Enhancing Branchformer with Dynamic Branch Merging Module for Code-Switching Speech Recognition

Hong-Jie Hu*, Yu-Chiao Lai† and Chia-Ping Chen‡
* National Sun Yat-sen University, Taiwan
E-mail: jie12504469@gmail.com
† National Sun Yat-sen University, Taiwan
E-mail: joelai0101@gmail.com
‡ National Sun Yat-sen University, Taiwan
E-mail: cpchen@cse.nsysu.edu.tw

***Abstract*—Branchformer combines the self-attention and convolutional gating multi-layer perceptron blocks as two branches to capture global and local features. To merge both features, Branchformer applies concatenation and weighted average methods. In this work, extending from Branchformer, we propose DBM-Branchformer in a novel way for the combination of both features. DBM-Branchformer employs Dynamic Branch Merging modules (DBMs) to merge both branches. These modules fuse two branches with dynamic weights for a more flexible combination than Branchformer. On TALCS, the code-switching dataset, DBM-Branchformer obtains a word error rate of 6.9%, outperforming the Branchformer with two methods by 0.1% and 0.2%.**

I. Introduction

Nowadays, we can see many applications of ASR in our surroundings, like Google Maps, SIRI, and the auto-generated subtitles on YouTube. The main idea of ASR is to translate the audio into the corresponding text. Learning from the datasets, the model can perform stochastic probability calculation or neural model decoding to decide the possible sentence with the acoustic features.

In commercial or academic contexts, speakers sometimes resort to code-switching when it is hard to translate some specialized words into their mother tongue. Although monolingual ASR systems are mature for various applications, code-switching ASR systems remain challenging. It is difficult for the model to identify the phonetic boundaries when different languages have similar phonemes. The model may output the wrong words with the incorrect language but has similar pronunciation.

In recent decades, end-to-end models have become popular in ASR due to their impressive performance, such as Convolutional Neural Networks (CNNs) and Transformers [1]. The main idea of CNNs is to apply convolution on the input sequences with multiple kernels, extracting the local features with several sliding windows. By contrast, the major part of the Transformer is the self-attention layer, which can extract long-range dependencies of the sequence according to the global relationship.

Recently, many studies [2]–[7] indicate that fusing self-attention and CNN will gain more improvement. Conformer [2] and Branchformer [3], both Transformer-based systems, use the self-attention layer for global features and convolution layers for local features. Conformer, known as the state-of-the-art structure for ASR tasks, stacks the self-attention and convolution layers in series. It considers the local and global features equally. The static single-branch architecture of Conformer makes it hard to explain and modify the usage of local and global relationships across the encoder layer. To solve these problems, Yifan Peng et al. [3] propose Branchformer that has comparable performance to Conformer. It connects the self-attention and convolution layers in parallel, making the model more flexible in deciding whether to pay more attention to the local or global context.

Shim et al. [8] demonstrated the relationship between phonetic and linguistic localization and explained that phonetic localization can be subdivided into two behavioral modes: one where similar phonemes are assigned high correlations with each other, and another where highly correlated feature frames are transformed into similar ones. They also noted that the diagonality of the attention matrix increases in the upper layers. This suggests that local feature extraction predominantly occurs in the upper layers during speech recognition tasks. Additionally, when merging branches with the weighted average method, it is evident that the upper layers in the Branchformer primarily focus on the local branch [3, Sec. 4.6]. Like Conformer, interleaving self-attention and convolution layers may not be the most efficient approach.

Moreover, according to the findings of Yifan Peng et al. on Aishell-1 [9] and Switchboard [10], we hypothesize that merging weight patterns vary with different languages. The low standard deviation of merging weights indicates that different speeches in the same language exhibit similar merging weight patterns. In other words, the merging behavior remains consistent even when the weight is computed based on the input speeches. Furthermore, when the merging weight is low, computational costs are still incurred on the branches that are

ultimately ignored.

In code-switching speeches, since different languages exhibit unique distributions of local and global context, the merging weight pattern varies across encoder layers and changes with language shifts. Therefore, if a model could adaptively switch weight patterns, it might improve parameter efficiency and performance for code-switching speeches, rather than using a consistent merging method. We believe that a flexible structure capable of adjusting the importance of local and global features is essential for optimal code-switching ASR systems.

