Double Decoder: Improving latency for Streaming End-to-end ASR Models

Riqiang Wang*, **Shreekantha Nadig***, **Daniil Kulko, Simon Vandieken,**
Chia-Tien Chang, Seyyed Saeed Sarfjoo, Jonas Robertson
Dialpad, Canada
{riqiang.wang, shreekantha.nadig, daniel.kulko, svandieken,
karol.chang, saeed.sarfjoo, jonas}@dialpad.com

Abstract

In this paper, we propose a novel decoding algorithm for streaming End-to-end (E2E) automatic speech recognition (ASR) models, the double decoder. By comparing it with existing decoding algorithms, we argue that this new method achieves a better balance between word error rate, latency and streaming stability, notably by reducing latency without WER degradation but with degradation in stability. The algorithm also does not require any change in model weights. We show results on a Conformer-CTC model trained on the LibriSpeech dataset, which indicates that the proposed double decoder maintains the same WER as buffered decoding while reducing the latency by the size of the look-ahead used in decoding. We also show that the proposed method is generalizable. For example, we apply it to the Zipformer-CTC-Transducer model, which traditionally uses the default decoding, and it achieves better WER and latency at the expense of increased computational cost.

1 Introduction

Real-time automatic speech recognition (ASR) systems are a critical part of many of the industrial speech understanding applications. For example, delays in the ASR cause delays in downstream natural language processing (NLP) tasks. Other factors like accuracy, measured by word error rate (WER), and streaming stability (Shangguan et al., 2020a) are also important factors to ensure a good overall user experience in real-time language processing systems.

While deep neural network-hidden Markov model (DNN-HMM) hybrid ASR models have traditionally been used for efficient streaming ASR inference, many advancements in end-to-end (E2E) ASR research have shown that E2E models have more performance potential. For instance, the connectionist-temporal-classification (CTC) loss (Graves et al., 2006) models longer context without pre-defined alignment and allows for low frame-rate decoding (Pundak and Sainath, 2016). RNN-Transducers (RNN-T) (Graves, 2012) jointly train an internal language model with an acoustic model, further improving the modelling capabilities of a single model. Attention encoder-decoder (AED) models such as the Listen-Attend-Spell (LAS) (Chan et al., 2016) models exploit the attention mechanism to model much longer context. With the introduction of Transformer (Vaswani et al., 2017) and its variants (Gulati et al., 2020), E2E models have been pushing the state-of-the-art WER on various ASR datasets (Chen et al., 2023).

However, for streaming ASR, the search for a good trade-off between latency, stability and accuracy remains an open problem. The aforementioned E2E models achieve good WER by incorporating more audio context, and exploiting more parameters, translating to higher latency and more compute cost. For example, an AED model for ASR requires the entire input sequence before starting to generate output tokens; the Transformer model uses absolute positional embeddings, limiting its applicability to streaming real-time ASR systems (Dai et al., 2019).

In our study, we propose an algorithm we call the *double decoder*. In essence, we run the decoder twice on the encoder outputs - once on the *look-ahead* or the most recent chunk of audio to speculatively display low latency results, then once on the chunk behind it, with a delay. No change in the weights of the model is needed. The algorithm builds on the existing buffered decoding method, which is designed for inference time and addresses the limitation of Transformers, which typically require the complete sequence for inference.

This simple algorithm has not been published at the time of writing and we believe many ASR engineers can benefit from this method. We show that our proposed method can improve the suitability of

*Equal contribution.

E2E models for streaming ASR by achieving a better balance between latency, accuracy, and stability. In particular, it enables streaming for Conformer-CTC (Graves et al., 2006; Gulati et al., 2020) and other CTC models, which would otherwise have undesirable performance for streaming. This is achieved by reducing the latency while keeping the WER of the buffered decoding method. We also show an example with the Zipformer-CTC-Transducer model where we reduce the WER and latency when we use the right most part of the original chunk size as look-ahead.

2 Traditional Decoding Algorithms for Streaming

There are a few assumptions and definitions to clarify for the description of decoding algorithms. For simplicity, we assume there is no sub-sampling so the model stride at output time is the same as the input. We also define *model* to be the neural network outputting log probabilities for each frame, whereas the *external decoder* is the external decoding algorithm, not to be confused with the decoder inside an AED model. Its forward function takes the log probabilities from the model given the latest input chunk, and outputs the text hypothesis for the entire audio so far. It updates its internal state for caching the history: for greedy decoding, the latest text output is appended to the text history; for beam search, the beam gets updated.

