Flexi-Transducer: Optimizing Latency, Accuracy and Compute for Multi-Domain On-Device Scenarios

Jay Mahadeokar, Yangyang Shi*, Yuan Shangguan*, Chunyang Wu*, Alex Xiao, Hang Su, Duc Le, Ozlem Kalinli, Christian Fuegen, Michael L. Seltzer

Facebook AI, USA
jaym@fb.com

Abstract

Often, the storage and computational constraints of embedded devices demand that a single on-device ASR model serve multiple use-cases / domains. In this paper, we propose a Flexible Transducer (FlexiT) for on-device automatic speech recognition to flexibly deal with multiple use-cases / domains with different accuracy and latency requirements. Specifically, using a single compact model, FlexiT provides a fast response for voice commands, and accurate transcription but with more latency for dictation. In order to achieve flexible and better accuracy and latency trade-offs, the following techniques are used. Firstly, we propose using domain-specific altering of segment size for Emformer encoder that enables FlexiT to achieve flexible decoding. Secondly, we use Alignment Restricted RNNT loss to achieve flexible fine-grained control on token emission latency for different domains. Finally, we add a domain indicator vector as an additional input to the FlexiT model. Using the combination of techniques, we show that a single model can be used to improve WERs and real time factor for dictation scenarios while maintaining optimal latency for voice commands use-cases.

Index Terms: Speech recognition, RNN-T, Transformers

1. Introduction

On-device automatic speech recognition (ASR) models have been enabled on many embedded devices, including mobile phones, smart speakers, and watches [1–3]. On one hand, on-device ASR eliminates the need to transfer audio and recognition results between devices and a server, thus enabling fast, reliable, and privacy-preserving speech recognition experiences. On the other hand, these devices operate with significant hardware constraints: e.g., memory, disk space, and battery consumption. Moreover, the embedded ASR models often serve multiple applications: e.g., video transcription, dictation, and voice commands. Each of these applications has its latency and accuracy requirements. For example, voice assistants demand an ASR model with low latency to respond to user queries as fast as possible. While server-based ASR might rely on running different models for different applications – more compact models for voice assistants and big, semi-streaming models [4,5] for dictation – the on-device environment prohibits such practice. Due to hardware constraints, and varied requirements of different applications optimizing the model size, compute and accuracy of one single ASR model becomes challenging. In this paper, we take a close look at the scenario where device constrained ASR model needs to be optimized for two different use-cases.

The first use case is voice commands, where the latency requirement is strict. Users expect immediate device responses when they ask the speech assistant to turn on the lights or play a song. The second use case uses the ASR model for dictation or audio transcription, where accuracy is more important than the model’s latency. Recurrent Neural Network Transducer (RNN-T) framework [1,6] is widely adopted to provide streaming ASR transcriptions for both voice commands [7,8] and dictation applications [9,10].

We focus on Emformer model [11] as an audio encoder for RNN-T, which uses both contextual audio information (in the form of an audio chunk) and future audio context (in the form of model look-ahead). A larger model look-ahead permits the model to access more future context and optimize ASR accuracy but hurts model latency.

This paper proposes a Flexi-Transducer (FlexiT) model that answers the requirements of two streaming ASR use-cases while still staying as one compact model. Further we also show that larger look-ahead enables improve the compute / real time factor trade-offs which help battery consumption.

2. Related Work

Inspired by the successful application of transformer [12], many works in ASR also adopted transformer across different model paradigms, such as the hybrid systems [13–16], the encoder-decoder with attention [17–21] and the sequence transducers [5,22,23]. In this work, we follow the neural transducer paradigm using Emformer [11,16] and alignment restricted transducer loss [24].

Many ASR applications demand real-time low latency streaming. The block processing method [11,16,25] with attention mask modifies the transformer to support streaming applications. In the block processing, self-attention’s receptive field consists of one fixed-size chunk of speech utterance and its historical context and a short window of future context. However, the fixed chunk size limits the encoder’s flexibility to trade-off latency, real-time factor, and accuracy.

