Interspeech 2024
1-5 September 2024, Kos, Greece

HuBERT-EE: Early Exiting HuBERT for Efficient Speech Recognition

Ji Won Yoon[1], *Beom Jun Woo*[2], *Nam Soo Kim*[2]

[1]Department of AI, Chung-Ang University, Seoul, South Korea
[2]Department of ECE and INMC, Seoul National University, Seoul, South Korea

jiwonyoon@cau.ac.kr, bjwoo@hi.snu.ac.kr, nkim@snu.ac.kr

Abstract

Pre-training with self-supervised models, such as Hidden-unit BERT (HuBERT) and wav2vec 2.0, has brought significant improvements in automatic speech recognition (ASR). However, these models usually require an expensive computational cost to achieve outstanding performance, slowing down the inference speed. To improve the model efficiency, we introduce an early exit scheme for ASR, namely HuBERT-EE, that allows the model to stop the inference dynamically. In HuBERT-EE, multiple early exit branches are added at the intermediate layers. When the intermediate prediction of the early exit branch is confident, the model stops the inference, and the corresponding result can be returned early. We investigate the proper early exiting criterion and fine-tuning strategy to effectively perform early exiting. Experimental results on the LibriSpeech show that HuBERT-EE can accelerate the inference of the HuBERT while simultaneously balancing the trade-off between the performance and the latency.

Index Terms: self-supervised learning, early exit, speech recognition, connectionist temporal classification

1. Introduction

Recently, self-supervised speech representation learning (speech SSL) [1, 2, 3, 4, 5] has achieved considerable improvements in automatic speech recognition (ASR) literature. Unlike fully-supervised learning approaches, which rely on manually annotated labels, speech SSL models can learn a meaningful speech representation by leveraging unlabeled speech data.

Among the various speech SSL models, Hidden-unit BERT (HuBERT) [2] is one of the most prominent models for speech recognition. On the LibriSpeech [6], fine-tuned HuBERT using connectionist temporal classification (CTC) [7] achieves the state-of-the-art word error rate (WER) results. However, such a model tends to have a large model size and high computational complexity to achieve promising performance. For example, the base version of the HuBERT has about 95 million parameters. Also, a large version utilizes twice as many Transformer layers [8] as in the base version, with almost 317 million parameters. These large-scale pre-trained models usually suffer from slow inference speed, which may hinder their usage in real-world applications where fast inference is desirable.

Typical approaches to improve model efficiency include knowledge distillation (KD) [9, 10], pruning [11, 12], and model quantization [13]. While those methods reduce the processing complexity, they still require samples to pass through the entire model. In contrast, early exiting is a technique to adaptively accelerate the inference speed by returning the result at an intermediate layer. Since multiple classifiers are attached to some intermediate layers and jointly trained with the original backbone model, each classifier yields the prediction and confidence score during the inference. When the intermediate prediction is confident enough, the corresponding result can be exited early. However, existing early exit methods [14, 15, 16, 17, 18] are mainly designed for natural language processing (NLP) classification tasks. Only a few studies have been investigated in the speech domain, including speech enhancement [19], speech separation [20], and limited-vocabulary commands recognition [21]. Since the ASR model does not use the commonly-used classifier for classification, it is challenging to directly apply the previous approaches to the ASR model.

In this paper, we introduce a simple yet effective early exit method for ASR, namely HuBERT Early Exiting (HuBERT-EE), that enables the HuBERT model to stop the inference dynamically. *To the best of our knowledge, this is the first attempt to apply the early exit framework to the speech SSL model.* Specifically, the proposed HuBERT-EE accelerates the inference procedure by adding multiple early exit branches at the intermediate layers of the HuBERT. When the early exit branch is confident in its prediction, the model stops the inference and outputs the intermediate prediction as the final result. Different from intermediate CTC-based approaches [22, 23, 24, 25], the HuBERT-EE aims to dynamically use the intermediate prediction as the model's final output with minimal WER degradation. Instead of simply applying the intermediate-CTC framework, we newly construct the self-attention-based early exit branch to perform the early exiting effectively. In addition, we explore the proper early exiting criterion and fine-tuning strategy to perform the early exiting effectively. Also, we newly design the self-attention-based early exit branch.

