Dynamic Multi-Branch Layers for On-Device Neural Machine Translation

Zhixing Tan [ID], Zeyuan Yang, Meng Zhang, Qun Liu [ID], Maosong Sun [ID], and Yang Liu [ID], *Senior Member, IEEE*

Abstract—With the rapid development of artificial intelligence (AI), there is a trend in moving AI applications, such as neural machine translation (NMT), from cloud to mobile devices. Constrained by limited hardware resources and battery, the performance of on-device NMT systems is far from satisfactory. Inspired by conditional computation, we propose to improve the performance of on-device NMT systems with dynamic multi-branch layers. Specifically, we design a layer-wise dynamic multi-branch network with only one branch activated during training and inference. As not all branches are activated during training, we propose shared-private reparameterization to ensure sufficient training for each branch. At almost the same computational cost, our method achieves improvements of up to 1.7 BLEU points on the WMT14 English-German translation task and 1.8 BLEU points on the WMT20 Chinese-English translation task over the Transformer model, respectively. Compared with a strong baseline that also uses multiple branches, the proposed method is up to 1.5 times faster with the same number of parameters.

Index Terms—Conditional computation, decoding, machine translation, natural language processing, transformers.

I. INTRODUCTION

MACHINE translation is a classic problem of artificial intelligence (AI) that aims to translate natural languages automatically. With the rapid development of deep learning, neural machine translation (NMT) [1], [21], [22] has achieved great success and become the dominant approach to MT. Recently, there has been an increasing interest in moving AI applications, such as NMT, from cloud to mobile devices. Compared with cloud-based NMT services, on-device NMT systems offer increased privacy, low latency, and a more compelling user experience.

However, it is challenging to deploy NMT models on edge devices. Due to high computation cost, on-device NMT systems face a trade-off between latency and performance [24]. Fig. 1
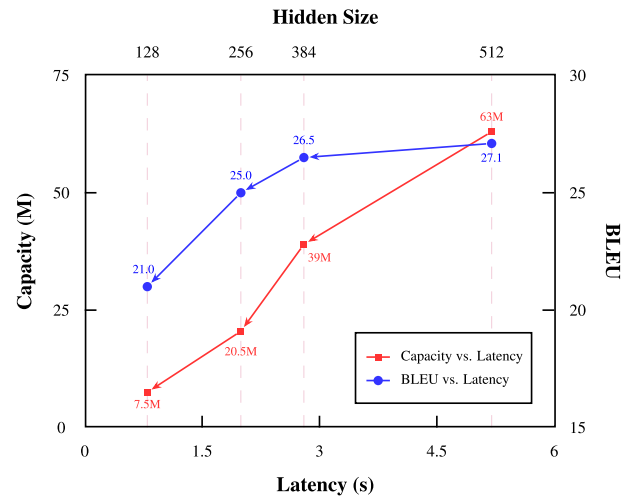
Fig. 1. Latency, performance (BLEU), and capacity of Transformer models with different hidden sizes on a Raspberry Pi 4 device. All models are trained on the WMT14 English-German dataset. Scaling down the hidden size of the model (from 512 to 128) reduces the translation latency, but also lowers the model's capacity and hurts the translation performance.

gives an illustration of the latency, performance, and capacity of Transformer models with different hidden sizes when translating a sequence with 30 tokens on a Raspberry Pi 4 device. As we can see, a Transformer-base model takes over 5 seconds to translate a sequence of 30 tokens and such long latency is not desired for real-time applications. Although the latency can be reduced by simply scaling down the hidden size of the network, it also weakens the model's capacity, making the translation performance of on-device NMT models far from satisfactory. As the model capacity is a key factor in determining the performance of neural networks [19], how to increase the capacity of on-device NMT models without sacrificing efficiency is an important problem for achieving a better trade-off between latency and performance.

This problem has received increasing attention in the community in recent years [2], [3], [7], [19], [24]. Among prior studies, conditional computation [3], [4], [6], which proposes to activate parts of a neural network in an input-dependent fashion, is a representative method for improving model capacity without a proportional increase in computation time. Recently, Shazeer *et al.* [19] have proposed a sparsely-gated mixture-of-experts (MoE) approach. While their approach dramatically increases model capacity by introducing multiple alternatives, it still inevitably incurs non-negligible computational overhead. Bapna *et al.* [2] propose conditional computation Transformer, which
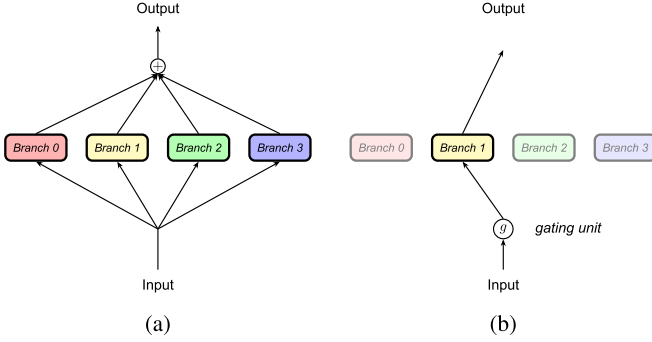
Fig. 2. (a) A conventional multi-branch layer and (b) a dynamic multi-branch layer. By dynamically activating only one branch during training and inference via an input-dependent gate, our goal is to enable on-device NMT to share the same model capacity with conventional multi-branch networks and maintain the same efficiency with standard single-branch networks.

can dynamically skip a sub-layer of the model based on the complexity of input and demonstrates its effectiveness over the vanilla Transformer within a small computational budget on an average basis. However, the computational time may vary significantly when translating sequences with the same length as in the worst case no sub-layer can be skipped, which is not desired for on-device NMT services. As a result, how to use conditional computation to improve model capacity without sacrificing efficiency still remains a major challenge for on-device NMT.

Along the line of conditional computation, we propose to use *dynamic multi-branch* (DMB) layers for on-device NMT. As shown in Fig. 2, a dynamic multi-branch layer is capable of dynamically activating a single branch using an input-sensitive gate, enabling the resulting NMT model to have increased capacity thanks to the use of more branches while keeping the same efficiency with the standard single-branch model. To ensure sufficient training of each branch, we propose shared-private reparameterization that requires zero computational and memory overhead during inference. Based on DMB layers, we also propose Transformer-DMB architecture, which extends both the feed-forward and attention layers of a Transformer model with DMB counterparts, to effectively increase the capacity of the model while keeping nearly the same efficiency.

