Partial Rewriting for Multi-Stage ASR

Antoine Bruguier, David Qiu, Yanzhang He
Google LLC, USA

{tonybruguier,qdavid,yanzhanghe}@google.com

Abstract

For many streaming automatic speech recognition tasks, it is important to provide timely intermediate streaming results, while refining a high quality final result. This can be done using a multi-stage architecture, where a small left-context only model creates streaming results and a larger left- and right-context model produces a final result at the end. While this significantly improves the quality of the final results without compromising the streaming emission latency of the system, streaming results do not benefit from the quality improvements. Here, we propose using a text manipulation algorithm that merges the streaming outputs of both models. We improve the quality of streaming results by around 10%, without altering the final results. Our approach introduces no additional latency and reduces flickering. It is also lightweight, does not require retraining the model, and it can be applied to a wide variety of multi-stage architectures.

1   Introduction

Streaming automatic speech recognition (ASR) processes the audio as it is being spoken, without waiting for the entire utterance to be finished before generating a transcript. The intermediate results are called *partial* results, while the single result produced at the end of the utterance is called a *final* result. This is in contrast to batch ASR which waits until the entire utterance is spoken before generating only a final result. Streaming ASR is useful in applications where it is important to provide feedback to the user as soon as possible, such as voice-activated assistants, dictation, or live video captioning.

The first streaming ASR systems were developed several decades ago Yu and Deng [2015], Hinton et al. [2012], Brown et al. [1982], Selfridge et al. [2011], Fink et al. [1998], Saraclar et al. [2002], but more recently new approaches relying on end-to-end models Li et al. [2019], Yeh et al. [2019], Sainath et al. [2020], He et al. [2019] have been developed. However, batch ASR usually outperforms streaming ASR because it can better understand the context of the speech and therefore make more accurate predictions.

Cascaded recognizers Narayanan et al. [2021], Sainath et al. [2021] attempt to have the best of both worlds by combining the strengths of both streaming and batch ASR systems. They do so by using two models. A *causal* model only uses the left context outputs and therefore has low latency. A *cascaded* model with both left and right contexts looks ahead in the audio to output results and therefore has better accuracy. The cascaded model is also more accurate because it has more weights than the causal one. While both models can generate partial and final results, typically only the causal model is used to generate the partial results so that the latency is low, and only the cascaded model is used to generate the final result so that it is more accurate Sainath et al. [2022]. This approach enables low-latency streaming ASR while achieving a high quality for the final results. However, this leaves a gap in quality because the partial results still have relatively lower quality.

Further, running two models independently can cause issues because even though the models decode the same audio and even share some of their weights, there is no coupling between the two outputs and the partial results may occasionally differ substantially from the final result. Recent improvements have been able to tie in several models together Mahadeokar et al. [2022], Li et al. [2023]. By running the two models together, they are able to improve the partial results, but this comes at the cost of not having a right context for high-quality final results. In essence, both models are causal, but one is still larger than the other. Further, the two models now have to be run synchronously, and the overall architecture is fixed at training time.

In parallel to these modelling efforts, there have been some improvements in how partial results can be evaluated along three dimensions: flickering, partial quality, and partial latency. Flickering occurs when words that have already been decoded in a previous partial results are changed in subsequent partial results. This negatively impacts the experience for the users Noble et al. [2022], Muller et al. [2016] as well as potentially increasing the computation cost of other systems (e.g. translation Iranzo-Sánchez et al. [2020] or assistants Chang et al. [2020]). The amount of flickering can be measured using the metric defined in Shangguan et al. [2020]. Work by Bruguier et al. [2022] defines metrics for partial quality and latency. This allows us to measure improvements to partial results along these three dimensions. We will describe the three metrics of interest in more detail in section 2.3.

2 Goals and constraints

2.1 Overview

In the present work, we want to improve the quality of the partial results by using a large model that has a right context, but we want to do so without scarifying latency. Contrary to Narayanan et al. [2021], Sainath et al. [2021] we no longer see partial results as a by-product of the decoding.