2.1 Default Method

The most straightforward default method is to decode using the same duration for input length and step size. This can be sufficient if the model incorporates an RNN component, where the left context is inherently represented in the RNN cell state. In this method, no computation is wasted as every input frame passes through the model exactly once. Conceptually, Figure 1a shows this decoding algorithm. At each time step t, the model reads input chunk x_t and outputs the log probabilities, and the external decoder processes them into text.

2.2 Buffered Decoding

For Transformer or Transformer-like models such as the Conformer (Gulati et al., 2020), every layer attends to the exact same context as the input audio. It does not incorporate history context directly, therefore we need to explicitly include history audio h_t besides x_t as input. Moreover, we have seen in the literature (Moritz et al., 2020) as well as empirically that adding the right context (look-ahead l_t) improves WER. This is called buffered decoding as we need to keep h_t, x_t and l_t together in a buffer, run the model on everything at each time frame and only keep the log probabilities for x_t. This method has been the default streaming method for Transformer-like models[1], and it is illustrated in Figure 1b. In practice, l_t is the actual latest chunk. To only output the results for x_t means a constant additional delay of $|l_t|$ after getting x_t.

3 Proposed Decoding Algorithm

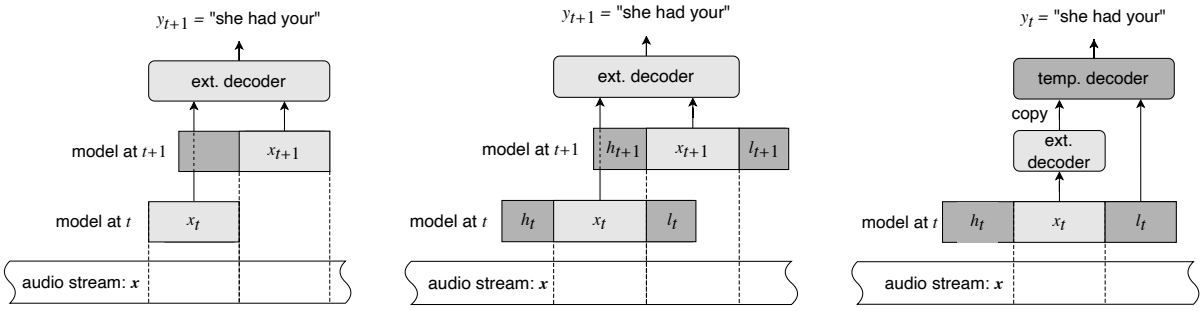
Our proposed method, the *double decoder*, builds on the buffered decoding with one simple modification: we use a temporary decoder on top of the main external decoder. For each time step t, the model produces output log probabilities p_t for h_t, x_t and l_t, where only the log probabilities given x_t are passed into the main external decoder. The state of the external decoder gets updated, then it is copied to be the state of the temporary decoder. Subsequently, the log probabilities given l_t are decoded by a temporary decoder to produce the partial hypothesis y_t, which is the text hypothesis given in the audio so far. The partials get replaced with new ones at each time step. See Algorithm 1 and Figure 1c.

Algorithm 1: Double Decoder

input : audio stream $\boldsymbol{x}$, same $|h_t|$, $|x_t|$, $|l_t|$ for every t
output : partial hypotheses $\boldsymbol{y}$
Initialize model, ext_decoder, $t \leftarrow 0$;
while $\boldsymbol{x}$ *not ended*:
  Get latest chunks h_t, x_t, l_t from $\boldsymbol{x}$;
  $p_t \leftarrow$ model.forward($h_t + x_t + l_t$);
  $t'_0 \leftarrow$ start time of x_t;
  $t'_1 \leftarrow$ end time of x_t;
  ext_decoder.forward(p_t *from* t'_0 *to* t'_1);
  temp_decoder $\leftarrow$ copy(ext_decoder);
  $y_t \leftarrow$ temp_decoder.forward(p_t *from* t'_1 *onward*);
  $t \leftarrow t + |x_t|$;

Using this algorithm, we ensure that the final

[1]Example code using the NeMo toolkit at https://github.com/NVIDIA/NeMo/tree/main/examples/asr/asr_chunked_inference



(a) Default decoding: decoding each current chunk

(b) Buffered decoding: decoding with look-ahead

(c) *Double decoder*: using a temporary decoder for outputting the look-ahead

Figure 1: Three decoding methods. The chunks x_t and the context h_t and l_t represents what the mdoel (encoder) has access to, and we skip the steps where the model encodes the audio and output log probabilities. The *copy* action copies the internal state of the external (ext.) decoder to the temporary (temp.) decoder, which includes the information about the history. Notice that y_t from the *double decoder* outputs the same text for y_{t+1} from the other two methods. This is to demonstrate that the *double decoder* can output the same text earlier than the other methods. Light grey colour means the result is cached and used for the final output, while dark grey means the content is discarded after one time step.

hypothesis is made up of the outputs given x_t where the accuracy is better. The hypothesis given the look-ahead l_t is only displayed but never saved into the main decoder's state. At the same time, we avoid waiting for getting l_t purely as context, compared to buffered decoding.

