ORXE: Orchestrating Experts for Dynamically Configurable Efficiency

Supplementary Material

A. Details of the Configuration Searching

A.1. Searching Process

The configuration search is formulated as an optimization problem. The objective function Eq. (2) is not derivativable, but we can still optimize it. Multiple methods can be used. In this paper, we use the procedure described in Algorithm 1. This searching method starts with an empty config, i.e. $\mathbf{t}_{2,\lambda} = \mathbf{1}_{N_{exp}-1}$. It takes multiple rounds to modify the configuration. In each round, it will try to make modifications on every node, but only the best modification is adopted at the end of this round. Each modification attempt means minimizing the objective function w.r.t. a single element of $\mathbf{t}_{2,\lambda}$. Eq. (2) is convex w.r.t. every $\mathbf{t}_{2,\lambda}^{(i)}$, we prove this property in Sec. A.2. Therefore, this minimization can be achieved by any convex optimization method. After searching for the optimal $\mathbf{t}_{2,\lambda}$, we can fix it and search for the optimal $\mathbf{t}_{1,\lambda}$ in the same way. The convergence analysis of $\mathbf{t}_{1,\lambda}$ is more complicated but Eq. (2) is still empirically convex w.r.t. every $\mathbf{t}_{1,\lambda}^{(i)}$. Even if Eq. (2) is not strictly convex and the searching process may not converge to the global minimum, the searching process can work well in practice. Similar to the training process of a deep learning model, converge to the global minimum on the training set is not required and desired.

Algorithm 1: Searching $\mathbf{t}_{2,\lambda}$

Input: The preference factor, $\lambda \in [0, 1]$.
Output: Thresholds for all Gate 2, $\mathbf{t}_{2,\lambda} \in [0, 1]^{N_{exp}-1}$.

Fix $\mathbf{t}_{1,\lambda} = \mathbf{0}_{N_{exp}-1}$, we search the threshold for gate 2 first.
Initialize $\mathbf{t}_{2,\lambda} = \mathbf{1}_{N_{exp}-1}$, i.e. all samples go to the last by default.
Initialize object metric, $f_{min} = f_2(\mathbf{t}_{2,\lambda})$.
while True **do**
  Candidate $\leftarrow$ None
  for $i = 1 \rightarrow N$ **do**
    $\{t_c,\ f_c\} \leftarrow min_{t_{2,\lambda}^{(i)}}\ f_2(\mathbf{t}_{2,\lambda})$
    if $f_c < f_{min}$ **then**
      Candidate $\leftarrow (i,\ t_c)$
      $f_{min} \leftarrow f_c$
  if Candidate *is not* None **then**
    $i,\ t_c \leftarrow$ Candidate
    $t_{2,\lambda}^{(i)} \leftarrow t_c$
  else
    break

A.2. Convergence

When $\mathbf{t}_{1,\lambda}$ is fixed as zero, the objective function is written as:

$$f(\mathbf{t}_{2,\lambda}) = (1-\lambda) \cdot Cost(\mathbf{t}_{2,\lambda}) + \lambda \cdot (1 - Perf(\mathbf{t}_{2,\lambda})) \quad (8)$$

Where $\mathbf{t}_{2,\lambda} \in [0, 1]^N, \lambda \in [0, 1]$ Although $Cost(\mathbf{t}_{2,\lambda})$ and $Perf(\mathbf{t}_{2,\lambda})$ are not actually derivativable, we can still analyze their derivatives. To prove that f has a minimum, we firstly compute the partial derivative of f with respect to $t_{2,\lambda}^{(i)}$ as

$$\frac{\partial f}{\partial t_{2,\lambda}^{(i)}} = (1-\lambda) \cdot \frac{\partial Cost}{\partial t_{2,\lambda}^{(i)}} - \lambda \cdot \frac{\partial Perf}{\partial t_{2,\lambda}^{(i)}} \quad (9)$$

At an extremum, the partial derivative with respect to $t_{2,\lambda}^{(i)}$ must vanish.

$$\frac{\partial f}{\partial t_{2,\lambda}^{(i)}} = (1-\lambda) \cdot \frac{\partial Cost}{\partial t_{2,\lambda}^{(i)}} - \lambda \cdot \frac{\partial Perf}{\partial t_{2,\lambda}^{(i)}} = 0 \quad (10)$$

$$\Rightarrow (1-\lambda) \cdot \frac{\partial Cost}{\partial t_{2,\lambda}^{(i)}} = \lambda \cdot \frac{\partial Perf}{\partial t_{2,\lambda}^{(i)}} \quad (11)$$

$$\Rightarrow \frac{\partial Perf}{\partial Cost} = \frac{1-\lambda}{\lambda} \quad (12)$$