In this work, we present a new method that is more flexible and dynamic in tuning the importance of local and global features, specifically aimed at code-switching ASR tasks, as shown in Fig. 1. Extended from Branchformer, we propose Dynamic Branch Merging-enhanced Branchformer (DBM-Branchformer), which employs the Dynamic Branch Merging module (DBM) to decide the weights of the local and global branches. The DBM evaluates the weights of the local and global features by considering the hidden vectors from the previous layer and the features extracted from the current layer. Similar to the learned weight method in Branchformer, which calculates the merging weight with the input vectors, DBM-Branchformer dynamically adjusts the weights for each frame of the hidden feature instead of applying uniform weights across the whole hidden sequence.

In Section II, we will introduce the methods used in this work. The details of the experiment will be mentioned in Section III. In Section IV, we will analyze and discuss the result. Finally, the conclusion will be made in Section V.

II. Method

This section takes a deep dive into the core structure of Branchformer and our merge method. We extend the Branchformer with DBM to calculate the weight of the output from local and global branches for the merging process.

A. Local Branch

Branchformer employs convolutional gating multi-layer perceptron (cgMLP) [11] to extract local features. cgMLP deploys a Convolutional Spatial Gating Unit (CSGU) [11] to get the local information. After projecting to the higher dimension, the input is split into two parts, one is the residual part $\mathbf{X}_r$, and the other one is the gate part $\mathbf{X}_g$. After layer normalization, $\mathbf{X}_g$ will be depth-wise convoluted. Then, use the linear gating by directly element-wise multiplication with $\mathbf{X}_r$ and $\mathbf{X}_g$ without activation function. Finally, use a linear layer to project the latent vectors back to the d dimension. The processes of the local branch are written as follows,

$$[\mathbf{X}_r, \mathbf{X}_g] = \text{GELU}(\mathbf{XU}) \tag{1}$$

$$\begin{aligned}\mathbf{Z} &= \text{Dropout}(\text{CSGU}(\mathbf{X}_g) \\ &= \text{Dropout}(\mathbf{X}_r \otimes \text{DWConv}(\text{LN}(\mathbf{X}_g))\end{aligned} \tag{2}$$

$$\mathbf{L} = \mathbf{ZV} \tag{3}$$

where $\mathbf{X} \in \mathbb{R}^{T\times d}$, $\mathbf{X}_r, \mathbf{X}_g \in \mathbb{R}^{T\times d_{\text{latent}}}$ are latent vectors, and $\mathbf{U} \in \mathbb{R}^{d\times 2d_{\text{latent}}}$, $\mathbf{V} \in \mathbb{R}^{d_{\text{latent}}\times d}$ are the weights of two projections before and after the CSGU. d_{latent} is the latent dimension in both projections.

B. Global Branch

To extract the global feature, Branchformer uses multi-head self-attention layers with relative position encoding [12]. The main idea of self-attention is to project the input to matrices $\mathbf{Q}$, $\mathbf{K}$, and $\mathbf{V} \in \mathbb{R}^{T\times d}$, calculating the relation score between $\mathbf{Q}$ and $\mathbf{K}$ for each frame, forming a square matrix $\mathbf{A}_{\text{h}}$. $\mathbf{A}_{\text{h}}$ contains the global information which shows the relationship between each frame. Then, multiply $\mathbf{V}$ with the attention matrix $\mathbf{A}_{\text{h}}$. As for multi-head self-attention, it splits the self-attention process into h groups. h is the number of heads in each self-attention layer, and each head calculates the relation score independently. The multi-head self-attention formula is shown as below,

$$\text{Score}_h = \frac{(\mathbf{Q}_h + \mathbf{u}_h)\mathbf{K}_h^\top}{\sqrt[2]{d/h}} + \frac{\text{RelShift}((\mathbf{Q}_h + \mathbf{v}_h)\mathbf{P}_h^\top)}{\sqrt[2]{d/h}} \tag{4}$$

$$\mathbf{A}_h = \text{Softmax}(\text{Score}_h) \tag{5}$$

$$\mathbf{G} = \text{Dropout}(\mathbf{A}_h)\mathbf{V}_h \tag{6}$$

where $\mathbf{Q}_h$, $\mathbf{K}_h$, and $\mathbf{V}_h \in \mathbb{R}^{h\times T\times d/h}$ are the projection that split into h groups, $\mathbf{u}_h$ and $\mathbf{v}_h \in \mathbb{R}^{h\times d/h}$ are the trainable bias, $\mathbf{P}_h^\top \in \mathbb{R}^{h\times(2T-1)\times d/h}$ is relative positional embedding. After the multiplication with $\mathbf{P}_h^\top$ and $\mathbf{Q}_h$, the relative shifting is applied. d_k/h represents the level of multi-head, used to normalize the feature to avoid the condition that a huge value occupies the output of Softmax, making others become zero.