4 Related Work

Many previous studies on streaming ASR have focused on improving the internal efficiency of the models. Modifications to the Transformer are proposed to make it more suited for real-time streaming ASR - this includes relative positional embeddings (Shaw et al., 2018), caching and reusing intermediate network states (Dai et al., 2019), limiting the look-ahead acoustic context (Noroozi et al., 2024) and using time-restricted self-attention (Moritz et al., 2020). Most of the proposed methods require changes to the original ASR architecture and re-training of the model, whereas our proposed method works on pre-trained models.

Other studies take a more holistic view of optimizing for streaming ASR, considering the trade-off between context size, latency, accuracy, and streaming stability, i.e. whether the displayed words gets revised as the transcript becomes more complete (Bruguier et al., 2016; Moritz et al., 2020; Shangguan et al., 2020a,b, 2021). In Shangguan et al. (2020a), the authors proposed increasing partial emission latency, unifying text normalization for different domains and using domain ID to improve the stability of partials. Whereas in Bruguier et al. (2016), the authors proposed an algorithm to select more stable hypotheses during beam search decoding. We take inspiration from these studies and analyze the stability of our proposed method.

In terms of architectures used for streaming, recent studies of streaming ASR have focused on improving RNN-T models as they showed superior WER performance. To further improve the final WER, earlier efforts use a second pass LAS model to do beam search or to rescore the RNN-T hypotheses (Sainath et al., 2020, 2019). He et al. (2019); Narayanan et al. (2021); Sainath et al. (2021); Shangguan et al. (2020b) use cascaded encoders, with a causal encoder which passes its output to another non-causal encoder. This eliminates the need for training a separate re-scoring model, unifying streaming and non-streaming models while improving the final transcription. Related, Yu et al. (2021) presented a more detailed exploration of a unified streaming and non-streaming RNN-T model.

Our proposed method is similar to the cascaded encoders (Narayanan et al., 2021), besides the fact that we make use of only a single model with a single encoder, saving the compute of running an extra encoder. It can be seen as a special case of cascaded encoders where the second encoder is just copying the embeddings from the first one. However, since we are using only one model, the final WER is bound by the offline model performance, whereas the cascaded encoders' second pass improves WER.

A similar idea of fast-slow two-head decoding is

explored for older hybrid ASR in Li et al. (2020). However, Li et al. (2020)'s method also involves changing the model architecture and training a second encoder. Our method is purely a decoding algorithm. It can be argued that their method is suitable for LSTM models without explicit access to look-ahead data, and our method is preferable for Transformer-like E2E models.

Finally, as many of the techniques mentioned above focus on RNN-T, we show that our decoding algorithm can be applied to not only RNN-Ts but also CTC models. Models trained with CTC are typically outperformed by RNN-T or AEDs in WER, but they have a lower real-time factor (RTF) (Zhang et al., 2021), making them suitable candidates for streaming ASR where latency is crucial.

5 Evaluation Metrics

5.1 Accuracy

We use WER to measure the accuracy performance of the streaming inference. For WER, if we use the same context (buffer) size, we expect the WER to be the same for buffered decoding and *double decoder*, since the chunks used to generate the final hypothesis are exactly the same.

5.2 Streaming Stability

To discuss streaming stability, we first define the relevant terms used: A partial hypothesis is a text sequence outputted by the ASR system before the end of an utterance is reached. A final hypothesis is produced after end-pointing - either the end of the audio is reached or the end-pointer detects the boundary of the utterance. As we replace previous partials on the display with new partials, the words may get revised which can be seen as the effect of instability. It should be noted that we do not measure the WER of partial hypotheses for the stability since both the reference and the hypothesis are incomplete and changing.