We conduct experiments on the WMT14 English-German (En-De) and the WMT20 Chinese-English (Zh-En) translation tasks to verify the effectiveness of our proposed method. Experiments show that our method achieves improvements of up to 1.7 BLEU points on the WMT14 En-De translation task and up to 1.8 BLEU points on the WMT20 Zh-En translation task over the Transformer model while keeping nearly the same computation efficiency. The proposed method is also up to 1.5 times faster than a strong MoE-based baseline with the same number of parameters and comparable translation quality.

In summary, our contributions are:

- We propose *dynamic multi-branch layers*, which is a simple yet effective method to increase the capacity of a network without sacrificing speed.
- We propose *shared-private reparameterization* to ensure the effective training of each branch in a DMB layer.

- We propose *Transformer-DMB architecture* that significantly increases the capacity while keeping nearly the same computation cost of the Transformer model.

II. BACKGROUND

A. The Transformer Model

Transformer [22] is the state-of-the-art neural architecture for machine translation. The encoder of the Transformer typically consists of 6 layers, where each encoder layer is composed of a multi-head attention (MHA) sub-layer and a feed-forward (FFN) sub-layer. The decoder of the Transformer also consists of 6 layers, where each decoder layer is composed of a masked MHA sub-layer, an encoder-decoder MHA sub-layer, and an FFN sub-layer.

Multi-head attention consists of H parallel heads, each of which is a scaled dot-product attention. Given a set of n queries $\mathbf{Q} \in \mathbb{R}^{n \times d}$, m keys $\mathbf{K} \in \mathbb{R}^{m \times d}$, and m values $\mathbf{V} \in \mathbb{R}^{m \times d}$, where d is the hidden size, the output of an MHA sub-layer is a combination of the outputs of each head. Formally, it can be described as

$$\text{MHA}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Concat}(\text{head}_1, \ldots, \text{head}_H)\mathbf{W}_o, \quad (1)$$

where $\mathbf{W}_o \in \mathbb{R}^{d \times d}$ are learnable parameters. Each attention head attends to different representation subspaces at different positions:

$$\text{head}_h = \text{Attention}(\mathbf{Q}\mathbf{W}_q^{(h)}, \mathbf{K}\mathbf{W}_k^{(h)}, \mathbf{V}\mathbf{W}_v^{(h)}), \quad (2)$$

where $\mathbf{W}_q^{(h)} \in \mathbb{R}^{d \times d_h}, \mathbf{W}_k^{(h)} \in \mathbb{R}^{d \times d_h}, \mathbf{W}_v^{(h)} \in \mathbb{R}^{d \times d_h}$, and $d_h = d/H$. The computation involved in scaled dot-product attention network can be described as

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}}\right)\mathbf{V}. \quad (3)$$

To compensate the attention's negligence of the order of input sequence, the Transformer adds *positional encoding* to the bottom of both the encoder and the decoder. Please refer to [22] for more details.

The FFN sub-layer consists of two linear layers with ReLU as the activation function. Formally, given an input vector $\mathbf{x} \in \mathbb{R}^d$, the output of an FFN sub-layer can be described as

$$\text{FFN}(\mathbf{x}) = \mathbf{W}_2\text{ReLU}(\mathbf{W}_1\mathbf{x}), \quad (4)$$

where $\mathbf{W}_1 \in \mathbb{R}^{d_f \times d}, \mathbf{W}_2 \in \mathbb{R}^{d \times d_f}$, and d_f is the hidden filter size.

The loss function for learning the Transformer model is the cross-entropy loss. Given a source sequence $\mathbf{x}$ of length S and a target sequence $\mathbf{y}$ of length T, the loss function $\mathcal{L}_m$ is defined as

$$\mathcal{L}_m = -\frac{1}{T}\sum_{t=1}^{T} \log P(\mathbf{y}_t|\mathbf{x}). \quad (5)$$

B. Sparsely-Gated Mixture-of-Experts Layer

Sparsely-gated mixture-of-experts (MoE) layer [19] is a method to realize the promise of conditional computation. Each

MoE layer consists of a set of N experts f_i and a gating network g. Given an input vector $\mathbf{x} \in \mathbb{R}^d$, the output $\mathbf{y}$ of the MoE layer is formally described as follows:

$$\mathbf{y} = \sum_{i=1}^{N} g(\mathbf{x}, i) \cdot f_i(\mathbf{x}), \tag{6}$$

where "·" denotes the scalar-matrix multiplication, and $g(\mathbf{x}, i)$ denotes the i-th element in $g(\mathbf{x})$.

To increase the number of experts N without a proportional increase of computation, Shazeer *et al.* [19] introduced a noisy top-k gating mechanism. Formally, the computation involved in $g(\cdot)$ can be described as

$$g(\mathbf{x}) = \text{softmax}(\text{KeepTopK}(H(\mathbf{x}), k)), \tag{7}$$

$$H(\mathbf{x}) = \mathbf{W}\mathbf{x} + \boldsymbol{\epsilon} \odot \text{softplus}(\mathbf{W}_n\mathbf{x}), \tag{8}$$

where $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{1})$ is a d-dimensional noise vector and "$\odot$" denotes the element-wise product. "softplus" is an activation function which is formally defined as

$$\text{softplus}(x) = \log(1 + \exp(x)). \tag{9}$$

The i-th element in the vector $\text{KeepTopK}(\mathbf{v}, k)$ is formally defined as

$$\text{KeepTopK}(\mathbf{v}, k, i) = \begin{cases} \mathbf{v}_i & \text{if } \mathbf{v}_i \text{ is in the top } k \\ & \text{elements in } \mathbf{v}, \\ -\infty & \text{otherwise.} \end{cases} \tag{10}$$

When setting $k > 1$, gradients can back-propagate through the noise top-k gating network to its inputs. As a result, the MoE layers can be learned in an end-to-end fashion. *However, gradients can no longer back-propagate through the network to its inputs when $k = 1$. The reason is that $g(\mathbf{x})$ is always a constant vector regardless of the input $\mathbf{x}$ in Eq (7).* Therefore, to enable end-to-end training, the MoE layers still introduce non-negligible computational overhead.

Shazeer *et al.* [19] also introduced two additional losses to each MoE layer. The first loss is the diversity loss, which encourages all experts to have equal importance. The second loss is the load balance loss, which ensures balanced loads when training on distributed hardware.

III. PROPOSED METHOD

In this section, we first describe the definition of dynamic multi-branch layers in Section III-A. Then we describe the shared-private reparameterization in Section III-B. Finally, we describe the architecture of our models in Section III-C.