From the experimental results on the LibriSpeech dataset, it is verified that the HuBERT-EE can be successfully applied to the ASR task. Compared to the other compression methods, HuBERT-EE enables the model to stop the inference dynamically while achieving a better speed-performance trade-off. This implies that the proposed method can be applied to a real-world scenario in which users have the flexibility to adjust the inference speed according to their demands.

2. HuBERT-EE

Conventional early exit methods have mainly focused on NLP classification tasks. Considering that the ASR model does not employ the typical classifier used in classification tasks, it is essential to develop a new early exiting framework specifically tailored for ASR. In this section, we introduce a pioneering early exit method designed for HuBERT, namely HuBERT-EE. Although our experiments utilize HuBERT as the backbone model, the proposed framework can be extended to other SSL models as well.

10.21437/Interspeech.2024-80

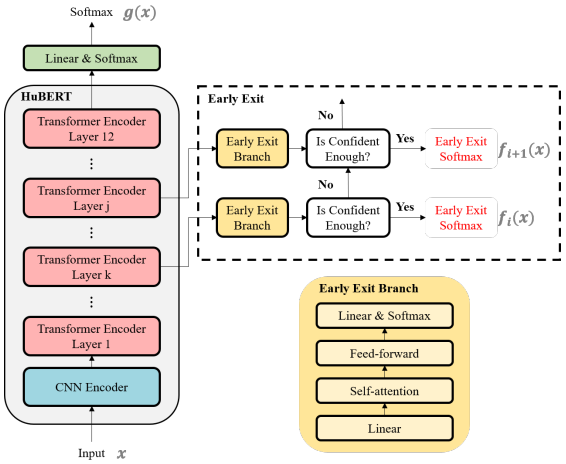

Figure 1: *Overview of the HuBERT-EE. In contrast to the original HuBERT model, our proposed approach can stop the inference dynamically. If an early exit branch is sufficiently confident in its prediction, the corresponding result can be exited early.*

2.1. Model architecture

As shown in Figure 1, HuBERT-EE mainly consists of a backbone network and multiple early exit branches. The backbone is built upon a 12-layers Transformer encoder with an additional linear projection layer. The early exit branches are located at some intermediate layers of the backbone network to enable early predictions.

Following the previous studies [14, 15], we apply the base version of the HuBERT as for the backbone, which contains 12 Transformer encoder layers. For the convenience of notation, we let HuBERT-base denotes the base version of the HuBERT. The backbone is composed of a convolutional neural network (CNN) encoder, 12 Transformer encoder layers, and the linear layer with softmax function. The structure of the CNN encoder and the Transformer encoder conform with those of the original HuBERT. In order to fine-tune the model for the speech recognition task, the linear projection layer is added to the final layer of the HuBERT-base model.

In the original HuBERT, the way to fine-tune the model is to attach one linear projection layer to the final Transformer encoder layer. Then, the linear layer outputs the prediction for the ASR task. In the proposed method, we add early exit branches to the intermediate layers of the HuBERT, enabling more efficient CTC predictions, like intermediate-CTC [22]. However, we experimentally found that simply applying intermediate-CTC with a linear layer did not perform well in the early exit framework. Instead, as shown in Figure 1, we newly construct the early exit branch with the self-attention layer at its core, motivated by the Transformer structure. It is designed carefully to balance the trade-off between performance and efficiency.

2.2. Pre-training

Before the fine-tuning stage, we start with pre-training the backbone model on unlabelled data with a self-supervised learning objective. This stage is identical to the vanilla HuBERT pre-training. Note that the linear layer, located at the final Transformer layer, and all early exit branches stay unaffected during the pre-training. In our experiments, we used the pre-trained HuBERT checkpoint, which is provided by the Fairseq [26] toolkit.