Meanwhile, $\frac{\partial Perf}{\partial Cost}$ means slope of the cost - performance curve. In general, when we add more resource to a system, the performance will increase. However, due to the marginal effect, the increase of performance per cost unit will decrease. i.e.

$$\frac{\partial^2 Perf}{\partial Cost^2} < 0 \quad (13)$$

This phenomenon is also verified by some research on the scaling law of deep learning models. Additionally, $\frac{\partial Perf}{\partial Cost}$ is ∞ when $t_{2,\lambda}^{(i)} = 0$ and approaching 0 when $t_{2,\lambda}^{(i)} \rightarrow \infty$. Therefore, Eq. (12) has only one solution $t_{2,\lambda}^{(i)} > t_{2,\lambda,opt}^{(i)}$, which means f has and only has one global extremum. Moreover, in our proposed system, we assume that every expert is placed in order. Therefore, the later nodes always have higher performance and more cost than the previous ones. For every $t^{(i)} \in \mathbf{t}_{2,\lambda}$, the increase of $t_{2,\lambda}^{(i)}$ will always make more samples processed by later experts. It means more overall cost and better performance, hence:

$$\frac{\partial Cost}{\partial t_{2,\lambda}^{(i)}} > 0, \frac{\partial Perf}{\partial t_{2,\lambda}^{(i)}} > 0 \quad (14)$$

Combining Eqs. (9), (13) and (14), we have:

$$\frac{\partial Perf}{\partial Cost}\begin{cases} < \frac{1-\lambda}{\lambda} \Rightarrow \frac{\partial f}{\partial t_{2,\lambda}^{(i)}} < 0, & t_{2,\lambda}^{(i)} < t_{2,\lambda,opt}^{(i)} \\ = \frac{1-\lambda}{\lambda} \Rightarrow \frac{\partial f}{\partial t_{2,\lambda}^{(i)}} = 0, & t_{2,\lambda}^{(i)} = t_{2,\lambda,opt}^{(i)} \\ > \frac{1-\lambda}{\lambda} \Rightarrow \frac{\partial f}{\partial t_{2,\lambda}^{(i)}} > 0, & t_{2,\lambda}^{(i)} > t_{2,\lambda,opt}^{(i)} \end{cases} \quad (15)$$

$f(t_{2,\lambda}^{(i)})$ is decreasing first and then increasing at around $t_{2,\lambda,opt}^{(i)}$. Therefore, f is convex at every dimension of $\mathbf{t}_{2,\lambda}$, which means that the minimization of f with respect to $\mathbf{t}_{2,\lambda}$ can converge to a global minimum.

B. Implementation Details

The ORXE metamodel consists of 20 models ranging from small to large, selected from about 70 well-trained models from timm[65]. For each selected model, we recorded predictions and confidence scores on the training set. Using the aforementioned method, we generated configurations with regularization coefficients $\alpha = 2.0$, $\beta = 0.2$. The search step was set to 0.01, resulting in 100 configurations, which were then interpolated with a step of 0.001. Configurations that were clearly suboptimal were removed. Hundreds of configurations are available through this process. Tab. B1 presents the complete version of Tab. 1, comparing ORXE with state-of-the-art models. The ORXE metamodel consistently demonstrates superior efficiency.