A unified framework is proposed in [26] to train one single ASR model for both streaming and non-streaming speech recognition applications. In [4] cascaded encoders are used to build a single ASR model that operates in streaming and non-streaming mode. These approaches support one fixed latency for the streaming use-case, and the other use-case is strictly non-streaming. In this paper, we tackle scenarios to support two different streaming ASR use-cases with different latency constraints.

More flexible latency is achieved by [27–29] where, in the training phase, the model is exposed to audio context variants up to the whole utterance length. In [27], the authors show that asynchronous revision during inference with convolutional encoders can be used to achieve dynamic latency ASR. The context size selection proposed in these works is purely random during training. In this work, besides random selection, we also

*Equal Contribution
explore context size selection based on a priori knowledge about the targeted use cases (domains). The use of domain information to improve the ASR performance of a single model being used to serve different dialects, accents, or use-cases has been studied previously in [10, 30, 31].

For RNN-T models, both the encoder’s context size and the potential delay of token emissions contribute to model latency. It is well known that streaming RNN-T models tend to emit ASR tokens with delay. Techniques like Ar-RNN-T [24], Fast-emit [32], constrained alignment approach [33, 34] or late injection of domain vectors in an Emformer layer [11] can provide flexible latency for decoding and, at the same time, as suggested by our results in Section 5.3, helps to improve the speech recognition model’s robustness.

3.2. Adding Domain Vector in Emformer Encoder
To improve the ASR model’s capability to learn domain-specific features, we append a domain vector to inputs of each layer of the Emformer encoder. The domain vector is simply represented by using 1-hot representation [30], the value of which depends on whether the training sample comes from $V_{cmd}$ or Dictation domain. More concretely, as illustrated in Figure 1 let $D_{vec}$ denote the 1-hot domain vector representation. We concatenate $D_{vec}$ to all components of $X_{l}$, to obtain concatenated input vectors $[L_{i,j,v}^{n}, C_{i,j,v}^{n}, R_{i,v}^{n}]$. These are used as inputs to the $n^{th}$ Emformer layer while training.

3.3. Domain Specific Alignment Restrictions Loss
In [24] using pre-computed token level alignment information, configurable thresholds $b_{i}$, $b_{r}$ are used to restrict the alignment paths used for RNN-T loss computation during training. Note that the right-buffer $b_{r}$ can be made stricter to ensure earlier token emissions, but stricter $b_{r}$ also leads to increased WER. In this work, our goal is to optimize the WERs for the domain while maintaining low latency for the $V_{cmd}$ domain. Therefore, we propose using domain-specific alignment restriction thresholds while optimizing the loss. We analyze domain-specific WER and token emission latency in Section 5.

4. Experimental Setup
4.1. Datasets
4.1.1. Training Data
We run our experiments on data-sets that contains 20K hours of human-transcribed data from 2 different domains.

**Voice Commands** ($V_{cmd}$) dataset combines two sources. The first source is in-house, human transcribed data recorded via mobile devices by 20k crowd-sourced workers. The data is deidentified and aggregated with personally identifiable information (PII) removed. We distort the collected audio using simulated reverberation and add randomly sampled additive background noise extracted from publicly available videos. The second source came from 1.2 million voice commands (1K hours), sampled from production traffic with PII removed, audio deidentified and aggregated, and morphed. Speed perturbations [36] are applied to this dataset to create two additional training data at 0.9 and 1.1 times the original speed. We applied distortion and additive noise to the speed perturbed data. From the corpus, we randomly sampled 10K hours.

**Dictation** (open-domain) dataset consists of 13K hours of data sampled from English public videos that are deidentified and aggregated with PII removed and annotator transcribed. We first apply the same above-mentioned distortions and then randomly sample 10k hours of the resultant data.