We employ the unstable partial word ratio (UPWR) introduced in Shangguan et al. (2020a) to measure instability. To calculate UPWR, we sum up the revised or unstable tokens in each partial hypothesis when compared to the next partial hypothesis, then divide it by the number of tokens in the final. The closer UPWR is to 0, the more stable the system. As an example, for the partial hypotheses produced by the *double decoder* in Table 1, we have three unstable tokens, *but*, *please* and *him*. The final hypothesis has 10 tokens, therefore UPWR = 0.3. It should be noted that we cannot directly compare the results of partial stability with the original paper, since they use formatted transcripts.

As we compare with the buffered decoding method, we expect stability to degrade, since we are incorporating more speculative text outputs in the hypothesis. As the buffered decoding method always takes the middle chunk which forms the final hypothesis, we expect the UPWR to be almost 0. On the other hand, *double decoder* will produce mostly non-zero scores.

5.3 Latency

To simplify the calculation of latency differences, we consider three parts of latency that make up the total latency in a continuous, streaming ASR system, presuming I/O and other system latency to be negligible. Firstly, there is the delay for accumulating audio stream T_a, composed of $|l_t + x_t|$ which makes up a constant delay. It should be noted that the history size $|h_t|$ corresponds to a one-time delay at the start of the audio stream. However, by padding the beginning of the audio with artificial silence, we can effectively reduce this delay to zero.

Furthermore, for every audio input x, we consider the model forward latency $T_m(x)$, which is the inference time for producing frame-wise log probabilities. Independently, we consider decoding latency $T_d(x)$, which is the time taken by the external decoder to produce one partial hypothesis, given the probabilities.

In this context, we can calculate the theoretical latency of buffered decoding as $|l_t + x_t| + T_m(h_t + x_t + l_t) + T_d(x_t)$. Given our algorithm, if we compare it with the buffered decoding approach, we can see that by outputting the hypotheses for the right contexts of each time step, we reduce the theoretical latency to $|x_t| + T_m(h_t + x_t + l_t) + T_d(x_t + l_t)$, where $\Delta T = |l_t| - T_d(l_t)$. We show ΔT in Section 7.

6 Experiment Setup

We have chosen two models to illustrate the effectiveness of the algorithm with or without explicit historical context. First we apply it to the small Conformer-CTC model from NVIDIA NGC[2] to

[2] https://catalog.ngc.nvidia.com/orgs/nvidia/teams/nemo/models/stt_en_conformer_ctc_small_ls

Table 1: Sample comparison of partials generated by buffered decoding vs. double decoder decoding. It is taken from Test-clean and the context size is 1.2 seconds.

Buffered partial hypotheses	*Double decoder* partial hypotheses
	i never
i never knew	i never knew of
i never knew but	i never knew but
i never knew but one ma	i never knew but one man
i never knew but one man who coul	i never knew but one man who could ever
i never knew but one man who could ever pleas	i never knew but one man who could ever please him
i never knew but one man who could ever pleasing	i never knew but one man who could ever pleasing

show its effectiveness comparing to buffered decoding. The model has 13M parameters. For the decoding method, we utilize a streaming beam search decoder built on PyCTCdecode[3]. For the beam search, we use a 3-gram language model (LM) with LM weight of 0.2, insertion penalty of 0.3. The beam width is set to 100, and max active tokens is 20. When we decode with buffered decoding or the *double decoder*, the middle chunk $|x_t|$ is 0.6 s. We evaluate WER, latency and streaming stability for Conformer-CTC given different context sizes.

We also show that our algorithm can be applied to a Zipformer-CTC-Transducer model to improve its WER against default decoding. We trained a 66M parameter Zipformer-medium(Yao et al., 2024) streaming cache-aware model. The final loss for the model was computed as a weighted sum of a CTC (Yao et al., 2023) and pruned RNN-T loss (Kuang et al., 2022) using K2[4] with CTC weight of 0.2. The decoding method for this model is greedy decoding. The training recipe can be found on the Icefall repository[5]. The model does not require history input, i.e. $|h_t| = 0$. Instead, it keeps a cached state of intermediate layers for the past history.

Both models are trained on the 960 hours of LibriSpeech data(Panayotov et al., 2015), and are evaluated on LibriSpeech test-sets. Latency is computed on a virtual machine using intel Cascade Lake CPUs.

7 Results

7.1 Results on Conformer-CTC

Table 2 shows the WER for both buffered decoding and *double decoder*, as well as the latency difference between the two decoding methods. As discussed in Section 5.1, WER is the same for buffered decoding and *double decoder*. We see that on LibriSpeech Test-clean and on Test-other, WER decreases by approximately the same rate. This demonstrates the benefit of using longer context, whose effect is consistent across different testing conditions.