A. Dynamic Multi-Branch Layers

We propose *dynamic multi-branch* (DMB) layers for increasing the model capacity while maintaining roughly the same computational cost. Similar to MoE layers, each DMB layer consists of a set of N identical branches with different parameters. We use d to denote the hidden size of the network. Given an input vector $\mathbf{x} \in \mathbb{R}^d$, we denote the i-th branch in the layer as $f_i(\mathbf{x}; \boldsymbol{\theta}_i)$, where f_i can be an arbitrary differentiable function with parameters $\boldsymbol{\theta}_i$.
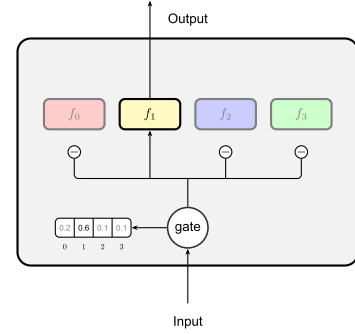

Fig. 3. Overview of a dynamic multi-branch layer with 4 branches. Based on the predictions of the gating unit, only one branch in a layer is active during training and inference.

The output of a DMB layer is described as

$$\mathbf{y} = \sum_{i=1}^{N} g(\mathbf{x}, i) \cdot f_i(\mathbf{x}; \boldsymbol{\theta}_i). \tag{11}$$

Different from MoE layers where $g(\mathbf{x}, i)$ is a real-valued scalar, $g(\mathbf{x}, i)$ in DMB layers is a *binary-valued* scalar indicating whether the i-th branch is activated or not. The value of $g(\mathbf{x}, i)$ is dynamically computed based on the input $\mathbf{x}$. For each DMB layer, we first employ a lightweight gating unit to learn a probability distribution for activating each branch. We use $a(\mathbf{x})$ to denote the predicted distribution and $a_i(\mathbf{x})$ to denote the i-th element in $a(\mathbf{x})$. The gating unit is simply a linear layer followed by a softmax activation function:

$$a(\mathbf{x}) = \text{softmax}(\mathbf{W}_g\mathbf{x} + \mathbf{b}_g), \tag{12}$$

where $\mathbf{W}_g \in \mathbb{R}^{N \times d}$ and $\mathbf{b} \in \mathbb{R}^N$ are learnable parameters. Then we activate the branch with the highest probability. Formally, the value of $g(\mathbf{x}, i)$ is described as

$$g(\mathbf{x}, i) = \begin{cases} 1 & \text{if } a_i(\mathbf{x}) \text{ is the top} \\ & \text{element in } a(\mathbf{x}), \\ 0 & \text{otherwise.} \end{cases} \tag{13}$$

Fig. 3 gives an illustration of a DMB layer. By introducing the gating unit, we can ensure that only one branch in a layer is active during training and inference. Unfortunately, binary outputs of the gating units are not differentiable, which complicates the learning of DMB layers. To enable the end-to-end training of gating units, we introduce two auxiliary losses: the diversity loss and the entropy loss.

The first auxiliary loss is the *diversity loss*. The goal of diversity loss is to encourage a balanced utilization of each branch. Following Shazeer *et al.* [19], we minimize the coefficient of variation among all branches. For a batch of M inputs $\{\mathbf{x}_i\}$, the loss function is formally described as

$$\mu = \frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{M} a_i(\mathbf{x}_j), \tag{14}$$

$$\sigma^2 = \sum_{i=1}^{N} \left(\sum_{j=1}^{M} a_i(\mathbf{x}_j) - \mu \right)^2, \tag{15}$$

$$\mathcal{L}_d = \frac{\sigma^2}{\mu^2}. \tag{16}$$

The second auxiliary loss is the *entropy loss*. We expect that the gating unit can give a high probability when activating a branch. Therefore, we minimize the entropy of predicted distribution $a(\mathbf{x})$. Formally, entropy loss is defined as

$$\mathcal{L}_e = -\frac{1}{M} \sum_{j=1}^{M} \sum_{i=1}^{N} a_i(\mathbf{x}_j) \cdot \log a_i(\mathbf{x}_j). \tag{17}$$

By using the auxiliary losses, the gating units can receive gradients directly from the loss functions. Despite of its simplicity, in our experiments, we found the two auxiliary losses are sufficient for learning the gating units.

Therefore, DMB layers achieve nearly the same computational efficiency as the corresponding single-branch layers. The only additional computational overhead of a DMB layer is calculating $a(\mathbf{x})$, which is negligible compared with the computation of an active branch f_i.

B. Shared-Private Reparameterization

Ideally, we expect a balanced utilization for all branches of a DMB layer. However, in this situation, each branch is only trained with a subset of training examples. We refer to this phenomenon as the *shrinking training examples* problem. For example, for a smaller training set or large choices of N, each branch is only trained with $1/N$ examples in expectation, which often leads to insufficient training of each branch.

To alleviate this problem, we propose a method called *shared-private reparameterization*: for a given DMB layer, the parameters $\boldsymbol{\theta}_i$ of the i-th branch f_i is composed of two separate set of parameters $\{\boldsymbol{\theta}_S, \boldsymbol{\theta}_{P_i}\}$. $\boldsymbol{\theta}_S$ is the *shared parameters* for all branches in the DMB layer. In contrast, $\boldsymbol{\theta}_{P_i}$ is the *private parameters* bound to the i-th branch in the DMB layer. $\boldsymbol{\theta}_i$ is reparameterized as a summation of $\boldsymbol{\theta}_S$ and $\boldsymbol{\theta}_{P_i}$:

$$\boldsymbol{\theta}_i = \boldsymbol{\theta}_S + \boldsymbol{\theta}_{P_i}. \tag{18}$$

Suppose the gating unit chooses the j-th branch in the forward pass. During the backward-pass, the gradients with respect to $\boldsymbol{\theta}_S$ and $\boldsymbol{\theta}_{P_i}$ are

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}_S} = \frac{\partial \mathcal{L}}{\partial \mathbf{y}} \frac{\partial f_j(\mathbf{x}; \boldsymbol{\theta}_j)}{\partial \boldsymbol{\theta}_j}, \tag{19}$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}_{P_i}} = \frac{\partial \mathcal{L}}{\partial \mathbf{y}} \frac{\partial f_j(\mathbf{x}; \boldsymbol{\theta}_j)}{\partial \boldsymbol{\theta}_j} \frac{\partial \boldsymbol{\theta}_j}{\partial \boldsymbol{\theta}_{P_i}}. \tag{20}$$

The shared parameters are always updated no matter which branch is chosen. In contrast, the private parameters $\boldsymbol{\theta}_{P_i}$ receive non-zero gradients only when i is equal to j. By introducing the shared-private reparameterization, we expect that each branch in a DMB layer is not only able to learn shared knowledge with $\boldsymbol{\theta}_S$, but also preserves distinct characteristics via $\boldsymbol{\theta}_{P_i}$.