Instead, we want to keep two models running completely independently as

in Narayanan et al. [2021] and still allow the larger model to have a right context. We also do not require a fixed architecture at training time. Our work will leave the final results unchanged as they already benefit from the larger model and the use of a right context, but it will improve the quality of partial results significantly.

We want the approach to be cheap computationally. It should not increase the memory budget in any measurable way, nor increase power consumption.

2.2   Combining results from two models

The only assumption we make is that we have two models that output a stream of partial results. The results of the first model have low latency, but relatively lower quality. The results from the second model have much higher latency, but are of better quality. Like Narayanan et al. [2021], Sainath et al. [2021, 2022] we want the best of both world, but this time for partial results without tying in models architectures as done in Mahadeokar et al. [2022], Li et al. [2023] nor require retraining.

In order to validate our approach experimentally, we modify how the cascaded architecture creates partials. While we still only use the cascaded model to create final results, we now allow for *both* causal and cascaded model to create partial results. Then, our proposed approach combines these two streams of partial results into a single stream of new partial results that still have low latency, but better quality. We do not rely on the specificity of the cascaded architecture and *only* assume that we have two models running concurrently and decoding the same audio. We only modify the partial results, and leave the final results unchanged.

2.3   Metrics of interest

In order to measure the validity of our approach, we must be able to measure the quality of our newly created partial results. For this we heavily rely on a *partial word error rate* (PWER) metric defined in section 2.2 of Bruguier et al. [2022]. Roughly speaking, PWER is the WER averaged over all the partials, ignoring the missing words at the end of each partial.

The causal model outputs partial results with a very low latency, and we do not wish to introduce any new latency. We thus rely on a metric that estimates the latency of *all* partials, as defined in section 2.3 of Bruguier et al. [2022]. Roughly speaking, the *partial latency* (PL) metric is the average appearance time of every correct word since the beginning of the utterance, where only the change between experiments is the meaningful number.

Finally, an algorithm could increase the amount of flickering of the partial results. In order to control for this potential negative change, we measure the *unstable partial word ratio* (UPWR) as defined in Shangguan et al. [2020]. Roughly speaking, UPWR is the fraction of already decoded words in all the partial results that are changed by a subsequent partial or final result. However, in order to measure the effect of flickering more accurately, we expand the

definition of the metric by measuring UPWR on different set of results. If the decoding of an utterance produces N partial results and 1 final result, we measure UPWR on three different sets: 1) Only the N partial results, 2) Only the single N^{th} partial result and the final result, and 3) The N partial results followed by the final result. This allows us to measure how much flickering occurs during the streaming decoding, during the transition from streaming decoding to final result, and overall, respectively.

3 Proposed algorithm

3.1 Intuition and core algorithm

The algorithm we propose relies on text manipulation only. We take as input the two streams of partial results, one from the causal model and one from the cascaded model, and then create a new stream of composite partial results.

	_ro	za	ee	_how	_are	_you
	0 C	1	2	3	4	5	6
_ro	1	0	1	2	3	4	5
sa	2 I	1	1	2	3	4	5
l	3	2	2 S| 2	3	4	5
ie	4	3	3	3 C	3	4	5
_how	5	4	4	4	3 D→	4 D→	5

Figure 1: Example of Levenshtein alignment of two decoded sequences. The vertical is the partial result from the cascaded model $x[1, \ldots, m]$ with $m = 5$, while the horizontal is the partial result from the causal model $y[1, \ldots, n]$ with $n = 6$. The numbers correspond to the alignment cost $C(i, j)$ and the arrows the best path with C=correct, I=insertion, S=substitution, and D=deletion. The algorithm finds $j^* = 4$ that minimizes the cost of the last row, with $C(m, j^*) = 3$.

Consider the example where in the middle of decoding of an utterance, the user has said "Rosalie how are you". The causal model, being smaller and less accurate, has incorrectly decoded it as "_ro za ee _how _are _you" ($n = 6$ word pieces[1]), while the cascaded is more accurate but has a delay, and thus has only decoded "_ro sa l ie _how" ($m = 5$ word pieces) at the same frame. As expected $m < n$ since the cascaded model uses a right context, but it is not true that the $n-m = 1$ word piece "_you" corresponds to amount of audio in the right context. Indeed, if we were to naively copy the first m word pieces from the cascaded model and then append the $n - m = 1$ last word pieces from the

---

[1]We use word-pieces Sainath et al. [2020] to decompose text into tokens even but the approach can also use Unicode codepoints as tokens

4

causal model, we would get an incorrect transcript: "␣ro sa l ie ␣how ␣you" because the word "␣are" is missing.