Regarding latency, as discussed in Section 5.3, *double decoder* reduces latency from buffered decoding by $|l_t| - T_d(l_t)$. We can clearly see from Table 2 $T_d(l_t)$ is an order of magnitude smaller than the look-ahead size $|l_t|$, and does not increase linearly with $|l_t|$. Therefore, we can conclude that with the studied context size the *double decoder* always provides a latency reduction of approximately the same duration as the look-ahead size. As the WER decreases when the context size increases, we can effectively achieve better WER while maintaining latency, or reduce latency while maintaining WER, by using the *double decoder*.

Figure 2 and 3 shows the UPWR results on Test-clean and Test-other. We confirm that using the *double decoder* results in higher UPWR scores. However, there are other interesting trends with regards to partial stability. Firstly, we observe a downward trend for the buffered decoding as we increase the context size. This indicates an improvement in the quality of the middle chunk. At the same time, UPWR for *double decoder* decreases in Test-clean, but increases in Test-other, as we increase the context size. This indicates that in noisy conditions, the lookahead becomes much more unstable, reflected both in raw UPWR score and its variation. As the acoustic condition in real-life ASR applications are not always clean, this trend suggests that we cannot blindly increase the context size, but we need to combine this metric together with latency and WER for hyper-parameter tuning.

Another interesting difference between the two

[3] https://github.com/kensho-technologies/pyctcdecode

[4] https://github.com/k2-fsa/k2

[5] https://github.com/k2-fsa/icefall/tree/master/egs/Librispeech/ASR/zipformer

Table 2: WER (both buffered and *double decoder* have the same value) for the Conformer-CTC given different context sizes. $|x_t| = 0.6$ s. $|h_t|$ and $|l_t|$ are shown in the table. Latency reduction by employing *double decoder* compared to buffered decoding is calculated by $|l_t| - T_d(l_t)$ as shown in Section 5.3.

Context size (s)	$\|h_t\|, \|l_t\|$ (s)	Test-clean (%)	Test-other (%)	$T_d(l_t)$ (ms) / 95% CI
1.2	0.28, 0.32	11.84	23.76	4.4 / [3.1,5.2]
1.8	0.56, 0.64	5.61	13.65	3.4 / [1.9,4.0]
2.4	0.88, 0.92	4.53	11.22	**3.0 / [1.8,3.6]**
3.0	1.2, 1.2	4.04	10.06	3.0 / [1.8,3.7]
4.0	1.68, 1.72	**3.6**	**9.01**	3.2 / [1.8,4.3]

decoding algorithms becomes clear as we look into the raw partial hypotheses in text. In Table 1, we can clearly see that buffered decoding produces unfinished non-word tokens such as *ma*, *coul*. On the other hand, the unstable tokens produced by the *double decoder* are actual words, which account for the high UPWR. It is arguable which provides a better user experience. For instance, we can argue that the partial hypothesis provided by the *double decoder* with *please him* is more grammatically correct than *pleasing*. Future studies with different metrics more targeted to partial hypotheses' accuracy and user studies are still needed.

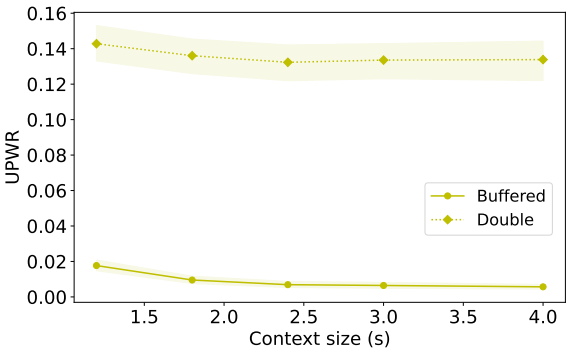


Figure 2: UPWR for LibriSpeech Test-clean, lower is better. The coloured band around the line is the 95% confidence interval.

7.2 Results on Zipformer-CTC-Transducer

As we have shown in the Conformer results that the trend in WER stays the same for Test-clean and Test-other, we feel that Test-clean is sufficient to capture the relevant metrics for the Zipformer demonstration. Table 3 shows the WER results of applying the *double decoder* on the cache-aware Zipformer model. Since this model incorporates a cache for history context, we typically decode with the default method. Comparing with row 1 in Table 2, we can see that this model achieves much better WER even with the default method.