Another benefit of using a summation in Equation (18) is that we can easily *pre-compute* $\boldsymbol{\theta}_i$ when the training is over. As a result, there is zero computational and memory overhead after training when introducing $\boldsymbol{\theta}_S$.
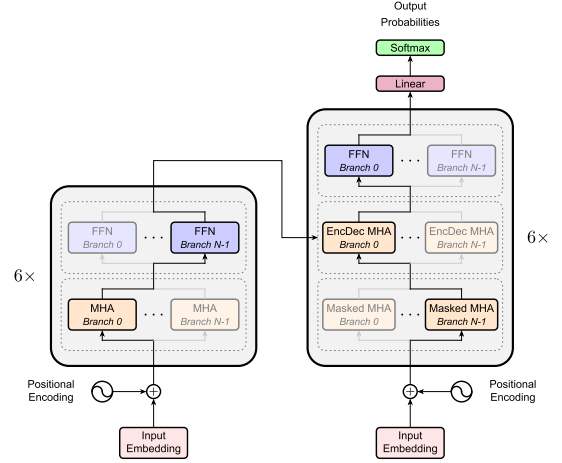

Fig. 4.   Transformer with dynamic multi-branch layers.

C. Architecture and Training Objective

We extend the Transformer architecture with DMB layers, which we refer to as the Transformer-DMB architecture. Fig. 4 depicts the Transformer-DMB architecture. Different from [19] where MoE layers are only applied for FFN layers, we apply DMB at a more fine-grained level. For DMB FFN layers, we introduce a single gating unit g_f. Suppose the gate predictions $g_f(\mathbf{x}, k) = 1$ for a given input $\mathbf{x}$, the computation of DMB FFN layers is formally defined as:

$$\text{FFN}_{\text{DMB}}(\mathbf{x}) = \mathbf{W}_2^{(k)} \text{ReLU}(\mathbf{W}_1^{(k)} \mathbf{x}), \tag{21}$$

where $\mathbf{W}_1^{(k)}, \mathbf{W}_2^{(k)}$ are parameters of the k-the branch of the DMB FFN layer.

For DMB MHA layers, we also introduce an additional gating unit g_a. Given a set of n inputs $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_n]$ and a set of N weights $\{\mathbf{W}^{(k)}\}$, we use $g(\mathbf{X}, \{\mathbf{W}^{(k)}\})$ to denote the following computations:

$$g(\mathbf{X}, \{\mathbf{W}^{(k)}\}) = [\mathbf{W}^{(i_1)}\mathbf{x}_1, \ldots, \mathbf{W}^{(i_n)}\mathbf{x}_n], \tag{22}$$

where $g(\mathbf{x}_k, i_k) = 1$ for $k \in \{1, \ldots, n\}$ and $i_k \in \{1, \ldots, N\}$. Then the computations involved in DMB MHA layers can be formally described as

$$\text{MHA}_{\text{DMB}}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = g_a(\mathbf{H}, \{\mathbf{W}_o^{(k)}\}), \tag{23}$$

where $\mathbf{W}_o^{(k)}$ are weights associated with the k-th MHA branch, $\mathbf{H} = \text{Concat}(\text{head}_1, \ldots, \text{head}_H)$, and

$$\text{head}_h = \text{Attention}(\mathbf{Q}^{(h)}, \mathbf{K}^{(h)}, \mathbf{V}^{(h)}). \tag{24}$$

Furthermore, we have

$$\mathbf{Q}^{(h)} = g_a(\mathbf{Q}, \{\mathbf{W}_q^{(h,k)}\}), \tag{25}$$

$$\mathbf{K}^{(h)} = g_a(\mathbf{K}, \{\mathbf{W}_k^{(h,k)}\}), \tag{26}$$

$$\mathbf{V}^{(h)} = g_a(\mathbf{V}, \{\mathbf{W}_v^{(h,k)}\}), \tag{27}$$

where $\mathbf{W}_q^{(h,k)}, \mathbf{W}_k^{(h,k)}$, and $\mathbf{W}_v^{(h,k)}$ are weights associated with the h-th head of the k-th MHA branch.

Suppose there are a total of L DMB layers. For the i-th gating unit $g(\cdot, i)$, the associated auxiliary losses $\mathcal{L}_{d,i}$ and $\mathcal{L}_{e,i}$ are

added to the original cross-entropy loss $\mathcal{L}_m$ (see Eq. (5)) with a weight α. As a result, the final loss function of our model becomes

$$\mathcal{L} = \mathcal{L}_m + \alpha \frac{1}{L} \sum_{i=1}^{L} (\mathcal{L}_{d,i} + \mathcal{L}_{e,i}). \qquad (28)$$

IV. EXPERIMENTS

A. Setup

1) Datasets: We evaluate our proposed models on English-German (En-De) and Chinese-English (Zh-En) translation tasks. The evaluation metric is case-sensitive BLEU [15]. We use the number of Mult-Adds [1] to measure the computational costs of an NMT model [24]. To quantify how well an on-device NMT system increases capacity while keeping efficiency, we introduce a measure called *performance-time ratio* (PTR) defined as

$$\text{PTR} = \frac{\text{BLEU}}{\sqrt{\#\text{Mult-Adds}}} \times 10^4. \qquad (29)$$

For the En-De translation task, we use the WMT14 training corpus which contains 4.5 M sentence pairs with 103 M English words and 96 M German words. We use a shared source-target vocabulary of about 37,000 tokens encoded by BPE [18]. We use `newstest2013` as the validation set and use `newstest2014` as the test set. We report the tokenized BLEU score as calculated by `multi-bleu.perl` to be in accordance with previous works.

For the Zh-En translation task, we use the training corpus provided by WMT20. After filtering duplicate entries, the corpus consists of 21 M sentence pairs with 417 M Chinese words and 452 M English words. We use 32 K BPE operations to build vocabularies. The validation set is `newsdev2017` and the test set is `newstest2020`. We report the detokenized BLEU score calculated by SACREBLEU [2] [16].

2) Settings: We compare our proposed models with the following baselines:

- *Transformer [22]:* State-of-the-art self-attention based neural machine translation architecture.
- *Transformer-MoE.* We implement the sparsely-gated MoE layer [19] on top of the Transformer architecture. The overall architecture is similar to ours, with two key differences. First, we mix top-2 experts from a total of N in the MoE architecture for end-to-end training of the gating unit. Second, we do not use the shared-private reparameterization in this approach.