2.3. Fine-tuning

In this subsection, we discuss how to fine-tune the proposed HuBERT-EE. Firstly, the CTC loss function for the last linear projection layer, which is located on top of the Transformer, can be formulated as

$$\mathcal{L}_{FT1} = CTC_{loss}(y, g(x)) \tag{1}$$

where x, y, g, CTC_{loss} denote the input sequence, the corresponding label, the output of the linear projection layer, and the CTC loss, respectively. This training is identical to the HuBERT fine-tuning in the original paper [2].

For fine-tuning the early exit branches on the ASR task, the CTC loss function of the i^{th} early exit branch is as follows:

$$\mathcal{L}_i = CTC_{loss}(y, f_i(x)) \tag{2}$$

where f_i denotes the output of the i^{th} early exit branch. When there are N branches in the HuBERT-EE, the loss for fine-tuning all early exit branches can be calculated as

$$\mathcal{L}_{FT2} = \sum_{i=1}^{N} \mathcal{L}_i. \tag{3}$$

Due to the performance degradation, we consider uniform weights for training the early exit branches instead of the weighted average [27].

Based on the two losses $\mathcal{L}_{FT1}$ and $\mathcal{L}_{FT2}$, we investigate the effective fine-tuning approach to train the HuBERT-EE. In the previous related studies, there are mainly two fine-tuning strategies for training the early-exit model: (1) joint training that jointly fine-tunes the final linear layer and all early-exit branches and (2) two-stage training that fine-tunes the two components separately. In Section 3.3, we compare these two fine-tuning strategies and look for the proper one to train our framework.

The straightforward fine-tuning approach is to jointly train the last linear layer and all the early exit branches [16, 17] by minimizing the sum of the two loss functions $\mathcal{L}_{FT1} + \lambda\mathcal{L}_{FT2}$. In our experiments, we experimentally set λ to 1.

When it is required to maintain the best performance of the final linear layer, the two-stage training is the desired fine-tuning approach [14]. In this training scheme, we first fine-tune the whole model weights with the loss function $\mathcal{L}_{FT1}$, except for the early exit branches. Then, we freeze all parameters fine-tuned in the previous stage and only update the early exit branches with CTC loss $\mathcal{L}_{FT2}$. Note that the reason for freezing parameters of the backbone and the final linear layer is to keep the high performance of the original HuBERT-base.

2.4. Early exit inference

After fine-tuning HuBERT-EE for ASR, the model is capable of making early exit decisions during the inference procedure. Each early exit branch, added at the intermediate Transformer layer, outputs the prediction and confidence score. If the intermediate prediction is confident enough, the forward inference is terminated, and the result is returned early. In this paper, we quantify the early exit branch's confidence in its prediction in two ways: entropy and maximum probability. In Section 3.2, we compare these two criteria and determine the optimal one.

2.4.1. Entropy

Since entropy is a well-known measure of uncertainty, we use the entropy-derived confidence measure as the early exit criterion. The entropy of the i^{th} early exit branch's output $f_i(x)$ can be computed as

$$\text{Entropy} = -\frac{1}{T \times C} \sum_{T} \sum_{C} f_i(x) \times \log f_i(x). \quad (4)$$

The prediction with lower entropy might be more confident to exit. If the entropy of $f_i(x)$ is lower than the preset threshold S, HuBERT-EE stops the inference, returning the result early.

2.4.2. Confidence

The maximum probability is another straightforward measure of certainty. Since we use the CTC framework, the softmax prediction of the i^{th} early exit branch can be expressed as $f_i(x) \in R^{T \times C}$, where T is the total number of frames and C is the number of label classes. Considering the maximum probability as the confidence measure, the average confidence score of $f_i(x)$ is given as

$$\text{Confidence} = \frac{1}{T} \sum_{T} \max_{c} f_i(x)^{(c)} \quad (5)$$

where $\max_c f_i(x)^{(c)} \in R^{T \times 1}$ represents the maximum probability for each frame. When the confidence of the intermediate output $f_i(x)$ is larger than the predefined threshold, the corresponding prediction can be exited early.