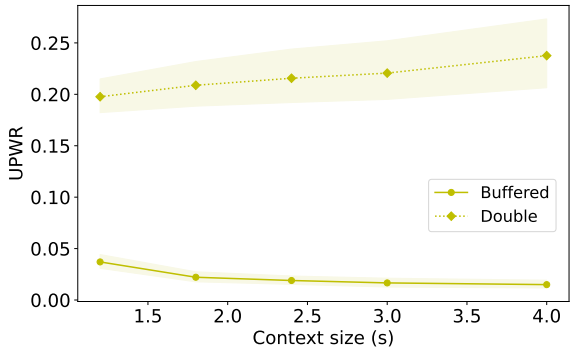


Figure 3: UPWR for LibriSpeech Test-other.

Nonetheless, the results highlights the importance of adding look-ahead context, using either buffered or *double decoding* in different ways. For example, if we compare the results with the same $|x_t|$, by adding 0.6 s of look-ahead, we reduce the WER from 4.45 to 2.98 for cache size of 1.28 s. Similarly, if we compare results given the same context size (see results on the same row), we see WER reduction. Lastly, if we compare row 3 to row 2 and row 6 in Table 3, we can see that for the *double decoder*, decreasing either the look-ahead or history by 0.6 s degrades WER by approximately the same amount, 0.2 for CTC and 0.1 for RNN-T decoding.

Both the buffered and *double decoder* utilizes a look-ahead, but the *double decoder* provides better latency than buffered decoding, given the small $T_d(l_t)$ results in Table 3, making it a better candidate for streaming applications.

Additionally, Table 3 highlights the suitability of *double decoder* for CTC models. We can see that CTC decoding generally provides worse WER, but the improvement from adding context is greater than RNN-T. CTC models also show smaller decoding overhead $T_d(l_t)$.

It should be noted, however, as we are keeping the context sizes constant between default decod-

Table 3: WER (%) on Test-clean of RNN-T greedy decoding for Zipformer-CTC-Transducer for different context sizes. For default decoding, $|x_t|$ is variable while $|l_t| = 0$, for buffered / double decoder, $|x_t|$ is fixed at 0.6 s. We also show $T_d(l_t)$ just as in Table 2.

Context size (s)		Default		Buffered/Double		$T_d(l_t)$ (ms)	
Cache	$\|x_t + l_t\|$	CTC	RNN-T	CTC	RNN-T	CTC	RNN-T
1.28	0.6	8.43	4.45	N/A	N/A	N/A	N/A
1.28	1.2	6.85	3.91	3.35	2.98	**0.06**	6.75
1.28	1.8	**4.99**	3.04	**3.16**	**2.87**	0.10	10.17
0.64	0.6	8.62	4.57	N/A	N/A	N/A	N/A
0.64	1.2	6.91	3.99	3.45	3.12	0.06	6.75
0.64	1.8	5.06	3.05	3.36	3.00	0.10	10.17

ing and *double decoder*, we reduce the step size $|x_t|$. It means the model is running at smaller intervals, therefore more times for the same duration of audio. The increase in computational cost is non-negligible and should be considered for real-life applications.

8 Conclusions

In this study, we introduce a simple addition to the buffered decoding algorithm, *double decoder* for improving streaming E2E models. Firstly, we show that the use of the *double decoder* improves the appeal of Conformer-CTC models for streaming. With the default or buffered method, it has either unacceptable latency or WER. With the *double decoder*, we reduce latency while maintaining low WER. Secondly, for Zipformer-CTC-Transducer, we show the importance of look-ahead context for further improving the WER. Given the same context size, we are able to achieve better WER and better latency. Given the latency benefits from the *double decoder*, we argue that it is the best method for incorporating look-ahead context. We also explore the side effect of this algorithm, for example, using streaming stability metric UPWR. We observe the degradation in streaming stability by using *double decoder*, and we argue that the context size cannot be too large in real-life noisy conditions. Similarly, we note that extra compute is needed to achieve the result for the Zipformer-CTC-Transducer. For future work, we will investigate whether there are other metrics to fully measure the user perceived readability of partial hypotheses, further improve stability and latency for the studied models, and further investigate the effect of the algorithm when it is applied to different architectures.

Limitations

The main limitation, as we have noted in the main text, is that streaming stability or computational cost worsens when WER or latency improves with our proposed method. Unfortunately the increase in computational cost for the Zipformer is unavoidable, since we are running the encoder at shorter intervals. However, the stability of the partial hypotheses can be further investigated and improved.