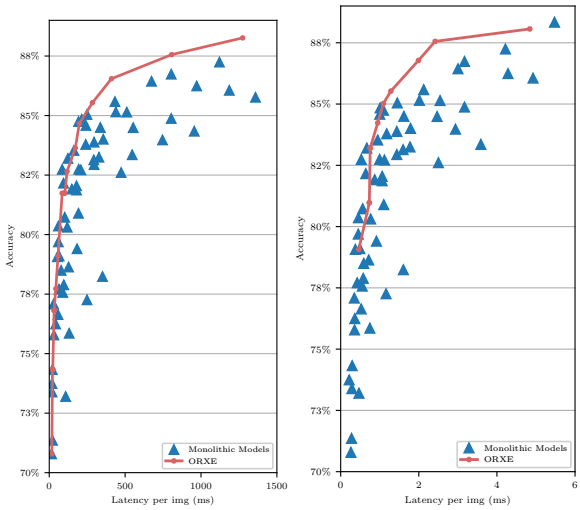
C. Speed Test on More Devices

As a supplement to Sec. 5.3, we further conducted speed tests on more practical scenarios. ORXE can be dynamically configurable while keeping better efficiency in most cases. Fig. C1a shows the results on Raspberry Pi which is a small edge device. The inference latency of the high performance model is reduced by approx. 30% to 50% with the ORXE metamodel. Fig. C1b shows the results on RTX 3090 with a batch size of 128. The batched input results in dynamic batch sizes for the ORXE experts. Modern GPUs support batched processing, allowing the input to utilize more computing units. The proposed method still works in this scenario and have good reduction on the high accuracy model.

D. Energy Consumption Test

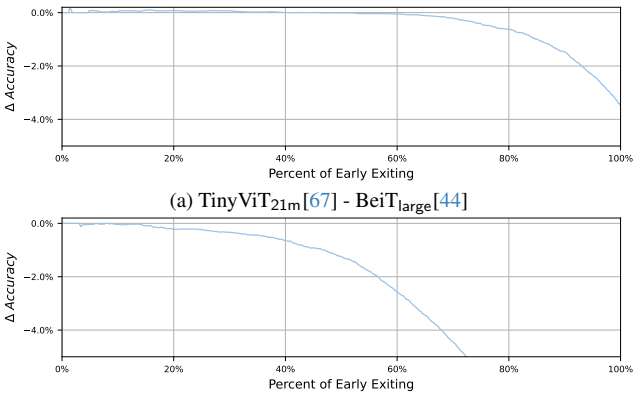
E. Efficiency in Exiting with Predicted Probability

The efficiency gains from orchestrating multiple models depend on the accuracy of the gating mechanism. A potential concern here is the unreliability of model confidence scores. Specifically, a sample that could have been correctly predicted by a larger model might be early exited due to a false positive confidence prediction by a smaller model, thus leading to errors. However, such situations do not occur frequently, especially at high confidence ranges. Fig. E2 illustrates the difference in performance for early-exited samples between smaller and larger models. As shown in Fig. E2a, a smaller model can safely rely on its confidence to select over 50% of the samples without any performance loss compared to the larger model. In other words, the larger model does not provide significantly better predictions for these samples. A more extreme example is combining a very small model with a much larger one, where their FLOPs have 300x differences. Even in this scenario, the smaller model can still effectively handle 40% of the workload, introducing only about 1% additional error. Therefore, gating errors do not impede the overall efficiency gains achievable by the proposed system.



(a) Raspberry Pi, BS 1 (b) RTX 3090, BS 128

Figure C1. The speed test of ORXE and other models on Raspberry Pi and GPU



(a) TinyViT$_{21m}$[67] - BeiT$_{large}$[44]

(b) MobileNetV4$_{conv-m}$[45] - BeiT$_{large}$[44]

Figure E2. Accuracy difference on early exited samples

Table B1. The full table of the FLOPs analysis of ORXE with the comparison of the state-of-the-art models. Models with underscore are employed as members of ORXE.