C. Merge method

Branchformer applies two types of merge methods. One is the concatenation, which combines $\mathbf{L}$ and $\mathbf{G}$, then directly projects the combination to the original size. The process is shown below,

$$\mathbf{Y}_{\text{merge}} = \text{Concat}(\mathbf{L}, \mathbf{G})\mathbf{W} \tag{7}$$

where $\mathbf{W} \in \mathbb{R}^{2d\times d}$ is the trainable matrix, $\mathbf{L}$ and $\mathbf{G}$ are the output of local and global branches.

The other method is the weighted average. The weighted average method calculates the importance score on local and global branches with attention-based pooling. Attention-based pooling squeezes the hidden vectors into a single dimension, then applies softmax on the time domain to get the score. After multiplying the score and the output from both branches, we can get the pooling feature with the d dimension. After projecting the pooling feature to a single dimension, we can get the weight on local or global branches. The formula is shown as follows,

$$\text{Score}_G = \text{Softmax}(\text{PoolProj}_G(\mathbf{G})^\top) \tag{8}$$

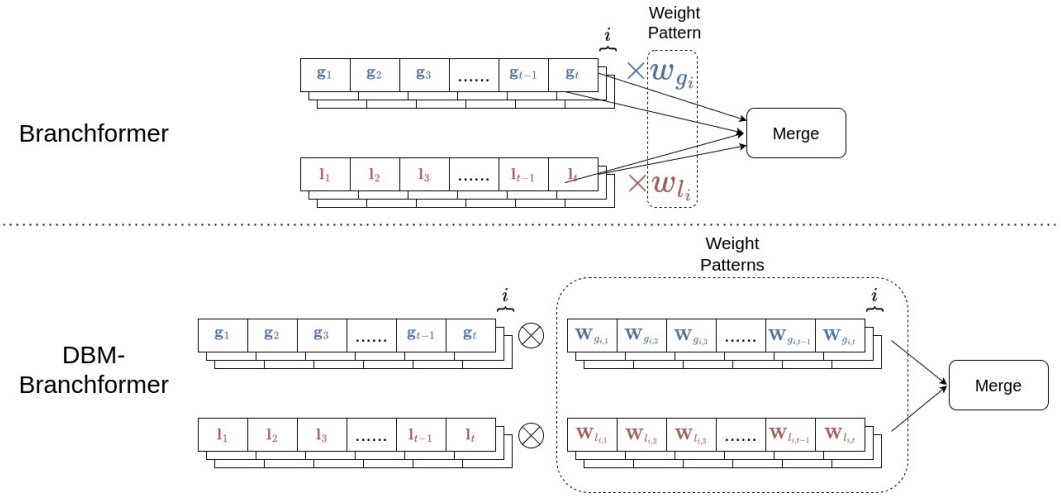

Fig. 1. The concept of dynamic branch merging. The upper part of the figure shows the branch merging method in Branchformer, called "Weighted Average," while the lower part depicts the method we proposed in this study. $g_i, l_i \in \mathbb{R}^{1\times d}$ are the outputs from the global and local branches, t denotes the frame number of the hidden vectors, and d is the feature dimension of the hidden vectors. i indicates the block number of the encoder. The weight pattern represents the merging weight distribution of the encoder. In Branchformer, each weight is calculated from the entire vector sequence from both branches, resulting in a consistent weight pattern over the time domain. In DBM-Branchformer, since the merging method weights are generated separately, each input frame has a unique weight pattern.