Furthermore, we have limited our study to a specific type of E2E models - the Transformer-like models such as the Conformer or the Zipformer. This is due to the fact that Transformer-like models generally achieves better accuracy but have poorer streaming suitability. It can be noted that the proposed method would not provide significant improvement on models such as the LSTM which does not make use of lookaheads or future context, unless it is a bidirectional LSTM. Additionally, we do not have comparisons with other types of fast-slow two-head methods, since their slow head typically involves training a different set of weight and is bound to achieve overall better WER but with more computational cost. We argue that our algorithm is more suitable for smaller-scale research or applications, while it does not achieve the most competitive WER scores.

Lastly, we note that we have used a small set of data for the experiments. Future work is needed to fully evaluate this algorithm on other larger models trained on a variety of datasets.

References

Antoine Bruguier, David Qiu, Trevor Strohman, and Yanzhang He. 2016. Flickering Reduction with Partial Hypothesis Reranking for Streaming ASR. In *2022 IEEE Spoken Language Technology Workshop (SLT)*, pages 38–45. IEEE.

William Chan, Navdeep Jaitly, Quoc Le, and Oriol Vinyals. 2016. Listen, attend and spell: A neural network for large vocabulary conversational speech recognition. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4960–4964.

William Chen, Brian Yan, Jiatong Shi, Yifan Peng, Soumi Maiti, and Shinji Watanabe. 2023. Improving Massively Multilingual ASR with Auxiliary CTC Objectives. In *ICASSP 2023 - 2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1–5.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc Le, and Ruslan Salakhutdinov. 2019. Transformer-XL: Attentive Language Models beyond a Fixed-Length Context. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2978–2988, Florence, Italy. Association for Computational Linguistics.

Alex Graves. 2012. Sequence Transduction with Recurrent Neural Networks. In *n International Conference on Machine Learning: Representation Learning Workshop*.

Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. 2006. Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks. In *Proceedings of the 23rd International Conference on Machine Learning - ICML '06*, pages 369–376. ACM Press.

Anmol Gulati, James Qin, Chung-Cheng Chiu, Niki Parmar, Yu Zhang, Jiahui Yu, Wei Han, Shibo Wang, Zhengdong Zhang, Yonghui Wu, and Ruoming Pang. 2020. Conformer: Convolution-augmented Transformer for Speech Recognition. In *Interspeech 2020*, pages 5036–5040.

Yanzhang He, Tara N. Sainath, Rohit Prabhavalkar, Ian McGraw, Raziel Alvarez, Ding Zhao, David Rybach, Anjuli Kannan, Yonghui Wu, Ruoming Pang, Qiao Liang, Deepti Bhatia, Yuan Shangguan, Bo Li, Golan Pundak, Khe Chai Sim, Tom Bagby, Shuo-yiin Chang, Kanishka Rao, and Alexander Gruenstein. 2019. Streaming End-to-end Speech Recognition for Mobile Devices. In *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6381–6385.

Fangjun Kuang, Liyong Guo, Wei Kang, Long Lin, Mingshuang Luo, Zengwei Yao, and Daniel Povey. 2022. Pruned RNN-T for fast, memory-efficient ASR training. In *Interspeech 2022, 23rd Annual Conference of the International Speech Communication Association, Incheon, Korea, 18-22 September 2022*, pages 2068–2072. ISCA.

Jinyu Li, Rui Zhao, Eric Sun, Jeremy H. M. Wong, Amit Das, Zhong Meng, and Yifan Gong. 2020. High-Accuracy and Low-Latency Speech Recognition with Two-Head Contextual Layer Trajectory LSTM Model. In *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 7699–7703.

Niko Moritz, Takaaki Hori, and Jonathan Le. 2020. Streaming Automatic Speech Recognition with the Transformer Model. In *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6074–6078.

Arun Narayanan, Tara N. Sainath, Ruoming Pang, Jiahui Yu, Chung-Cheng Chiu, Rohit Prabhavalkar, Ehsan Variani, and Trevor Strohman. 2021. Cascaded Encoders for Unifying Streaming and Non-Streaming ASR. In *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5629–5633.

Vahid Noroozi, Somshubra Majumdar, Ankur Kumar, Jagadeesh Balam, and Boris Ginsburg. 2024. Stateful Conformer with Cache-Based Inference for Streaming Automatic Speech Recognition. In *ICASSP 2024 - 2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 12041–12045.

Vassil Panayotov, Guoguo Chen, Daniel Povey, and Sanjeev Khudanpur. 2015. Librispeech: An ASR corpus based on public domain audio books. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5206–5210.