Model	Acc	FLOPs (G)	Params (M)	timm[65] Checkpoint Name / ORXE Preference
ORXE	**88.35%**	**45.3**	**516.3**	$\lambda = 0.995$
ORXE	**88.35%**	**42.5**	**516.3**	$\lambda = 0.994$
BeiT$_{large}$[44]	88.34%	61.6	304.4	beitv2_large_patch16_224.in1k_ft_in22k_in1k
ORXE	**87.34%**	**10.3**	**815.6**	$\lambda = 0.793$
CAFormer$_{b36}$[73]	87.25%	23.9	211.9	caformer_b36.sail_in22k_ft_in1k
ORXE	**86.91%**	**7.5**	**815.6**	$\lambda = 0.785$
Hiera$_{huge}$[51]	86.90%	127.9	949.0	hiera_huge_224.mae_in1k_ft_in1k
ConvNeXtV2$_{base}$[66]	86.74%	15.4	152.3	convnextv2_base.fcmae_ft_in22k_in1k
CAFormer$_{m36}$[73]	86.44%	14.9	211.9	caformer_m36.sail_in22k_ft_in1k
Swin$_{large}$[40]	86.24%	34.9	265.0	swin_large_patch4_window7_224.ms_in22k_ft_in1k
Hiera$_{large}$[51]	86.06%	45.6	949.0	hiera_large_224.mae_in1k_ft_in1k
ORXE	**85.78%**	**5.1**	**1,064.5**	$\lambda = 0.739$
ConvNeXtV2$_{large}$[66]	85.77%	34.4	342.1	convnextv2_large.fcmae_ft_in1k
CAFormer$_{s36}$[73]	85.59%	10.2	211.9	caformer_s36.sail_in22k_ft_in1k
Hiera$_{baseplus}$[51]	85.15%	14.6	168.6	hiera_base_plus_224.mae_in1k_ft_in1k
Swin$_{base}$[40]	85.14%	16.0	187.7	swin_base_patch4_window7_224.ms_in22k_ft_in1k
Next-ViT$_{base}$[35]	85.05%	8.5	170.6	nextvit_base.bd_ssld_6m_in1k
ORXE	**84.88%**	**3.5**	**704.2**	$\lambda = 0.552$
TinyViT$_{21m}$[67]	84.84%	4.9	96.5	tiny_vit_21m_224.dist_in22k_ft_in1k
MaxViT$_{large}$[56]	84.83%	45.6	352.1	maxvit_large_tf_224.in1k
MaxViT$_{base}$[56]	84.80%	24.9	333.6	maxvit_base_tf_224.in1k
Next-ViT$_{small}$[35]	84.75%	6.0	170.6	nextvit_small.bd_ssld_6m_in1k
HGNet-V2$_{b5}$[8]	84.59%	6.5	170.6	hgnetv2_b5.ssld_stage2_ft_in1k
Hiera$_{base}$[51]	84.50%	9.9	168.6	hiera_base_224.mae_in1k_ft_in1k
DaViT$_{base}$[13]	84.49%	16.5	333.6	davit_base
MaxViT$_{small}$[56]	84.34%	12.9	128.9	maxvit_small_tf_224.in1k
DaViT$_{small}$[13]	84.01%	9.5	464.9	davit_small
XCiT$_{small}$[15]	83.98%	21.1	96.5	xcit_small_12_p8_224.fb_dist_in1k
ORXE	**83.90%**	**2.6**	**639.1**	$\lambda = 0.387$
ConvNeXtV2$_{tiny}$[66]	83.88%	4.5	152.3	convnextv2_tiny.fcmae_ft_in22k_in1k
Hiera$_{small}$[51]	83.79%	8.3	949.0	hiera_small_224.mae_in1k_ft_in1k
EfficientFormerV2$_{l}$[36]	83.53%	2.7	147.0	efficientformerv2_l.snap_dist_in1k
MaxViT$_{tiny}$[56]	83.35%	5.7	150.6	maxvit_tiny_tf_224.in1k
Swin$_{small}$[40]	83.25%	10.8	352.1	swin_small_patch4_window7_224.ms_in22k_ft_in1k
TinyViT$_{11m}$[67]	83.20%	2.2	96.5	tiny_vit_11m_224.dist_in22k_ft_in1k
EfficientViT$_{b3}$[5]	83.14%	4.5	187.7	efficientvit_b3.r224_in1k
ORXE	**82.93%**	**1.7**	**793.5**	$\lambda = 0.289$