Besides these baselines, we also compare our model with recent works on efficient NMT models such as Lite Transformer [24] whenever possible.

We experiment with two commonly used settings for on-device NMT:

- *Tiny setting:* We set the hidden size d of the model to 128. The filter size of feed-forward layers is set to 512. Under

this setting, the Transformer has 229.0 M Mult-Adds for the En-De translation task.
- *Small setting:* A larger setting for more capable devices, which resulting around 623.2 M Mult-Adds for a Transformer model trained on the En-De translation task. We set the hidden size d of the model to 256. The filter size of feed-forward layers is set to 1,024.

For all our models, we use Adam [13] ($\beta_1 = 0.9$, $\beta_2 = 0.98$ and $\epsilon = 1 \times 10^{-9}$) as the optimizer. We empirically set $\alpha = 0.1$ for our MoE and DMB approaches, and use $N = 4$ experts or branches. For DMB layers, we initialize all shared parameters to zero at the beginning of training. Each mini-batch contains approximately 64 K source and 64 K target tokens. All other settings are the same with Vaswani *et al.* [22]. We train all models on a machine with 8 RTX 2080Ti GPUs for 10,000 steps and save a checkpoint every 1,000 steps. For all models, we averaged the last five checkpoints to obtain a single model. Decoding is performed using beam search. Following [22], we set the beam size to 4 in all our experiments. We set the length penalty to 0.6 for En-De translation and 1.0 for Zh-En translation.

B. Results

Table I shows the results on the En-De and the Zh-En translation tasks, respectively.

On the En-De translation task, our model outperforms the Transformer model by 1.7 BLEU points under tiny model settings, and 0.7 BLEU points under small model settings. The results of the Zh-En translation task are similar. Our model outperforms the Transformer model over 1.8 BLEU points under tiny model settings, and 1.0 BLEU point under small model settings. The performance of our model is on par with the Transformer model extended with MoE. The Transformer-DMB and Transformer-MoE models also have the same number of parameters. However, the computational overhead of our model is much less than the MoE approach. Our approach only adds 0.6 M computational overhead for the tiny model compared to the corresponding Transformer architecture whereas the MoE approach adds 84.2 M Mult-Adds. Under the small model settings, our Transformer-DMB model is 1.5 times faster than the Transformer-MoE model (624.3 M Mult-Adds vs. 956.6 M Mult-Adds), which is, therefore, more preferable for edge devices. We also compare our Transformer-DMB models with other models, such as Evolved Transformer [20], Lite Transformer [24], and HAT [23]. With comparable performance, our models achieve the best performance-time ratio. Nevertheless, we believe our work is also applicable to Lite Transformer and HAT.

We report the inference latency of a sequence with 30 tokens in Table II on a Raspberry 4 device, which has 4 Cortex-A72 cores at 1.5 GHz and 8 GB RAM. For greedy search, the Transformer-DMB model costs about 7.7% more time than the Transformer model under the tiny model settings and costs about 10.9% more time under the small model settings. Compared with Transformer-MoE models, the Transformer-DMB model is about 1.4 times faster under the tiny model settings, and 1.5 times faster under the small model settings. For beam search,

---

[1] We use the `torchprofile` library to count the number of Mult-Adds.
[2] Signature: BLEU+case.mixed+numrefs.1+smooth.exp+tok.13a+version. 1.4.14.

TABLE I
EXPERIMENTS ON WMT14 EN-DE AND WMT20 ZH-EN DATASETS

Method	En-De				Zh-En			
	#Params.	#MA.	BLEU	PTR	#Params.	#MA.	BLEU	PTR
Transformer (tiny) [22]	7.5M	229.0M	21.0	13.9	11.0M	209.7M	19.0	13.2
Evolved Transformer [20]	15.9M	-	22.0	-	-	-	-	-
Lite Transformer [24]	16.0M	286.6M	22.6	13.3	-	-	-	-
Transformer-MoE (tiny)	15.8M	313.2M	22.5	12.7	19.3M	293.9M	21.0	12.2
Transformer-DMB (tiny)	15.8M	229.6M	22.7	**15.0**	19.3M	210.3M	20.8	**14.3**
Transformer (small) [22]	20.5M	623.2M	25.0	10.0	27.4M	584.6M	24.3	10.1
Lite Transformer [24]	38.5M	761.1M	25.6	9.3	-	-	-	-
HAT [23]	80.0M	2.1G	25.9	5.7	-	-	-	-
Transformer-MoE (small)	53.7M	956.6M	25.7	8.3	60.6M	918.1M	25.0	8.3
Transformer-DMB (small)	53.7M	624.3M	25.7	**10.3**	60.6M	585.7M	24.8	**10.2**

"#Params." denotes the number of parameters, "#MA." denotes the number of multi-adds, and "PTR" denotes the performance-time ratio. different from wu *et al.* [24] and wang *et al.* [23], we also count the number of parameters and mult-adds in the embedding and classification layers. "-" indicates that the comparison is not applicable. the number of experts/branches N is set to 4 for MoE and DMB.

TABLE II
RASPBERRY PI 4 ARM CPU LATENCY AND BLEU COMPARISONS WITH DIFFERENT MODELS ON THE WMT14 EN-DE TRANSLATION TASK. THE NUMBER OF EXPERTS/BRANCHES N IS SET TO 4 FOR MOE AND DMB

Model	Latency	BLEU
greedy search		
Transformer (tiny) [22]	0.39s	20.0
Transformer-MoE (tiny)	0.60s	21.5
Transformer-DMB (tiny)	0.42s	21.8
Transformer (small) [22]	1.37s	24.0
Transformer-MoE (small)	2.26s	24.6
Transformer-DMB (small)	1.52s	24.6
beam search		
Transformer (tiny) [22]	0.82s	21.0
Transformer-MoE (tiny)	1.33s	22.5
Transformer-DMB (tiny)	0.99s	22.7
Transformer (small) [22]	1.95s	25.0
Transformer-MoE (small)	4.15s	25.7
Transformer-DMB (small)	2.72s	25.7

TABLE III
ABLATION STUDY OF OUR PROPOSED TRANSFORMER-DMB MODEL ON THE WMT14 EN-DE TRANSLATION TASK

Method	BLEU
Transformer-DMB (tiny)	22.7
— MHA DMB Layers	22.0
— Shared-Private Reparameterization	21.9
— Diversity Loss	21.0
— Entropy Loss	21.1
— Auxiliary Losses	21.1

can further increase the capacity of the model, and is helpful for improving the translation performance.

- Without using shared-private reparameterization, the Transformer-DMB model achieves 21.9 BLEU points, which underperforms the result of a model trained with shared-private reparameterization by 0.8 points. This shows that shared-private reparameterization is helpful in improving the learning of each branch in DMB layers.