4.1.2. Evaluation Datasets
For evaluation, we use the following datasets, representing two different domains:

**Voice Commands** evaluation set consists of 15K hand-transcribed deidentified and aggregated utterances from volunteer participants as part of an in-house pilot program.

**Dictation** evaluation set consists of 66K hand-transcribed deidentified and aggregated utterances from vendor collected data where speakers were asked to record unscripted open domain dictation or voice conversations.
4.2. Evaluation Metrics

To measure the model’s performance and analyze trade-offs, we track the following metrics:

Accuracy: We use word-error-rate (WER) to measure model accuracy on evaluation sets. Note that we measure the WERs for dictation domain without end-pointer and the $V_{cmd}$ domain with end-pointer. We also keep track of the deletion errors (DEL), which are proportional to early cutoffs in $V_{cmd}$.

Latency: We measure model latency on $V_{cmd}$ domain using following metrics:

1. **Token Finalization Delay (FD)**: as defined in [24] is the audio duration between the time when user finished speaking the ASR token, and the time when the ASR token was surfaced as part of 1-best partial hypothesis, also referred as emission latency in [26], or user-perceived latency in [37].

2. **Endpointing Latency (L)**: [24, 38] is defined as the audio time difference between the time end-pointer makes endpointing decision and the time user stops speaking.

   We track the Average Token Finalization Delay and Average Endpointing latency ($L_{AVG}$) metrics. In all experiments, we use a fixed neural end-pointer (NEP) [38], running in parallel to ASR being evaluated every 60ms to measure $V_{cmd}$ domain metrics. A detailed study with other end-pointing techniques besides NEP (static, E2E [39]) is beyond the scope of this paper.

ASR Compute: On device power consumption / battery usage is usually well co-related with the amount of compute being used by the ASR model. We use the real time factor (RTF) measured on a real android device as an indirect indicator of the of the model’s compute usage.

4.3. Experiments

We use the RNN-T model with Emformer encoder [40], LSTM with layer norm as predictor, and a joiner with 45M total model parameters. As inputs, we use 80-dim log Mel filter bank features at a 10 ms frame rate. We also apply SpecAugment [28] without time warping to stabilize the training. We use a stride of 6 and stack 6 continuous vectors to form a 480 dim vector projected to a 512 dim vector using a linear layer. The model has 10 Emformer layers, each with eight self-attention heads and 512 dimension output. The inner-layer has a 2048 dim vector using a linear layer. The model without time warping to stabilize the training. We use a stride 4.

2. **Random / Domain Specific Emformer Context:** We analyze randomly selected $EmfCtx$ during training as described in 3.1. Context size is randomly selected from 120ms to 1200ms. As shown later in Section 5 we find domain specific Ar-RNN-T $b_r$ of 420, 900ms achieves best results. Therefore, to reduce the number of combinations in this experiment, we fix $b_r$ of 420, 900ms and run experiments $R_1$ and $R_2$ without and with use of Domain vector respectively. Lastly, we analyze using domain specific $EmfCtx$ during training using domain-specific Ar-RNN-T $b_r$ of 420, 900ms. Corresponding experiments $S_1$ and $S_2$ without and with Domain vector respectively.

Inference: We always evaluate models such that for each domain, the training time $EmfCtx$ matches the context provided to the encoder while doing inference. Only exception being Random Emformer Context experiments $R_1$ and $R_2$ where we use inference context size of 120ms, 600ms for $V_{cmd}$ and dictation domains respectively.

Figure 2: Dictation WERs and $V_{cmd}$ token finalization delay tradeoffs for various experiments. The labels in the plot show the Ar-RNN-T $b_r$ used for the experiment. Increasing $b_r$, as well as Emformer context improves WER, but degrades latency. Methods ($R_2$, $S_2$) achieve best trade-offs.