3. Experiments

3.1. Experimental setup

We used the LibriSpeech [6] (about 1000 hours) for pre-training and supervised fine-tuning. As the training dataset, "train-clean-100", "train-clean-360", and "train-other500" were used. For validation, we used "dev-other". We applied "test-clean" and "test-other" for evaluation.

We applied HuBERT-EE to the HuBERT-base model, containing 12 Transformer encoder layers. For implementation, the Fairseq [26] toolkit was mainly utilized to build the models. In the case of the early exit branch, we added early exit branches to three layers: 5^{th}, 8^{th}, and 11^{th} layers of the HuBERT-base. Each early exit branch module had the self-attention dimension D_{EE} of 512 with four heads. The additional early exit branches corresponds to about 22 M parameters, resulting in a total of 116 M parameters for the HuBERT-EE. Instead of directly pre-training the HuBERT backbone model, we used the pre-trained checkpoint, provided by the Fairseq toolkit. When fine-tuning the HuBERT-EE, we followed the fine-tuning scheme of the original paper [2], and the training was performed on four NVIDIA Quadro RTX 8000 GPUs.

We compared the HuBERT-EE with other compression techniques, including DistilHuBERT [28] and LayerDrop [29]. All the models were pre-trained and fine-tuned using 960 hours of LibriSpeech. To fairly compare the results, a single linear layer was employed as the ASR module, placed on top of the SSL model. Both the pre-trained SSL model and the ASR module were fine-tuned together during the training. We found that the original DistilHuBERT model, which consisted of two Transformer encoder layers, did not perform well when using a linear layer as the ASR module. To address this, we experimented with 8 Transformer encoder layers of DistilHuBERT, referred to as DistilHuBERT-8L. Regarding LayerDrop, we applied it to the HuBERT-base model during the fine-tuning procedure and set the LayerDrop rate p to 0.1 and 0.3.

		test-clean	
		WER	RTF ($\times 10^{-3}$)
HuBERT-base (backbone)		3.88 %	3.529
HuBERT-EE (Ours)	Entropy Thres.=0.0040	8.05 %	2.879
	Entropy Thres.=0.0035	6.50 %	2.999
	Entropy Thres.=0.0025	4.17 %	3.312
	Confidence Thres.=0.950	8.25 %	2.887
	Confidence Thres.=0.955	6.82 %	3.043
	Confidence Thres.=0.960	5.62 %	3.308

Table 1: *WER (%) on test-clean dataset using different predefined thresholds.*

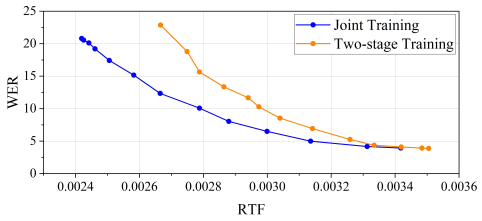

Figure 2: *Quality–efficiency trade-offs on test-clean dataset using different fine-tuning strategies for HuBERT-EE. We set entropy thresholds S from 0.008 to 0.002.*

We measured two performance metrics: word error rate (WER) and real time factor (RTF). WER is a widely used metric to evaluate the accuracy of ASR task, and RTF measures a decoding speed with the ratio between the ASR processing time and the utterance duration.

For the inference, we applied greedy decoding without a language model. Since some model configurations did not support CPU-only inference, we evaluated GPU-based inference for each model. The ASR models with a large model size typically use GPU resources for inference, so it is reasonable to utilize the GPU for decoding models. RTF was measured on a single NVIDIA Quadro RTX 8000 GPU with single batch size, and we averaged RTF results over three runs.

3.2. Exploring suitable early exiting criterion

To determine the proper early exit criterion, we examined the predefined threshold values for both entropy (in Eq. (4)) and confidence (in Eq. (5)). As shown in Table 1, we observed that both confidence and entropy-derived criterions performed well on ASR. However, the entropy criterion was more supportive in making early exit decisions, achieving better WER performance with lower RTF values. Specifically, on the test-clean dataset, HuBERT-EE with the entropy threshold 0.0035 achieved a WER of 6.50 %. In contrast, HuBERT-EE with the confidence threshold 0.955 resulted in a slightly worse WER of 6.82 %, while exhibiting slower inference speed. This suggests that utilizing the entropy criterion in HuBERT-EE leads to better trade-offs between WER and inference speed compared to the confidence one. Therefore, we applied the entropy-derived criterion in Eq. (4) as the baseline metric to decide the exiting.