$$w_G = \text{Score}_G \mathbf{G}\mathbf{U}_G \tag{9}$$

$$\text{Score}_L = \text{Softmax}(\text{PoolProj}_L(\mathbf{L})^\top) \tag{10}$$

$$w_L = \text{Score}_L \mathbf{L}\mathbf{U}_L \tag{11}$$

$$w'_G, w'_L = \text{Softmax}(w_G, w_L) \tag{12}$$

$$\mathbf{Y}_{\text{merge}} = w'_G \mathbf{G} + w'_L \mathbf{L} \tag{13}$$

where the learnable weight in $\text{PoolProj}_G, \text{PoolProj}_G \in \mathbb{R}^{d\times 1}$, $\mathbf{U}_G, \mathbf{U}_L \in \mathbb{R}^{d\times 1}$ are matrix to project the pooling features to one dimension. L and G are the output from local and global branches. The weights for the local and global branches are denoted by w'_G and w'_L, respectively, where $w'_G, w'_L \in \mathbb{R}$.

Although merging with the concatenation method has been proven that it can get better performance than the weighted average on monolingual datasets, it can not show the importance of local or global features in each layer.

D. Dynamic Branch Merging

One of the challenges in code-switching speech recognition is that the model tends to output words with similar pronunciations but in the wrong language. According to the research by Shim et al., we speculate that one of the reasons for this issue is the lack of sufficient data. Different languages often have unique distributions of local and global features, meaning that these feature distributions in code-switching speech are not fixed but change over time. However, when a model processes all acoustic features with fixed weights equally, it can easily interpret similar-sounding acoustic features from different languages as the same parts.

In the lower layers, due to insufficient data, these similar-sounding feature frames tend to have high correlations among themselves without generating sufficient correlations with adjacent blocks. Consequently, in the upper layers, these features gradually assimilate, leading to errors in judgment.

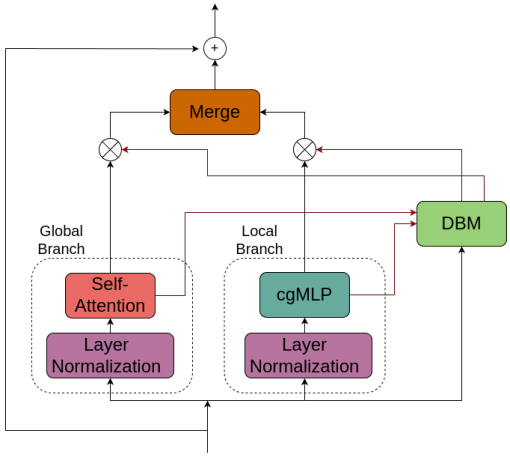

Fig. 2. DBM-Branchformer encoder block architecture. Instead of evaluating $w_G, w_L \in \mathbb{R}$ with the whole hidden sequence which is the weighted average method used in Branchformer, using the DBM module to calculate the $W \in \mathbb{R}^{T\times 2d}$, each element in the hidden vectors not only has the own weight individually but also multiplies with the dynamic weights depending on the input.

Therefore, when using the concatenation method, since the projection merely merges two branches with static weights W, the same projection is applied despite language changes. Similarly, using the weighted average method to merge branches might not be the best choice either. Although w_G and w_L in Section II-C are calculated based on the outputs from the two branches, this calculation is done based on the entire sequence, not on each frame individually. Hence, even if the

language switches, the proportion of the two branches remains unchanged.

To enable the model to adjust its recognition strategy according to input variations, and to make the branch computations more efficient and flexible, preventing some branches from remaining unused, we propose dynamic branch merging (DBM).

DBM-Branchformer, shown as Fig. 2, applies the DBM module deciding the weights to merge both branches. The DBM module determines the attention each frame in the current layer should pay to global and local branches based on the output of each frame from the previous layer.

The process of the DBM module is shown as follows,

$$\mathbf{Y}_{\text{cat}} = \text{Concat}(\mathbf{X}, \mathbf{L} \otimes \mathbf{G}) \tag{14}$$

$$\mathbf{W} = \text{GELU}(\mathbf{Y}_{cat}\mathbf{D}) \tag{15}$$

$$\mathbf{Y}_{\text{merge}} = \mathbf{W} \otimes \text{Concat}(\mathbf{L}, \mathbf{G}) \tag{16}$$

where $\mathbf{W} \in \mathbb{R}^{T \times 2d}$ is the weight for merging the local and global branches, $\mathbf{D} \in \mathbb{R}^{2d \times 2d}$ is the learnable weight. Since the weight $\mathbf{W}$ varies with $\mathbf{X}$, $\mathbf{L}$, and $\mathbf{G}$, the DBM module merges both branches, depending on each hidden vector instead of the static value or the score of each branch.