Golan Pundak and Tara N. Sainath. 2016. Lower Frame Rate Neural Network Acoustic Models. In *Interspeech*, pages 22–26.

Tara N. Sainath, Yanzhang He, Bo Li, Arun Narayanan, Ruoming Pang, Antoine Bruguier, Shuo-yiin Chang, Wei Li, Raziel Alvarez, Zhifeng Chen, Chung-Cheng Chiu, David Garcia, Alex Gruenstein, Ke Hu, Anjuli Kannan, Qiao Liang, Ian McGraw, Cal Peyser, Rohit Prabhavalkar, Golan Pundak, David Rybach, Yuan Shangguan, Yash Sheth, Trevor Strohman, Mirkó Visontai, Yonghui Wu, Yu Zhang, and Ding Zhao. 2020. A Streaming On-Device End-To-End Model Surpassing Server-Side Conventional Model Quality and Latency. In *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6059–6063.

Tara N. Sainath, Yanzhang He, Arun Narayanan, Rami Botros, Ruoming Pang, David Rybach, Cyril Allauzen, Ehsan Variani, James Qin, Quoc-Nam Le-The, Shuo-Yiin Chang, Bo Li, Anmol Gulati, Jiahui Yu, Chung-Cheng Chiu, Diamantino Caseiro, Wei Li, Qiao Liang, and Pat Rondon. 2021. An Efficient Streaming Non-Recurrent On-Device End-to-End Model with Improvements to Rare-Word Modeling. In *Proc. Interspeech 2021*, pages 1777–1781.

Tara N. Sainath, Ruoming Pang, David Rybach, Yanzhang He, Rohit Prabhavalkar, Wei Li, Mirkó Visontai, Qiao Liang, Trevor Strohman, Yonghui Wu, Ian McGraw, and Chung-Cheng Chiu. 2019. Two-Pass End-to-End Speech Recognition. In *Interspeech 2019*, pages 2773–2777.

Yuan Shangguan, Kate Knister, Yanzhang He, Ian McGraw, and Françoise Beaufays. 2020a. Analyzing the Quality and Stability of a Streaming End-to-End On-Device Speech Recognizer. In *Interspeech 2020*, pages 591–595.

Yuan Shangguan, Jian Li, Qiao Liang, Raziel Alvarez, and Ian McGraw. 2020b. Optimizing Speech Recognition For The Edge. In *Third Conference on Machine Learning and Systems, On-Device Intelligence Workshop*.

Yuan Shangguan, Rohit Prabhavalkar, Hang Su, Jay Mahadeokar, Yangyang Shi, Jiatong Zhou, Chunyang Wu, Duc Le, Ozlem Kalinli, Christian Fuegen, and Michael L. Seltzer. 2021. Dissecting User-Perceived Latency of On-Device E2E Speech Recognition. In *Proc. Interspeech 2021*, pages 4553–4557.

Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. 2018. Self-Attention with Relative Position Representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 464–468, New Orleans, Louisiana. Association for Computational Linguistics.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention Is All You Need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 6000–6010, Red Hook, NY, USA. Curran Associates Inc.

Zengwei Yao, Liyong Guo, Xiaoyu Yang, Wei Kang, Fangjun Kuang, Yifan Yang, Zengrui Jin, Long Lin, and Daniel Povey. 2024. Zipformer: A faster and better encoder for automatic speech recognition. In *The Twelfth International Conference on Learning Representations*.

Zengwei Yao, Wei Kang, Fangjun Kuang, Liyong Guo, Xiaoyu Yang, Yifan Yang, Long Lin, and Daniel Povey. 2023. Delay-penalized CTC Implemented Based on Finite State Transducer. In *Proc. INTERSPEECH 2023*, pages 1329–1333.

Jiahui Yu, Wei Han, Anmol Gulati, Chung-Cheng Chiu, Bo Li, Tara N Sainath, Yonghui Wu, and Ruoming Pang. 2021. Dual-mode ASR: Unify and Improve Streaming ASR with Full-context Modeling. In *ICLR 2021*. ICLR 2021.

Xiaohui Zhang, Frank Zhang, Chunxi Liu, Kjell Schubert, Julian Chan, Pradyot Prakash, Jun Liu, Ching-Feng Yeh, Fuchun Peng, Yatharth Saraf, and Geoffrey Zweig. 2021. Benchmarking LF-MMI, CTC And RNN-T Criteria For Streaming ASR. In *2021 IEEE Spoken Language Technology Workshop (SLT)*, pages 46–51.