Hiera$_{tiny}$[51]	82.74%	6.2	168.6	hiera_tiny_224.mae_in1k_ft_in1k
HGNet-V2$_{b3}$[8]	82.74%	1.8	170.6	hgnetv2_b3.ssld_stage2_ft_in1k
ORXE	**82.74%**	**1.4**	**636.1**	$\lambda = 0.242$
DaViT$_{tiny}$[13]	82.72%	5.2	464.9	davit_tiny
XCiT$_{tiny}$[15]	82.61%	12.3	96.5	xcit_tiny_24_p8_224.fb_dist_in1k
EfficientFormerV2$_{s2}$[36]	82.16%	1.4	187.7	efficientformerv2_s2.snap_dist_in1k
ConvNeXtV2$_{nano}$[66]	82.05%	2.4	152.3	convnextv2_nano.fcmae_ft_in22k_in1k
EfficientViT$_{b2}$[5]	81.91%	1.7	265.0	efficientvit_b2.r224_in1k
ORXE	**81.83%**	**1.2**	**1,092.0**	$\lambda = 0.192$
Swin$_{tiny}$[40]	80.90%	5.8	333.6	swin_tiny_patch4_window7_224.ms_in22k_ft_in1k
ORXE	**80.76%**	**0.6**	**787.5**	$\lambda = 0.142$
TinyViT$_{5m}$[67]	80.73%	1.7	96.5	tiny_vit_5m_224.dist_in22k_ft_in1k
MobileNetV4$_{hybrid-m}$[45]	80.36%	1.2	342.1	mobilenetv4_hybrid_medium.e500_r224_in1k
EfficientFormerV2$_{s1}$[36]	79.69%	0.8	147.0	efficientformerv2_s1.snap_dist_in1k
EfficientNet$_{b4}$[54]	79.41%	1.5	128.9	efficientnet_b4.ra2_in1k
EfficientViT$_{b1}$[5]	79.10%	0.6	187.7	efficientvit_b1.r224_in1k
MobileNetV4$_{conv-m}$[45]	79.07%	0.8	41.4	mobilenetv4_conv_medium.e500_r224_in1k
ORXE	**78.95%**	**0.4**	**257.7**	$\lambda = 0.200$
EfficientNet$_{b3}$[54]	78.64%	1.0	733.0	efficientnet_b3.ra2_in1k
ResNet$_{152}$[24]	78.24%	11.5	464.9	resnet152.tv_in1k
EfficientNet$_{b2}$[54]	77.89%	0.7	187.7	efficientnet_b2.ra_in1k
EfficientNet$_{b0}$[54]	77.71%	0.4	187.7	efficientnet_b0.ra_in1k
EfficientNet$_{b1}$[54]	77.57%	0.6	187.7	efficientnet_b1.ft_in1k
ResNet$_{101}$[24]	77.26%	7.8	333.6	resnet101.tv_in1k
EfficientViT$_{m5}$[38]	77.08%	0.6	75.9	efficientvit_m5.r224_in1k
EfficientFormerV2$_{s0}$[36]	76.25%	0.4	342.1	efficientformerv2_s0.snap_dist_in1k
ResNet$_{50}$[24]	75.86%	4.1	352.1	resnet50.tv_in1k
MobileNetV3$_{large100}$[28]	75.78%	0.2	70.0	mobilenetv3_large_100.ra_in1k
ORXE	**75.77%**	**0.2**	**115.3**	$\lambda = 0.054$
EfficientViT$_{m4}$[38]	74.33%	0.3	75.9	efficientvit_m4.r224_in1k
MobileNetV4$_{conv-s}$[45]	73.75%	0.2	45.4	mobilenetv4_conv_small.e2400_r224_in1k
EfficientViT$_{m3}$[38]	73.39%	0.3	75.9	efficientvit_m3.r224_in1k
ResNet$_{34}$[24]	73.20%	3.7	352.1	resnet34.tv_in1k
EfficientViT$_{b0}$[5]	71.36%	0.1	147.0	efficientvit_b0.r224_in1k
EfficientViT$_{m2}$[38]	70.79%	0.2	75.9	efficientvit_m2.r224_in1k
ResNet$_{18}$[24]	69.54%	1.8	352.1	resnet18.tv_in1k
EfficientViT$_{m1}$[38]	68.32%	0.2	75.9	efficientvit_m1.r224_in1k
MobileNetV3$_{small100}$[28]	67.64%	0.1	34.4	mobilenetv3_small_100.lamb_in1k
EfficientViT$_{m0}$[38]	63.27%	0.1	75.9	efficientvit_m0.r224_in1k