5. Results and Analysis

5.1. Emformer Context and Ar-RNN-T Thresholds

Figure 2 demonstrates the trade-offs as we vary Emformer context and the Ar-RNN-T right-buffer $b_r$. Dictation WER improves as we increase the Emformer context in experiments $B_1$-$B_3$. Similarly, for a fixed Emformer context, WERs improve with larger Ar-RNN-T $b_r$ parameter. Note that only hand-picked variants are detailed in Table 1.
Table 1: Fixed Emformer Context without Domain Vector: Dictation WERs improve with larger Emf Ctx. $V_{cmd}$ WERs and Deletions with neural endpointer, also increase due to increased $\text{Avg}(FD)$.

<table>
<thead>
<tr>
<th>Emf Ctx</th>
<th>AR $b_r$</th>
<th>Dict WER</th>
<th>$V_{cmd}$ WER</th>
<th>$V_{cmd}$ DEL</th>
<th>Avg FD</th>
<th>$L_{Avg}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$B_1$</td>
<td>120</td>
<td>420</td>
<td>15.4</td>
<td>6.7</td>
<td>2.8</td>
<td>148</td>
</tr>
<tr>
<td>$B_2$</td>
<td>300</td>
<td>600</td>
<td>13.8</td>
<td>7.4</td>
<td>3.6</td>
<td>272</td>
</tr>
<tr>
<td>$B_3$</td>
<td>600</td>
<td>900</td>
<td>13.2</td>
<td>9.7</td>
<td>5.4</td>
<td>470</td>
</tr>
<tr>
<td>$C_2$</td>
<td>300</td>
<td>420/600</td>
<td>13.7</td>
<td>7.5</td>
<td>3.6</td>
<td>271</td>
</tr>
<tr>
<td>$C_3$</td>
<td>600</td>
<td>420/900</td>
<td>13.2</td>
<td>10.8</td>
<td>6.4</td>
<td>483</td>
</tr>
</tbody>
</table>

On the other hand, $V_{cmd}$ domain’s $L_{avg}$ also increases. To achieve low $L_{Avg}$ throughout experiments, which is important for a better user experience, we use a fixed NEP. Delays in token emissions result in more early cuts and increased deletion errors as shown in Table 1. To achieve best latency and simultaneously reduce early cuts, we must maintain a smaller Emformer context and maintain a strict Ar-RNN-T $b_r$ parameter. Therefore, experiments for random and domain-specific Emformer context are performed with $b_r = 420$ and context 120.

5.2. Domain Vector and Domain specific Ar-RNN-T

Table 2: Fixed Emformer Context with Domain Vector: With Domain vector $D_1$-$D_3$ achieve better dictation WERs compared to $B_1$-$B_3$. Further, with Domain specific Ar-RNN-T, $E_2,E_3$ achieve better latency compared to $D_2,D_3$.

<table>
<thead>
<tr>
<th>Emf Ctx</th>
<th>AR $b_r$</th>
<th>Dict WER</th>
<th>$V_{cmd}$ WER</th>
<th>$V_{cmd}$ DEL</th>
<th>Avg FD</th>
<th>$L_{Avg}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$D_1$</td>
<td>120</td>
<td>420</td>
<td>14.0</td>
<td>6.8</td>
<td>2.9</td>
<td>159</td>
</tr>
<tr>
<td>$D_2$</td>
<td>300</td>
<td>600</td>
<td>12.8</td>
<td>7.7</td>
<td>3.8</td>
<td>297</td>
</tr>
<tr>
<td>$D_3$</td>
<td>600</td>
<td>900</td>
<td>12.4</td>
<td>10.5</td>
<td>6.2</td>
<td>543</td>
</tr>
<tr>
<td>$E_2$</td>
<td>300</td>
<td>420/600</td>
<td>12.8</td>
<td>7.23</td>
<td>3.28</td>
<td>265</td>
</tr>
<tr>
<td>$E_3$</td>
<td>600</td>
<td>420/900</td>
<td>12.5</td>
<td>9.6</td>
<td>5.7</td>
<td>464</td>
</tr>
</tbody>
</table>