3.3. Proper fine-tuning strategy for HuBERT-EE

In Section 2.3, we discussed two fine-tuning strategies for HuBERT-EE: (1) joint training and (2) two-stage training. In Figure 2, we visualized the trade-off while setting different entropy thresholds from 0.008 to 0.002. Entropy is adopted as the early exit criterion, as it performed better than confidence in the previous experiment. We measured both RTF and WER per-

Exit Layer	Joint Training	Two-stage Training
5	21.11 %	37.36 %
8	8.60 %	11.99 %
11	4.04 %	4.21 %
12	3.90 %	3.88 %

Table 2: *Each exit layer's WER (%) on test-clean dataset using different fine-tuning strategies.*

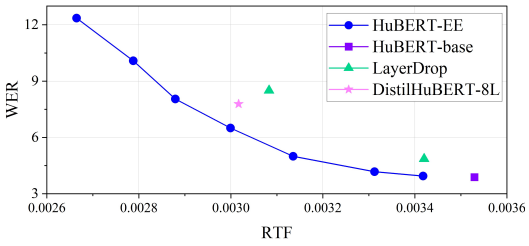

Figure 3: *Performance comparison on test-clean. All results were evaluated based on greedy decoding. We set different thresholds S from 0.005 to 0.002 for HuBERT-EE. The proposed model was fine-tuned with joint training.*

		test-other		
		WER	RTF ($\times 10^{-3}$)	Speed
HuBERT-base (backbone)		9.09 %	3.629	42.12 Hz
DistilHuBERT-8L		19.21 %	3.023	50.55 Hz
LayerDrop	p=0.1	10.93 %	3.493	43.75 Hz
	p=0.3	16.92 %	3.085	49.54 Hz
HuBERT-EE (Ours)	S=0.0055	18.20 %	2.955	51.72 Hz
	S=0.005	16.13 %	3.043	50.23 Hz
	S=0.003	10.04 %	3.439	44.44 Hz

Table 3: *Performance comparison on test-other. Speed of k Hz means that the model can process k samples per second.*

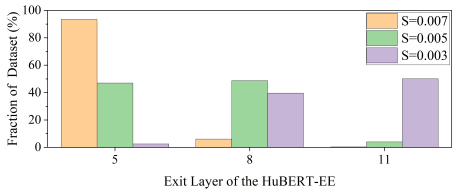

Figure 4: *The number of exiting samples on text-clean. Samples that did not exit earlier were returned via the last linear layer.*

formance on the test-clean. The trade-off curves demonstrate that the HuBERT-EE with joint training showed a better trade-off compared to the two-stage training approach. As the RTF value decreases, the difference between the two methods became more apparent. This is because the two-stage training was considerably weaker in earlier layers, as shown in Table 2. From the results, it is confirmed that the joint training was preferable as the fine-tuning strategy of the HuBERT-EE.

3.4. Performance comparison with conventional methods

We compared the performance of the proposed approach with the conventional compression methods for HuBERT, including DistilHuBERT [28] and LayerDrop [29]. DistilHuBERT is the recent distillation method to reduce the size of HuBERT, and LayerDrop is an effective structured pruning technique for Transformer network. We used the entropy-based metric as the early exit criterion and fine-tuned HuBERT-EE with joint training due to their promising results in previous experiments. Figure 3 shows quality–efficiency trade-offs on test-clean. From the results, it is verified that HuBERT-EE indeed achieved a better speed-performance trade-off compared to the others. In addition, the proposed framework enabled HuBERT to adjust the inference speed without requiring model retraining. This flexibility is particularly advantageous in resource-constrained scenarios. By fine-tuning specific early exit branches, HuBERT-EE could provide greater control over the inference speed. *It's important to note that small RTF gains were a result of our GPU-based evaluation since some baseline model configurations did not support CPU-only inference.* The technique's significance goes beyond RTF gains. HuBERT-EE outperformed DistilHuBERT and LayerDrop, allowing the model to stop the inference dynamically. As shown in Table 3, HuBERT-EE still performed better on the test-other dataset. Overall, the experimental results suggest that HuBERT-EE could be a promising solution for efficient ASR inference. It striked a favorable balance between performance and efficiency, making it an attractive choice for practical ASR applications.