- Without introducing auxiliary losses (diversity loss or entropy loss or both) during training, the Transformer-DMB model only achieves around 21.1 BLEU points, which is nearly the same as a Transformer model with the same hidden size (21.0 BLEU points). The results indicate that both diversity and entropy loss are essential for learning DMB layers.

Transformer-DMB modes introduce more computational burdens to Transformer models compared with the results when using greedy search. This is because our current implementation splits a batched matrix into smaller matrices to enable conditional computations, which reduces the degree of parallelism for matrix multiplications. We believe the gap can be significantly narrowed with a dedicated DMB-aware matrix multiplication kernel.

C. Ablation Study

We conduct an ablation study on the WMT14 En-De translation task to investigate the performance of each component of our model, the results are reported in Table III . Based on the results, we have the following observations:

- Replacing all DMB MHA layers in the Transformer-DMB model with plain MHA layers lowers the BLEU score by 0.7 points. The results suggest that besides extending FFN layers with DMB FFN layers, extending MHA layers on both the encoder and the decoder with DMB MHA layers

D. Effect of Reparameterzation and the Number of Branches

We investigate the effect of the number of branches and the effectiveness of the shared-private reparameterization on the WMT14 En-De translation task. Fig. 5 shows the results with different settings. We have two observations:

- The number of branches is a key factor that affects the performance of our Transformer-DMB models. The performance of our methods improves as the number of branches increases. This validates our assumption that the model capacity is a key bottleneck for current on-device NMT models. Our methods can significantly increase the model

TABLE IV
EXPERIMENTS ON WMT14 EN-DE AND WMT20 ZH-EN DATASETS *With Knowledge Distillation*

Method	En-De				Zh-En			
	#Params.	*#MA.*	*BLEU*	*PTR*	*#Params.*	*#MA.*	*BLEU*	*PTR*
Transformer (big) [22]	214M	6.6G	28.7	3.5	274M	6.3G	28.4	3.6
Transformer (tiny) [22]	7.5M	229.0M	23.0	15.2	11.0M	209.7M	21.7	15.0
Transformer-MoE (tiny)	15.8M	313.2M	24.9	14.1	19.3M	293.9M	23.3	13.6
Transformer-DMB (tiny)	15.8M	229.6M	24.8	**16.4**	19.3M	210.3M	23.4	**16.1**
Transformer (small) [22]	20.5M	623.2M	26.7	10.7	27.4M	584.6M	26.4	10.9
Transformer-MoE (small)	53.7M	956.6M	27.1	8.8	60.6M	918.1M	27.2	9.0
Transformer-DMB (small)	53.7M	624.3M	27.7	**11.1**	60.6M	585.7M	26.9	**11.1**

"transformer (big)" is used as the teacher model in knowledge distillation. "#params." denotes the number of parameters, "#MA." denotes the number of multi-adds, and "PTR" denotes the performance-time ratio. the number of branches N is set to 4 for MoE and DMB.
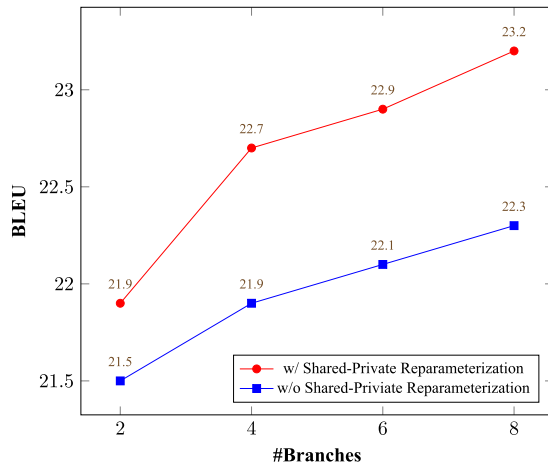

Fig. 5. Effect of branch number and shared-private reparameterization on the WMT14 En-De dataset.

capacity while introducing negligible computational overhead.

- The shared-private reparameterization is very effective. Under all settings, we can see that the model with shared-private reparameterization outperforms the corresponding counterpart. We can see that the gap between models with/without shared-private reparameterization becomes larger as N increases, which proves that shared-private reparameterization can effectively address the shrinking training examples problem.

E. Effect of Knowledge Distillation and Quantization

As knowledge distillation (KD) [12] is a commonly used technique to improve the performance of small models, we conduct additional experiments to investigate the effectiveness of our method on the data generated with sequence-level knowledge distillation.

On the En-De translation task, our model outperforms the Transformer model over 1.8 BLEU points under tiny model settings, and 1.0 BLEU point under small model settings. The results of the Zh-En translation task are similar. Our model outperforms the Transformer model over 1.7 BLEU points under tiny model settings, and 0.5 BLEU points under small model settings. The performance of our model is also on par with the Transformer model extended with MoE. The results suggest that our method is still effective when combined with knowledge distillation.

One potential concern of the DMB approach is the increase of memory requirements. Fortunately, this problem can be alleviated with quantization techniques [7]. Quantization is a very effective approach to reduce the memory consumption of NMT models, and it also improves the translation speed by utilizing integer instructions instead of floating-point operations. We show our model is orthogonal with quantization techniques. We further conduct experiments with a tiny Transformer-DMB model on the WMT14 En-De dataset with 8-bit quantization. The model has 8 branches and is trained on the data generated by knowledge distillation. The BLEU score of the quantized model is 25.3, which is the same as the model without quantization.

Fig. 6 shows the BLEU, #Mult-Adds and disk spaces of models with DMB, knowledge distillation, and quantization. Combining all techniques, we achieve 25.3 BLEU points with only 230.1 M Mult-Adds. The model trained with 8-bit quantization only occupies 26.9 MB of space, which we believe is affordable for most edge devices. By using even lower bit quantization techniques [7], we believe our methods can achieve a better space-time tradeoff and further improve the model performance.

F. Analysis

1) Size of Training Dataset: It is well-known that increased capacity is helpful for neural models to absorb more training data [19]. Therefore, we conduct experiments to study the comparisons between Transformer-DMB models (tiny) and Transformer models (tiny) at different data sizes. Fig. 7 shows the results. To our surprise, we find that Transformer-DMB models always outperform Transformer models regardless of the input data size. The results indicate that increasing the capacity of the model not only helps learn from more data but also is helpful for learning expressive representations.