We observe that providing domain vector in encoder improves the WERs for dictation domain in general for all experiments as shown in Table 2, which is consistent with previous works [10,30]. Unfortunately, the WER improvements come in tandem with an increased average FD of $V_{cmd}$ when the models are trained with domain vector (comparing experiments $B_1$-$B_3$ and $D_1$-$D_3$). We hypothesize that this is because in the absence of stricter AR-RNNT $r_v$ restrictions for the $V_{cmd}$ domain, the model learns to delay token emissions to improve accuracy.

Ar-RNN-T helps achieve fine grained control on token delays [24]. However, in multi-domain setting, comparing $C_2,C_3$ to $B_2,B_3$, we observe that simply imposing domain specific AR-RNN-T thresholds does not improve $V_{cmd}$ FD. The use of domain vector, alongside domain specific Ar-RNN-T thresholds, enables us to achieve a more refined control over $V_{cmd}$ domain’s FD. This is demonstrated in Table 2 where $D_1$-$D_3$ have larger Avg(FD) compared to $B_1$-$B_3$, but models $E_2$-$E_3$ learn to explicitly emit $V_{cmd}$ domain tokens earlier than $C_2$-$C_3$.

5.3. Random / Domain Specific Emformer Context

Results from random context training, $R_1$ suggests that in the absence of domain information, the model achieves worse trade-offs than $V_{cmd}$ FD of $B_1$ and Dictation WER of $B_3$. In $R_2$ adding domain vector, improves the WER for dictation domain significantly which is consistent with Section 5.2. Overall, $R_2$ achieves better trade-offs for optimizing both use-cases.

Table 3: Random and Domain Specific Emformer Context: Experiments use 420 / 900 Ar-RNN-T $r_v$ parameters and 120 / 600 EmCtx during inference.

<table>
<thead>
<tr>
<th>Emf Ctx</th>
<th>$D_{vec}$</th>
<th>Dict WER</th>
<th>$V_{cmd}$ WER</th>
<th>$V_{cmd}$ DEL</th>
<th>Avg FD</th>
<th>$L_{Avg}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R_1$</td>
<td>Random</td>
<td>No</td>
<td>13.6</td>
<td>7.2</td>
<td>3.1</td>
<td>175</td>
</tr>
<tr>
<td>$R_2$</td>
<td>Random</td>
<td>Yes</td>
<td>12.7</td>
<td>7.1</td>
<td>3.1</td>
<td>162</td>
</tr>
<tr>
<td>$S_1$</td>
<td>120/600</td>
<td>No</td>
<td>12.5</td>
<td>7.7</td>
<td>3.3</td>
<td>185</td>
</tr>
<tr>
<td>$S_2$</td>
<td>120/600</td>
<td>Yes</td>
<td>12.6</td>
<td>7.0</td>
<td>2.9</td>
<td>157</td>
</tr>
</tbody>
</table>

In Figure 3 we analyze the RTF of FlexiT models. For $R_2$ we evaluate the RTF and WERs while varying inference context size. We observe that the RTF reduces with larger context size, which is mainly because of improved batching inside Emformer layers across time dimension.

6. Conclusion

This paper proposed a single Flexi-Transducer (FlexiT) model that supports domain-dependent trade-off of latency and accuracy. The domain-specific or random context modeling is achieved jointly via segment size altering operation for encoder and the domain vector. Ar-RNN-T loss imposes a domain-specific constraint to limit the token emission latency for different domains. Using the combination of techniques, we achieve better WER, RTF and latency trade-offs when a single model supports multiple streaming ASR use-cases.

7. Acknowledgment

We’d like to thank Rohit Prabhavalkar and Jiatong Zhou for valuable discussions and advice.
8. References