While improving reasoning throughput and reducing latency, the proposed method also achieves a reduction in energy consumption. Tab. D2 presents the experimental results obtained on a Raspberry Pi, where the measured CPU core energy consumption shows a substantial decrease compared with baseline models of comparable accuracy.

Table D2. Energy consumption test on Raspberry Pi 5B

Model	ACC	Largest Expert	Num Experts	in Mem. Params	Latency (ms/img)	Core Energy (mWh/img)
BeiT-L	88.30%	-	1	304.4M	1582	3.00
ORXE	88.20%	BeiT-L	3	668.6M	782	1.42
ORXE	87.00%	BeiT-L	3	668.6M	519	0.91
ConvNextV2-B	86.74%	-	1	152.3M	815	1.32
Hiera-L	86.06%	-	1	939.0M	1242	2.30
ORXE	86.00%	BeiT-L	5	912.1M	245	0.42
MobilenetV4-CM	79.07%	-	1	41.4M	52	0.10
ORXE	79.16%	MobilenetV4-CM	2	117.3M	38	0.07

F. Training Confidence Estimators

For classification models, the output probability naturally serves as an indicator of sample-level confidence. However, many models designed for other tasks do not inherently provide confidence estimates for individual samples. For instance, regression models typically output numerical predictions without accompanying confidence scores. To enable early exiting in models that do not natively provide confidence estimates, an additional evaluator is required. Such an evaluator can be trained to approximate sample-level metrics. For example, in regression tasks, the evaluator could be trained to predict the absolute error of model outputs. Nevertheless, some metrics cannot be directly obtained at the level of individual inputs. An illustrative case is the mean average precision (mAP) metric used in object detection, which is defined over an entire dataset rather than individual samples. Furthermore, these metrics often exhibit non-uniform distributions, and label imbalance can lead to overfitting of the evaluator.

However, confidence measures used for early exiting do not necessarily need to be strictly calibrated against specific performance metrics. The primary goal of the early exit routing is to distinguish samples by the difficulty. We hope to divide the dataset by the score into two parts where the prediction for one subset is better than the other one. Thus, samples with more reliable predictions can exit from further computation. Therefore, the confidence function $Conf(\cdot)$ should ideally have:

$$\forall x_1, x_2, \quad \begin{array}{c} \big(Conf(x_1) > th \wedge Conf(x_2) \leq th\big) \\ \implies Metr(x_1) > Metr(x_2) \end{array} \tag{16}$$

Moreover, the threshold value should be arbitrary while configuring the overall cost and performance. Hence, the actual target of $Conf(\cdot)$ in a configurable early exiting system is ideally to obtain the property of:

$$\forall x_1, x_2, \quad \begin{array}{c} Conf(x_1) > Conf(x_2) \\ \implies Metr(x_1) > Metr(x_2) \end{array} \tag{17}$$

Where $Metr(x)$ represents the actual metric value of model's prediction on x and $Conf(x)$ denotes the estimated confidence value of that prediction.

Therefore, the target of $Conf(\cdot)$, which is expressed in Eq. (17), is ranking samples by $Metr(x_1)$ rather than fitting to it. Now the metric can be any value indicating the relative prediction reliability of a sample, like the negative loss value which is available in every supervised learning task.

Ranking as Calibration Pairwise comparison is a simple but effective method to train neural networks to relative relationship. It asks models to compare arbitrary two samples and learn to predict which one should rank higher. With comprehensive training, the model will be able to produce appropriate score to every sample with a sorted ranking. Concretely, we form $\frac{B(B-1)}{2}$ pairs for a batch with B samples. As described in Eq. (18), each pair between sample predictions i and j is assigned with a label to denote which prediction is better.

$$target_{i,j} = \begin{cases} 1 & \text{if } Metr(x_i) > Metr(x_j) \\ 0.5 & \text{if } Metr(x_i) = Metr(x_j) \\ 0 & \text{if } Metr(x_i) < Metr(x_j) \end{cases} \tag{18}$$

Then the binary cross entropy loss is employed to guide the model to match the relative order between the predicted logit and the metric within a pair. Such a loss function is only sensitive to the relative relationship but not the absolute value of the metric. It simplifies the design of metric value and also resolves the label imbalance issue.

$$\ell = \frac{2}{B(B-1)} \sum_{i=1}^{B} \sum_{j=i}^{B} BCE(Gate(x_i) - Gate(x_j), target_{i,j}) \tag{19}$$

Finally, the used confidence value is derived from the gate output with a sigmoid transformation. The sigmoid function does not change the relative order of its input, but limiting the output to $[0, 1]$ for the convenience of threshold searching.

$$Conf(x) = Sigmoid(Gate(x)) \tag{20}$$

Gates Architecture The purpose of early exiting is to save computation. Therefore, the gate must be implemented in a highly simple form to reduce the overhead to the system. In this work, the gate is designed to be small MLPs. As depicted in Fig. F3, the gate receives inputs from the model backbone and the output. The feature from the backbone is pooled for reducing its shape. The model output, subject to the specific model, may need an embedding layer before entering the MLP. Please refer the experiment section for the specific design of the gate. The output of the MLP is a logit value $Gate(x)$ which is trained to express the ranking for the prediction of the attached expert model on the sample x.