To achieve flexible branch merging and allow the model to skip certain branches, we use GELU for calculating the merging weights instead of Softmax. This prevents the DBM module from being forced to choose between two branches when it considers both branch outputs unsuitable, thereby increasing training stability.

To make the DBM module consider not only the output of the previous layer but also the position information of local and global features extracted in the current layer, $\mathbf{Y}_{\text{cat}}$ is composed by $\mathbf{X}$ and the tensor product of the outputs from both branches. We found that this approach makes the generated weights $\mathbf{W}$ more dynamic and improves recognition performance on code-switching speeches.

III. Experiment

This section introduces the experiment method, such as parameter setting, dataset, and data preprocessing.

A. Dataset

In this work, TALCS [13] is used to evaluate the performance which has about 587 hours of Mandarin-English code-switching speech audio, provided by TAL Education Group. All of the audio in dataset are sampled at 16 kHz.

B. Data Preprocessing and Augmentation

For data augmentation, we apply speed perturbation with the ratios 0.9, 1.0, and 1.1 [14]. We also utilize SpecAug [15] with 5 time-warp-windows, 2 frequency-masks which range in [0,27], and 10 time-masks which range in [0,0.05].

To lower the dictionary size, we apply the byte encode module provided by Icefall [5] to encode the Chinese characters. For example, "你好 ITS'S OKAY 的" is converted to "ţN̪ ţ ITS'S OKAY L'ĥ". With the byte encode module, the dictionary size decreases to 1,000.

C. Experiment setting

We use NVIDIA RTX A4500 with 20GB memory for the training process. In this study, we utilize the ESPnet toolkit [16] for the training process.

The ASR model we use consists of three main parts: an encoder, a decoder, and a CTC (Connectionist Temporal Classification) module [17].

We present the total loss function of the entire model below,

$$Loss_{\text{asr}} = (1-\lambda)Loss_{\text{dec}} + \lambda Loss_{\text{ctc}} \tag{17}$$

$Loss_{\text{asr}}$ is the total loss of the entire ASR model. $Loss_{\text{dec}}$ is the loss from the decoder. $Loss_{\text{ctc}}$ is the loss which CTC accounting for. λ is the hyperparameter to adjust the influence from the decoder and the CTC model. In this work, we don't use a language model to analyze the model performance. In the training process, we set λ to 0.3.

During training, we set the label smoothing weight to 0.1 and use Adam [18] as the optimizer with weight decay 1e-6.

1) Encoder Setup: In Section II-B, it is mentioned that multi-head self-attention is used in the calculation of the global branch. In this study, we set the number of heads h to 8 and the feature dimension d to 512.

We set the number of encoder layers for the DBM-Branchformer to 18, and to ensure a fair comparison with a similar number of parameters, we set the number of encoder layers for the Branchformer to 22.

The cgMLP in Branchformer and DBM-Branchformer has kernel size 31 and latent dimension d_{latent} 3072.

To speed up the process and decrease memory usage in self-attention, we apply the subsampling module in the front of the encoder. The subsampling module consists of two 2d-convolution layers, each with kernel size (3x3) and stride (2x2) to downsample the input sequence by a factor of 4.

2) Decoder Setup: In this study, the Transformer decoder is employed in all cases, maintaining the same dimension d as the encoder. The decoder would calculate the relationship between the acoustic features by the multi-head cross-attention and between the previous output texts by the multi-head causal self-attention. Then, similar to the CTC module, outputs the most likely element as a vector from the dictionary.

The causal self-attention is similar to the original self-attention method but has a multiplication of an upper triangular matrix. This modification aims to eliminate the relationship between the hidden vectors and their successors.

In this study, all the models used for comparison employed the same decoder architecture. We set the number of heads to 8, the feature dimension to 512, matching the encoder, and the number of layers to 6.

In the inference stage, we use "Rescoring" as the decoding method [19]. First, it uses beam search decoding to take the top n_{bs} list of hypotheses from the decoder's output. Second, it rescores the hypotheses with the log probabilities of the output from CTC and the decoder. We set n_{bs} to 10, and CTC weight to 0.4.