3.5. Number of exiting samples

We experimentally attached three early exit branches to the intermediate layer: 5^{th}, 8^{th}, and 11^{th} layers of the HuBERT-base model. As displayed in Figure 4, we further showed the distribution of exit layers while varying the entropy threshold S from $\{0.007, 0.005, 0.003\}$. For instance, when the entropy threshold was set to $S = 0.007$, approximately 94 % of the samples completed the inference at the first early exit branch. This indicates that a significant majority of the samples were able to exit early based on the given criterion. The results further demonstrated that as the entropy threshold increased, a larger proportion of samples exited earlier, highlighting the effectiveness of the utterance-level entropy criterion in making early exit decisions for the ASR task.

4. Limitations

In our study, we employed an entropy-based metric as the criterion for early exiting. However, we observed that the entropy values were relatively small due to the peak feature of the CTC softmax outputs. As a result, the entropy-based metric became sensitive and required careful selection of an appropriate threshold. This was crucial to prevent premature exit or unnecessary computations during the inference process. Therefore, it is important to consider the specific task and dataset characteristics and carefully choose an appropriate threshold to ensure optimal performance and avoid any potential drawbacks related to early exiting decisions.

5. Conclusions

In this paper, we introduced a novel early exit mechanism for ASR, namely HuBERT-EE, that can dynamically accelerate the inference of a large-scale HuBERT model. From the experimental results on the LibriSpeech, it is verified that the HuBERT-EE was successfully applied to the ASR task while achieving a better quality–efficiency trade-off compared to other compression techniques. Moreover, we conducted detailed analyses to determine the optimal training strategy and early exit criterion for the early exit branch.

6. Acknowledgements

This work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No.2021-0-00456, Development of Ultra-high Speech Quality Technology for Remote Multi-speaker Conference System)

7. References

[1] A. Baevski, H. Zhou, A. Mohamed, and M. Auli, "Wav2vec 2.0: a framework for self-supervised learning of speech representations," in *Proc. NIPS*, 2020.

[2] W. N. Hsu, B. Bolte, Y. H. H. Tsai, K. Lakhotia, R. Salakhutdinov, and A. Mohamed, "Hubert: self-supervised speech representation learning by masked prediction of hidden units," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 29, pp. 3451–3460, 2021.

[3] S. Chen, C. Wang, Z. Chen, Y. Wu, S. Liu, Z. Chen, J. Li, N. Kanda, T. Yoshioka, X. Xiao, J. Wu, L. Zhou, S. Ren, Y. Qian, Y. Qian, J. Wu, M. Zeng, X. Yu, and F. Wei, "Wavlm: large-scale self-supervised pre-training for full stack speech processing," *arXiv preprint arXiv:2110.13900v4*, 2022.

[4] C. Wang, Y. Wu, S. Chen, S. Liu, J. Li, Y. Qian, and Z. Yang, "Improving self-supervised learning for speech recognition with intermediate layer supervision," in *Proc. ICASSP*, 2022.

[5] A. Baevski, W. Hsu, Q. Xu, A. Babu, J. Gu, and M. Aulim, "Data2vec: a general framework for self-supervised learning in speech, vision and language," in *Proc. ICML*, 2022, pp. 1298–1312.

[6] V. Panayotov, G. Chen, D. Povey, and S. Khudanpur, "Librispeech: an asr corpus based on public domain audio books," in *Proc. ICASSP*, 2015, pp. 5206–5210.

[7] A. Graves, S. Fernandez, F. Gomez, and J. Schmidhuber, "Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks," in *Proc. ICML*, 2006, pp. 369–376.