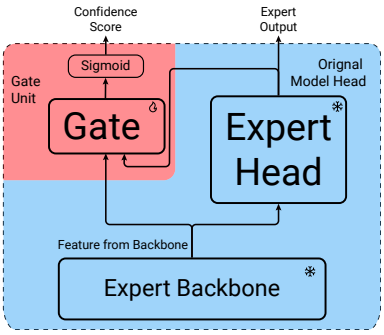
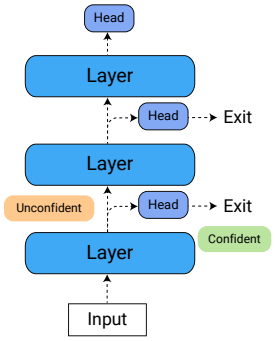


Figure F3. The gate architecture

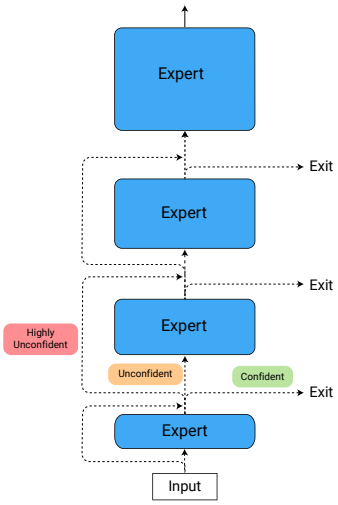
G. Relation to EE and MoE

The proposed ORXE model can be viewed as a special case of early exiting (EE) and top-1 mixture of experts (MoE). However, there are fundamental differences between our work and conventional EE and MoE.

In conventional EE systems, the model is partitioned into multiple sequential segments, each augmented with an auxiliary head for intermediate predictions. Computation must always begin from the first segment, deciding at each head whether to proceed or stop further computation. In contrast, within ORXE, each segment functions as an independent expert model; therefore, computation is not constrained to always start from the initial segment. During routing, ORXE not only supports the standard EE decision-making (continue or exit) but also enables very challenging samples to bypass multiple intermediate experts directly. Consequently, ORXE provides significantly greater flexibility compared to conventional EE methods. The comparative results in Fig. 2b further demonstrate that ORXE achieves higher efficiency than conventional EE approaches. Within a MoE framework, the primary advantage of ORXE lies in leveraging a subset of experts themselves to determine the data routing path rather than a dedicated router. In conventional Top-1 MoE systems, the routing decision is made prior to invoking any experts. Consequently, the router must possess sufficient complexity and capacity to accurately understand the relative strengths of various experts. Such a router is computationally costly, especially in systems designed explicitly for computational efficiency. In contrast, ORXE adopts an approach where experts are selectively used on-demand as implicit routers. This design eliminates the additional computational overhead associated with dedicated routing modules. Furthermore, ORXE can be viewed as employing a form of partial post-expert routing, inherently providing greater accuracy compared to purely pre-expert routing methods.

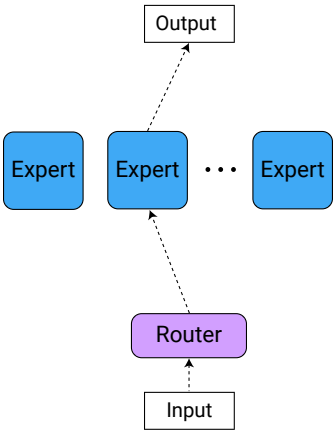


(a) Conventional Early Exit

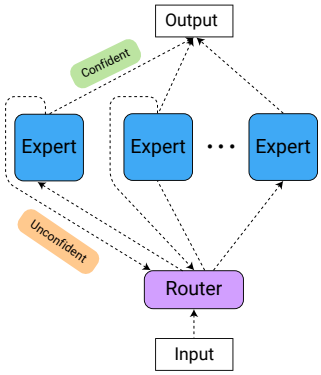


(b) ORXE's equivalence in the context of Early Exit

Figure G4. Differences between ORXE and conventional EE



(a) Conventional Top-1 Mixture of Experts



(b) ORXE's equivalence in the context of Mixture of Experts

Figure G5. Differences between ORXE and conventional Top-1 MoE