Algorithm 1 Streaming algorithm that recomposes results.

```
latest_partial_used_for_rewriting ← ""
latest_cascaded_partial ← ""
while AudioStillAvailable() do
    partial_result ← PullPartialResult()
    if partial_result.origin == CASCADED then
        latest_cascaded_partial ← partial_result.text
    else
        OutputPartial(RewriteResult(partial_result.text))
    end if
end while
```

Algorithm 2 Algorithm that attempts rewriting with a fall-back.

```
procedure REWRITERESULT(causal_partial)
    cost, composite ← CreateComposite(causal_partial, latest_cascaded_partial)
    if cost < cost_threshold then
        latest_partial_used_for_rewriting ← latest_cascaded_partial
        return composite
    else
        _, composite ← CreateComposite(causal_partial, latest_partial_used_for_rewriting)
        return composite
    end if
end procedure
```

Instead, we need a way to blend the two partial results correctly. Every time we merge two partial results, we should correctly estimate how many extra tokens to copy from the causal results.

We propose to use a Levenshein alignment Cormen et al. [2001] of the two sequences, as shown in figure 1. By aligning the two sequences, we are able to compute the alignment costs. We then modify the Levenshtein alignment algorithm in a similar fashion as Bruguier et al. [2022] by allowing a variable reference length. Since we want to consume the *entire* cascaded sequence, we must have a final best alignment path on the last row (shown in a red box on figure 1). However, we do *not* want to consume all the causal sequence because it is likely to have more token due to the emission delay from the cascaded model. In other words, the deletions due to the time delay should be ignored. Thus, instead of using the bottom right cell as the end-path of our alignment, we sweep all the costs on the bottom row and use the cell with the lowest cost. In essence, the procedure allows us to discover how many causal tokens correspond to the time delay. We are now able to create a composite transcript: First, we copy *all* the tokens from the cascaded model, namely "␣ro sa l ie ␣how". Then, we append the *remaining* causal tokens, namely "␣are ␣you" to get a composite transcript "␣ro sa l ie ␣how ␣are ␣you".

Mathematically, if the cascaded transcript x has length m and the causal transcript y has length n, then the Levenshtein algorithm will compute the cost $C(i,j)$ of aligning substrings $x[1,\ldots,i]$ and $y[1,\ldots,j]$ for $i \in [1,m]$ and

$j \in [1, n]$. We then compute:

$$j^* = \underset{j \in [1,n]}{\arg\min} \left(C(m, j) \right) \tag{1}$$

and the composite transcript is (where $\oplus$ is a string concatenation):

$$z \triangleq x[1, \ldots, m] \oplus y[(j^* + 1), \ldots, n] \tag{2}$$

3.2 Cropping partial results for non-quadratic growth

Because we run the algorithm on streaming audio, we must make sure it doesn't delay the appearance of results. Further, on device applications require a low power consumption. One of the issue with the naive approach of section 3.1 is that its run time is quadratic. For short audio segments, this is not a problem, but when decoding longer duration segments, the cost can become prohibitive.

This can be alleviated by capping the length of the sequences so that the shorter sequence is at most of length M. Thus, we remove the first $P = \max(\min(m, n) - M, 1)$ tokens of each string and only align $x[P, \ldots, m]$ against $y[P, \ldots, n]$. Then, the composite transcript is:

$$z \triangleq x[1, \ldots, P] \oplus x[P + 1, \ldots, m] \oplus y[(P + j^* + 1), \ldots, n] \tag{3}$$

The intuition is that we only to have at most the last M tokens of both transcripts to align them properly. Using the approach, we can make the algorithm linear without much reduction in its quality.