IV. Results

In this chapter, we compare the performance differences between DBM-Branchformer and Branchformer in code-switching tasks to demonstrate the impact of the dynamic branch merging strategy on the recognition ability of code-switched speeches. Additionally, in Section II-C, we mentioned that the concatenation method of Branchformer performs better than the weighted average method on monolingual datasets. We are also curious whether their performance on code-switching datasets will be consistent with the results on monolingual datasets, so we also include them in the comparison.

A. Main Result

Table I compares the result of both Branchformer and DBM-Branchformer oh TALCS. Both methods in Branchformer are trained for 60 epochs, while the DBM-Branchformer goes through 49 epochs of training. The DBM-Branchformer outperforms the Branchformer with the concatenation merging method in WER by 0.1%. It also outperforms the weighted average method by 0.2%.

B. Analysis of Merging Weight

In the TALCS test set, the average weight distribution for each layer when Branchformer uses weighted averaging to merge branches is shown in the Fig. 3. According to our measurements, the highest variance of branch merging weights across layers on the TALCS test set is in the first layer, at approximately 0.0012. This indicates that even in the case of code-switching, the branch merging weights calculated by the weighted average method remain nearly static, similar to the monolingual result, despite these values being based on the branches' outputs at each layer. We believe this is because the pooling computation in the weighted average method overlooks the dynamic of the features, leading the merging modules to disregard the uniqueness of different inputs.

In contrast, DBM-Branchformer shows a minimum variance of 0.0057 in the merging weights across layers on the TALCS test set. This suggests that DBM-Branchformer can handle global and local dependencies more flexibly than Branchformer.

The variance distribution of the merge weights of each layer is shown in Fig. 4. It can be seen that both DBM-Branchformer and Branchformer have higher variance at the first layer. We believe that this is because the encoder in the bottom layer focuses on phonetic localization, which is more variable than linguistic localization.

V. Conclusions

In this study, we propose a different method for merging the local and global branches in Branchformer. Aiming at code-switching auto speech recognition, extending from the merging methods used in Branchformer, DBM-Branchformer uses a Dynamic Branch Merging module to calculate the importance of each element in the hidden vectors. Instead of considering the whole input to weigh the local and the global branches, the DBM module calculates the weights independently to adapt the code-switching speech. By incorporating the DBM module, the DBM-Branchformer outperforms the Branchformer in TALCS without using a language model.

Level	Local	Global
22	0.9432	0.0568
21	0.9282	0.0718
20	0.9397	0.0603
19	0.9553	0.0447
18	0.9587	0.0413
17	0.1904	0.8096
16	0.1062	0.8938
15	0.1086	0.8914
14	0.1135	0.8865
13	0.0091	0.9909
12	0.1119	0.8881
11	0.1113	0.8887
10	0.0816	0.9184
9	0.0903	0.9097
8	0.0753	0.9247
7	0.0977	0.9023
6	0.9377	0.0623
5	0.1000	0.9000
4	0.9458	0.0542
3	0.0897	0.9103
2	0.9582	0.0418
1	0.0012	0.9988

Fig. 3. The weight pattern of Branchformer with the weighted average method on TALCS Test set. When training on code-switching datasets, we can observe that the branch merging strategy in the lower layers of Branchformer can be roughly divided into three parts. The lower part alternates between using global and local branches. The middle part tends to use the self-attention layer to capture global features. The upper part focuses on using cgMLP to process local information.

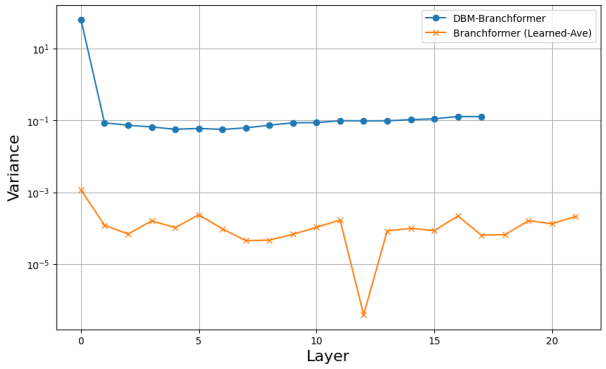

Fig. 4. The distribution of the variances of the merge weights of each layer in DBM-Branchformer and Branchformer.The variances of the merge weights of each layer in DBM-Branchformer are higher than those of Branchformer.

