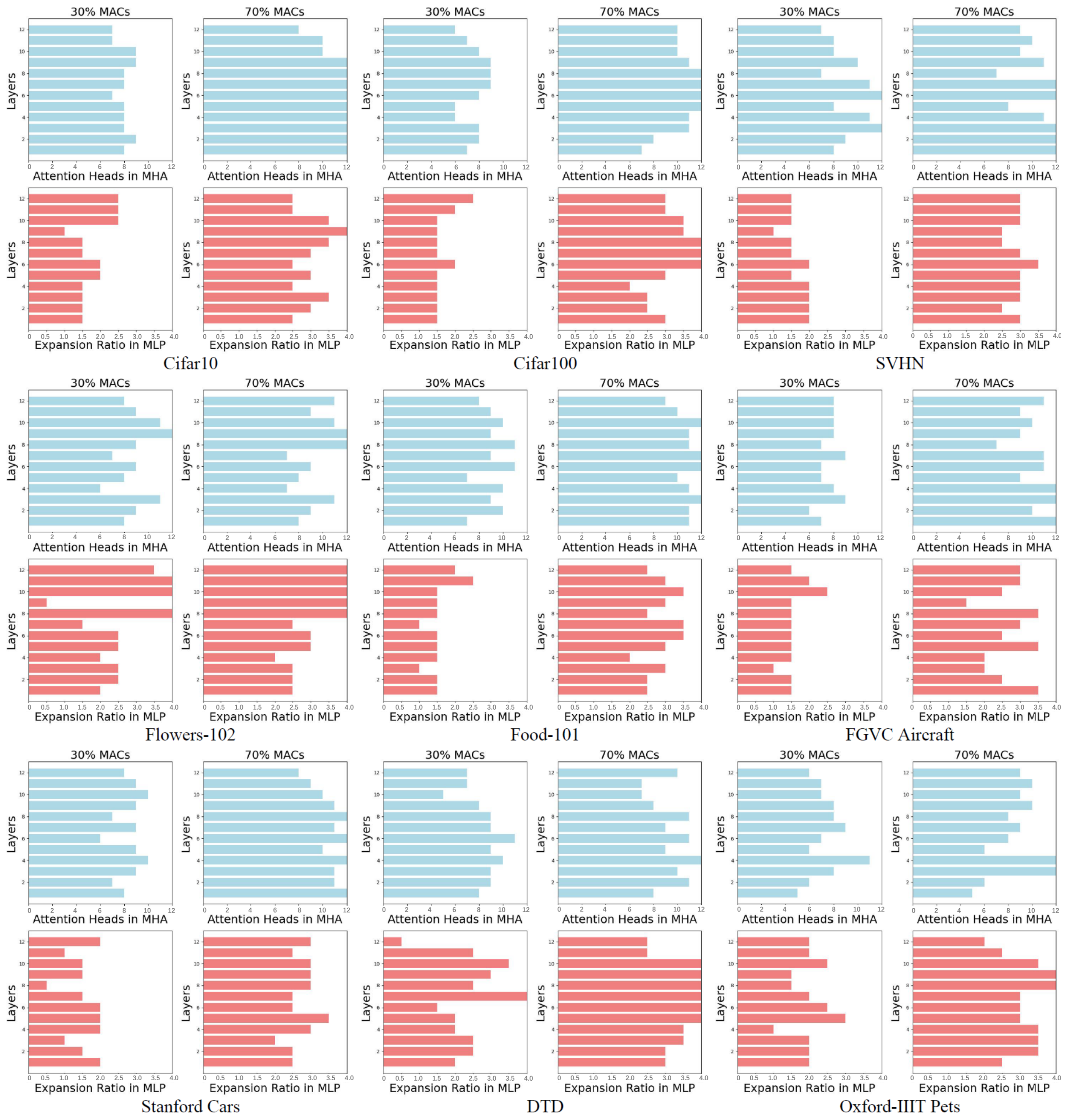


Figure 10. **Layer-wise visualization of submodel architectures under normalized MACs constraints of 0.3 and 0.7 across nine datasets.** For each submodel, we plot the MLP expansion ratio and the number of attention heads in each layer. The comparison reveals distinct architectural patterns that emerge under different computational budgets and dataset characteristics.