[8] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proc. NIPS*, 2017, pp. 5998–6008.

[9] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," in *Proc. NIPS Workshop Deep Learn.*, 2014.

[10] C. Bucila, R. Caruana, and A. Niculescu-Mizil, "Model compression," in *Proc. ACM SIGKDD*, 2006, p. 535–541.

[11] S. Han, H. Mao, and W. J. Dally, "Deep compression: compressing deep neural network with pruning, trained quantization and huffman coding," in *Proc. ICLR*, 2016.

[12] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," in *Proc. ICLR*, 2017.

[13] J. Wu, C. Leng, Y. Wang, Q. Hu, and J. Cheng, "Quantized convolutional neural networks for mobile devices," in *Proc. CVPR*, 2016, p. 4820–4828.

[14] J. Xin, R. Tang, J. Lee, Y. Yu, and J. Lin, "Deebert: dynamic early exiting for accelerating bERT inference," in *Proc. ACL*, 2020, pp. 2246–2251.

[15] W. Liu, P. Zhou, Z. Zhao, Z. Wang, H. Deng, and Q. Ju, "Fastbert: a self-distilling bert with adaptive inference time," in *Proc. ACL*, 2020, p. 6035–6044.

[16] R. Schwartz, G. Stanovsky, S. Swayamdipta, J. Dodge, and N. A. Smith, "The right tool for the job: matching model and instance complexities," in *Proc. ACL*, 2020, p. 6640–6651.

[17] W. Zhou, C. Xu, T. Ge, J. McAuley, K. Xu, and F. Wei, "Bert loses patience: fast and robust inference with early exit," in *Proc. NIPS*, 2020.

[18] J. Xin, R. Tang, Y. Yu, , and J. Lin, "Berxit: early exiting for bert with better fine-tuning and extension to regression," in *Proc. EACL*, 2021, p. 91–104.

[19] A. Li, C. Zheng, L. Zhang, and X. Li, "Learning to inference with early exit in the progressive speech enhancement," in *Proc. EUSIPCO*, 2021, pp. 466–470.

[20] S. Chen, Y. Wu, Z. Chen, T. Yoshioka, S. Liu, J. Li, and X. Yu, "Don't shoot butterfly with rifles: multi-channel continuous speech separation with early exit transformer," in *Proc. ICASSP*, 2021, pp. 6139–6143.

[21] R. Tang, K. Kumar, J. Xin, P. Vyas, W. Li, G. Yang, Y. Mao, C. Murray, and J. Lin, "Temporal early exiting for streaming speech commands recognition," in *Proc. ICASSP*, 2022.

[22] J. Lee and S. Watanabe, "Intermediate loss regularization for ctc-based speech recognition," in *Proc. ICASSP*, 2021.

[23] J. Nozaki and T. Komatsu, "Relaxing the conditional independence assumption of ctc-based asr by conditioning on intermediate predictions," in *Proc. INTERSPEECH*, 2021.

[24] J. Lee, J. Kang, and S. Watanabe, "Layer pruning on demand with intermediate ctc," in *Proc. INTERSPEECH*, 2021.

[25] J. W. Yoon, B. J. Woo, S. Ahn, H. Lee, and N. S. Kim, "Inter-kd: intermediate knowledge distillation for ctc-based automatic speech recognition," in *Proc. IEEE SLT*, 2022.

[26] M. Ott, S. Edunov, A. Baevski, A. Fan, S. Gross, N. Ng, D. Grangier, and M. Auli, "Fairseq: a fast, extensible toolkit for sequence modeling," in *Proc. NAACL*, 2019, p. 48–53.

[27] S. Kaya, Y. Hong and T. Dumitras, "Shallow-deep networks: understanding and mitigating network overthinking," in *Proc. ICML*, 2019.

[28] H. Chang, S. Yang, and H. Lee, "Distilhubert: speech representation learning by layer-wise distillation of hidden-unit bert," in *Proc. ICASSP*, 2022.

[29] A. Fan, E. Grave, and A. Joulin, "Reducing transformer depth on demand with structured dropout," in *Proc. ICLR*, 2020.