2) Sentence Length: As shown in Fig. 8, it is clear that the Transformer-DMB model (tiny) is much better than the Transformer model (tiny) at translating long sentences. We believe this is due to the Transformer-DMB models can use different branches to learn different semantic features of the
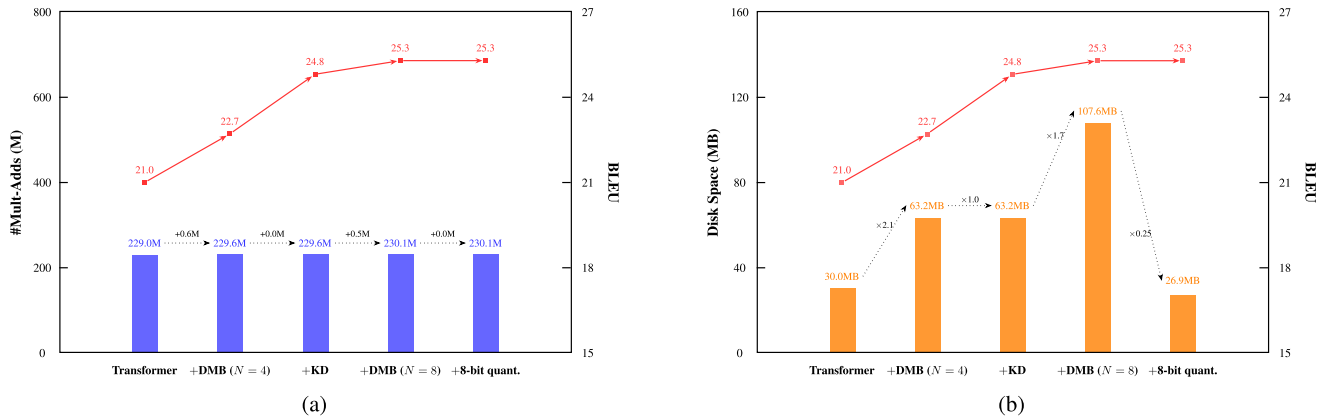
Fig. 6. (a) #Mult-Adds vs. BLEU and (b) Disk space vs. BLEU on the WMT14 En-De translation tasks. Combined with DMB, knowledge distillation, and 8-bit quantization, we are able to achieve an improvement of 4.3 BLEU points over the Transformer model with nearly the same number of Mult-Adds and disk spaces. "quant." indicates quantization.
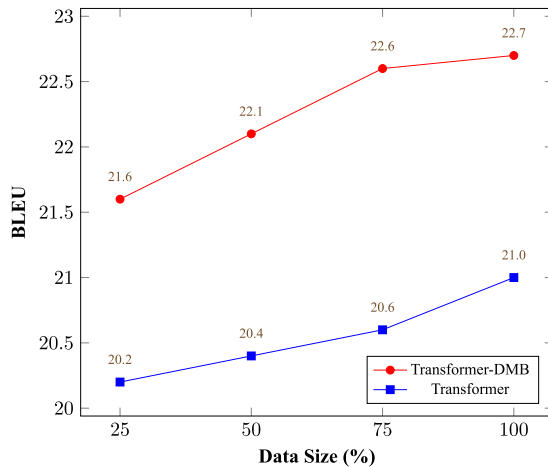

Fig. 7. Comparisons between Transformer-DMB models and Transformer models on the WMT14 En-De translation task with different data sizes.
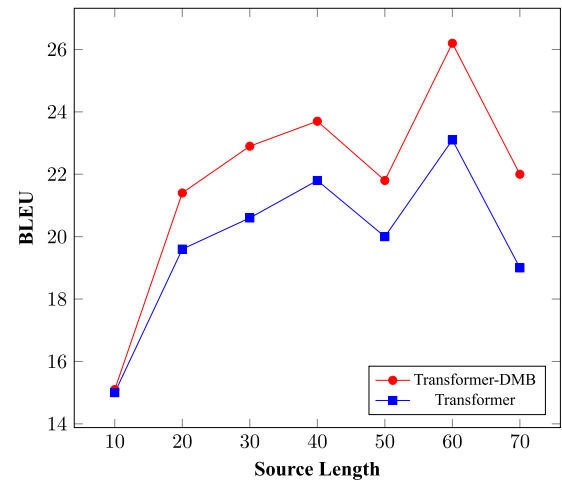

Fig. 8. BLEU scores of the generated translations on the `newstest2014` with respect to the lengths of the source sentences.

input sentence, which is more powerful and flexible than the single-branch Transformer models.

3) Visualization: We visualize the gate predictions to find out how DMB layers process sequences. Fig. 9 shows visualizations of 5 sentences. The predictions are based on the gating unit at the 4th attention sub-layer of the encoder. As we can see, different branches typically process different types of words. For example, pronouns and articles are classified into the branch highlighted in purple, named entities usually processed by the green branch. The results confirm that different branches can learn different semantic representations, thus improving the performance of NMT models.

V. RELATED WORK

Efficient NMT: There are many works devoted to efficient NMTs, which can be roughly divided into two categories. The first line is related to model compression. Pruning [11], [25] and quantization [7], [17] are two widely used methods to reduce the model size. We believe our work is orthogonal with these works. The second line focuses on reducing the computations of NMT

Fig. 9. Visualization of gate predictions on the En-De translation task. Different colors represent different branches.

models. Wu *et al.* [24] proposed Lite Transformer for mobile applications. They introduced a Long-Short Range Attention network to reduce the computational cost of the Transformer model. Lu *et al.* [14] proposed the multi-head attention Res-Block and the position-wise feed-forward network ResBlock to speed up the Transformer model. So *et al.* [20] and [23] introduced architecture search methods to find a hardware-aware architecture. Compared with these works, our work does not need additional search time. Besides, we believe our methods are easily applicable to these new architectures.

b) Conditional Computation: Bengio *et al.* [5] first proposes the concept of conditional computation to enhance the capacity of neural networks without incurring additional computations.

Various forms of conditional computation have been proposed since then [3], [4], [6], [8], [9]. Shazeer *et al.* [19] introduce a sparsely-gated mixture-of-experts approach. They build an NMT model with thousands of feed-forward neural networks between two recurrent layers. The MoE layer is similar to ours but has two major differences. First, instead of a mixture of top-k experts, we only choose one branch in a layer. Second, we introduce shared-private reparameterization in each layer during training. Bapna *et al.* [2] propose conditional computation transformer (CCT) architecture. They add a budget loss to control the balance between quality and computation. Our work is different from theirs in several aspects. First, while they adopt an independent gating approach, we use a dynamic multi-branch approach. Second, our architecture is different from the conditional computation Transformer. Third, given a fixed sequence length, the computational costs of our models are fixed, whereas CCT depends on the complexity of the input sequence. Very recently, Fedus *et al.* [10] propose Switch Transformer architecture for scaling to trillion parameter models, which is concurrent with our work. They introduce a top-1 MoE layer which reduces the computation required in the previous approach [19]. The differences between our DMB layers and their top-1 MoE layers are two folds: First, our gating decisions are binary-valued whereas their gating outputs are real-valued. Our method requires less computation because we do not need to multiply the gating output with the branch output. Second, we use shared-private reparameterization during training whereas they do not have a similar approach. Besides, Switch Transformer focuses on scaling up large models, while we show the effectiveness of our approach for resource-constrained applications.