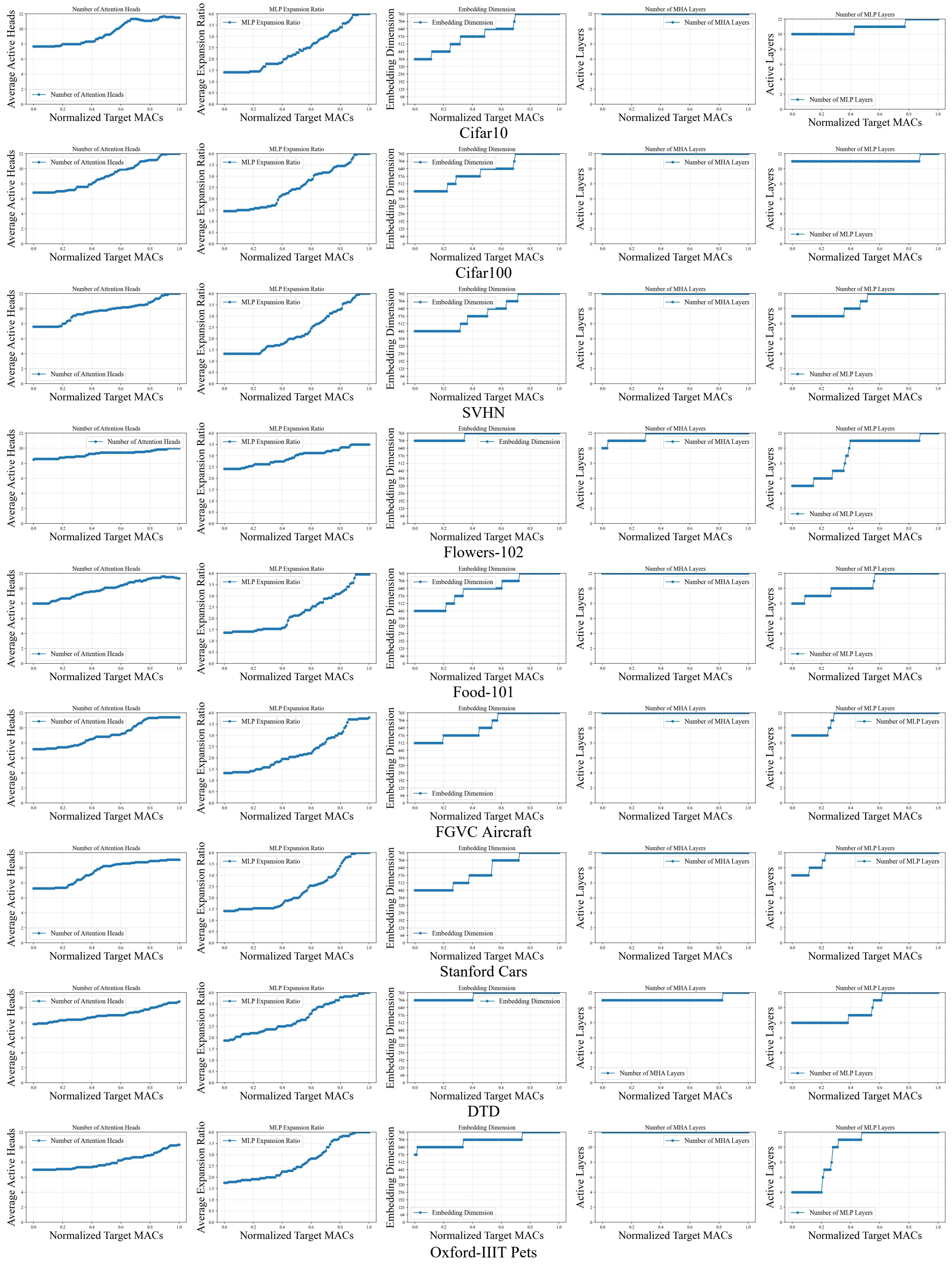


Figure 11. **Trends in architectural configurations selected by the router as the normalized MACs constraint varies.** For each of the nine datasets, we report the average number of MHA attention heads, MLP expansion ratio, embedding dimension, and the number of MHA and MLP layers. The results demonstrate how the router dynamically adapts submodel complexity based on resource constraints and dataset demands.