TABLE I
COMPARISON OF BRANCHFORMER AND DBM-BRANCHFORMER IN THE TALCS CODE-SWITCHING DATASET. WE ANALYZE THE PERFORMANCE WITH WORD ERROR RATE (WER) AND APPLY THE BYTE ENCODE MODULE.

Model Type	Params(M)	Test set
Branchformer		
With Concat	127.76	7.0
With Learned Ave	122.04	7.1
DBM-Branchformer	129.64	**6.9**

REFERENCES

[1] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[2] A. Gulati, J. Qin, C.-C. Chiu, *et al.*, "Conformer: Convolution-augmented transformer for speech recognition," *arXiv preprint arXiv:2005.08100*, 2020.

[3] Y. Peng, S. Dalmia, I. Lane, and S. Watanabe, "Branchformer: Parallel mlp-attention architectures to capture local and global context for speech recognition and understanding," in *International Conference on Machine Learning*, PMLR, 2022, pp. 17 627–17 643.

[4] K. Kim, F. Wu, Y. Peng, *et al.*, "E-branchformer: Branchformer with enhanced merging for speech recognition," in *2022 IEEE Spoken Language Technology Workshop (SLT)*, IEEE, 2023, pp. 84–91.

[5] Z. Yao, L. Guo, X. Yang, *et al.*, "Zipformer: A faster and better encoder for automatic speech recognition," *arXiv preprint arXiv:2310.11230*, 2023.

[6] J. Li, Z. Duan, S. Li, X. Yu, and G. Yang, "Gncformer enhanced self-attention for automatic speech recognition," *arXiv preprint arXiv:2305.12755*, 2023.

[7] A. Andrusenko, R. Nasretdinov, and A. Romanenko, "Uconv-conformer: High reduction of input sequence length for end-to-end speech recognition," in *ICASSP 2023-2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE, 2023, pp. 1–5.

[8] K. Shim, J. Choi, and W. Sung, "Understanding the role of self attention for efficient speech recognition," in *International Conference on Learning Representations*, 2021.

[9] H. Bu, J. Du, X. Na, B. Wu, and H. Zheng, "Aishell-1: An open-source mandarin speech corpus and a speech recognition baseline," in *2017 20th conference of the oriental chapter of the international coordinating committee on speech databases and speech I/O systems and assessment (O-COCOSDA)*, IEEE, 2017, pp. 1–5.

[10] J. J. Godfrey, E. C. Holliman, and J. McDaniel, "Switchboard: Telephone speech corpus for research and development," in *Acoustics, speech, and signal processing, ieee international conference on*, IEEE Computer Society, vol. 1, 1992, pp. 517–520.

[11] J. Sakuma, T. Komatsu, and R. Scheibler, "Mlp-based architecture with variable length input for automatic speech recognition," 2021.

[12] Z. Dai, Z. Yang, Y. Yang, J. Carbonell, Q. V. Le, and R. Salakhutdinov, "Transformer-xl: Attentive language models beyond a fixed-length context," *arXiv preprint arXiv:1901.02860*, 2019.

[13] C. Li, S. Deng, Y. Wang, *et al.*, "Talcs: An open-source mandarin-english code-switching corpus and a speech recognition baseline," *arXiv preprint arXiv:2206.13135*, 2022.

[14] T. Ko, V. Peddinti, D. Povey, and S. Khudanpur, "Audio augmentation for speech recognition," in *Sixteenth annual conference of the international speech communication association*, 2015.

[15] D. S. Park, W. Chan, Y. Zhang, *et al.*, "Specaugment: A simple data augmentation method for automatic speech recognition," *arXiv preprint arXiv:1904.08779*, 2019.

[16] S. Watanabe, T. Hori, S. Karita, *et al.*, "Espnet: End-to-end speech processing toolkit," *arXiv preprint arXiv:1804.00015*, 2018.

[17] S. Kim, T. Hori, and S. Watanabe, "Joint ctc-attention based end-to-end speech recognition using multi-task learning," in *2017 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, IEEE, 2017, pp. 4835–4839.

[18] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[19] T. Hori, S. Watanabe, and J. R. Hershey, "Joint ctc/attention decoding for end-to-end speech recognition," in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2017, pp. 518–529.