VI. Conclusion

We have proposed to use dynamic multi-branch layers to improve performance without sacrificing efficiency for on-device neural machine translation. This can be done by dynamically activating a single branch during training and inference. We also propose shared-private reparameterization for sufficient training of each branch. Experiments show that our approach achieves higher performance-time ratios than state-of-the-art approaches to on-device NMT.

References

[1] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," in *Proc. Int. Conf. Learn. Representations*, 2015.

[2] A. Bapna, N. Arivazhagan, and O. Firat, "Controlling computation versus quality for neural sequence models," 2020, *arXiv:2002.07106.*

[3] Y. Bengio, N. Léonard, and A. Courville, "Estimating or propagating gradients through stochastic neurons for conditional computation," 2013, *arXiv:1308.3432.*

[4] E. Bengio, P.-L. Bacon, J. Pineau, and D. Precup, "Conditional computation in neural networks for faster models," in *Proc. Conf. Learn. Representations, Workshop Track.*, 2016.

[5] Y. Bengio, "Deep learning of representations: Looking forward," in *Proc. Int. Conf. Stat. Lang. Speech Process.*, 2013, pp. 1–37.

[6] K. Cho and Y. Bengio, "Exponentially increasing the capacity-to-computation ratio for conditional computation in deep learning," 2014, *arXiv:1406.7362.*

[7] I. Chung *et al.*, "Extremely low bit transformer quantization for on-device neural machine translation," in *Proc. Empirical Methods Natural Lang. Process., Findings*, 2020, pp. 4812–4826.

[8] A. Davis and I. Arel, "Low-rank approximations for conditional feed-forward computation in deep neural networks," in *Proc. Conf. Learn. Representations, Workshop Track.*, 2014.

[9] D. Eigen, M. Ranzato, and I. Sutskever, "Learning factored representations in a deep mixture of experts," in *Proc. Conf. Learn. Representations, Workshop Track.*, 2014.

[10] W. Fedus, B. Zoph, and N. Shazeer, "Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity," 2021, *arXiv:2101.03961.*

[11] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," in *Proc. Int. Conf. Learn. Representations*, 2019.

[12] Y. Kim and A. M. Rush, "Sequence-level knowledge distillation," in *Proc. Empirical Methods Natural Lang. Process.*, pp. 1317–1327.

[13] P. D.Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. Int. Conf. Learn. Representations.*, 2015.

[14] S. Lu, M. Wang, S. Liang, J. Lin, and Z. Wang. "Hardware accelerator for multi-head attention and position-wise feed-forward in the transformer," *IEEE 33rd Int. Syst.-on-Chip Conf.*, 2020, pp. 84–89.

[15] K. Papineni, S. Roukos, T. Ward, and W. Zhu, "BLEU: A method for automatic evaluation of machine translation," in *Proc. Assoc. Comput. Linguistics*, 2002, pp. 311–318.

[16] M. Post, "A call for clarity in reporting BLEU scores," in *Proc. 3rd Conf. Mach. Transl.*, 2018, pp. 186–191.

[17] G. Prato, E. Charlaix, and M. Rezagholizadeh, "Fully quantized transformer for machine translation," in *Proc. Empirical Methods Natural Lang. Process., Findings*, 2020, pp. 1–14.

[18] R. Sennrich, B. Haddow, and A. Birch, "Neural machine translation of rare words with subword units," in *Proc. Assoc. Comput. Linguistics*, 2016, pp. 1715–1725.

[19] N. Shazeer *et al.*, "Outrageously large neural networks: The sparsely-gated mixture-of-experts layer," in *Proc. Int. Conf. Learn. Representations*, 2017.

[20] D. R. So, C. Liang, and Q. V. Le, "The evolved transformer," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 5877–5886.

[21] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Proc. Neural Inf. Process. Syst.*, 2014, pp. 496–527.

[22] A. Vaswani *et al.*, "Attention is all you need," in *Proc. Neural Inf. Process. Syst.*, 2017, pp. 5998–6008.

[23] H. Wang *et al.*, "HAT: Hardware-aware transformers for efficient natural language processing," in *Proc. Assoc. Comput. Linguistics*, 2020, pp. 7675–7688.

[24] Z. Wu, Z. Liu, J. Lin, Y. Lin, and S. Han, "Lite transformer with long-short range attention," in *Proc. Int. Conf. Learn. Representations*, 2020.

[25] M. Zhu and S. Gupta, "To prune, or not to prune: Exploring the efficacy of pruning for model compression," in *Proc. Conf. Learn. Representations, Workshop Track.*, 2018.

Zhixing Tan received the Ph.D. degree from the Department of Artificial Intelligence, Xiamen University, Xiamen, China, in 2018. He is currently a Postdoctoral Researcher with Tsinghua University, Beijing, China. His research interests include machine translation and natural language processing.

Zeyuan Yang is currently working toward the master's degree with the Department of Computer Science and Technology, Tsinghua University, Beijing, China. His research interests focuses on continual learning and its application to natural language processing.

Meng Zhang received the Ph.D. degree from the Department of Computer Science and Technology, Tsinghua University, Beijing, China, in 2018. He is currently a Researcher with Huawei Noah's Ark Lab. His research interests include the intersection of machine learning and natural language processing.

Maosong Sun received the Ph.D. degree in computational linguistics from the City University of Hong Kong, Hong Kong, in 2004. He is currently a Professor with the Department of Computer Science and Technology, Tsinghua University, Beijing, China. His research interests include natural language processing, web intelligence, and machine learning.

Qun Liu received the Ph.D. degree from the Institute of Computational Linguistics, Peking University, Beijing, China, in 2004. He is currently the Chief Scientist of speech and language computing with Huawei Noah's Ark Lab. His research interests include computational linguistics and natural language processing.

Yang Liu (Senior Member, IEEE) received the Ph.D. degree from the Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China, in 2007. He is currently a tenured Full Professor with the Department of Computer Science and Technology, Tsinghua University, Beijing, China. His research interests include natural language processing and machine translation.