3.3 Trimming cascaded input transcript

In order to reduce the amount of flickering, we can trim the cascaded transcript. Instead of simply using the full $x[1, \ldots, m]$ word pieces as an input, we can trim T word pieces and only use $x[1, \ldots, \max(m - T, 1)]$. While this does not create an additional latency (since we still use the full causal transcript), this may cause the composite transcript to have lower quality.

3.4 Alignment cost bailing and adding hysteresis

If the causal and cascaded transcripts differ too much, alignment may not produce desirable results. Misalignment reduce the quality of the composite result and when they get corrected cause flickering. We can use the alignment cost to determine the distance between the two transcripts, and if it is too high, we simply do not create a composite transcript and keep the causal transcript. We defined two costs measurements. The first uses the full sequence, namely:

$$\rho_f \triangleq \frac{C(m, n)}{m} \tag{4}$$

We can also limit ourselves to the end of the alignment and only consider the last K tokens:

Algorithm 3 Algorithm that creates a composite result.

```
procedure CREATECOMPOSITE(causal_partial,cascaded_partial)
    causal_tok ← Split(causal_partial)
    cascaded_tok ← Trim(Split(cascaded_partial))
    composite_tok ← {}
    P ← max(min(causal_tok.size, cascaded_tok.size) - M, 1)
    codes ← LevAlign(causal_tok[P..], cascaded_tok[P..])
    for j ← 1 to P do
      composite_tok.append(cascaded_tok[j])
    end for
    i ← P
    m ← 1
    cost ← 0.0
    for code ← codes do
       if code == SUBSTITUTE || code == CORRECT then
         composite_tok.append(cascaded_tok[j])
           if j > cascaded_tok.size - K then
             if code == SUBSTITUTE then
                 c ← c + 1.0
             end if
             m ← m + 1
           end if
           i, j ← i + 1, j + 1
       end if
       if code == INSERT then
         composite_tok.append(cascaded_tok[j])
           if j > cascaded_tok.size - K then
             c ← c + 1.0
             m ← m + 1
           end if
           j ← j + 1
       end if
       if code == DELETE then
           if j > cascaded_tok.size - K then
             c ← c + 1.0
             m ← m + 1
           end if
           i ← i + 1
       end if
    end for
    while i < causal_tok.size do
      composite_tok.append(causal_tok[i])
    end while
       return c / m, composite_tok
end procedure
```

$$\rho_r(K) \triangleq \frac{C(m,n) - C(\max(m-K,0),\max(n-K,0))}{\min(K,m)} \tag{5}$$

We only rewrite the transcript if the cost is below a settable threshold. However, if we had previously used a cascaded transcript to create a composite transcript, we fall back to the previously used one, so that we do not suddenly stop rewriting transcript.

3.5   Overall algorithm

We decompose the overall approach in three algorithms. In algorithm 1, we show the entry point of our approach. The code keeps pulling for new results as long as there is still audio to be decoded. In case a partial result comes

from the cascaded model, it is suppressed and we only store the text that would have been outputted. In case a partial result comes from the causal model, we rewrite it and show it to the user.

In algorithm 2, we show the hysteresis that allows to fall back on previously used cascaded partial results. We attempt to rewrite the causal partial using the latest cascaded partial. If the cost of rewriting such partial is low enough, we record the latest cascaded result and return the composite transcript. If the cost is too high, then we fall back to unconditionally rewriting the causal transcript using the latest cascaded result used. Thus, if the two models completely disagree, we fall back on the last time they agreed.

Finally, algorithm 3 shows how we create a composite partial result. It computes the alignment on a subset of the tokens and then outputs the composite tokens. The cost is computed during the reconstruction. For brevity, the algorithm for trimming of the tokens is omitted. In order to reduce the flickering of both the causal and cascaded models themselves, we applied the same deflickering algorithm as Bruguier et al. [2022] with $\alpha = 0.2$ for our test model.

4   Results

4.1   Base model

In order to evaluate the quality of our algorithm, we reused the model of Sainath et al. [2021].

It uses a streaming comformer-transducer where the encoder consists of 12 causal comformer layers Yu et al. [2021], each with 23 frames of left context, a self-attention with 8 heads, and a convolution kernel size of 15. We use an embedding prediction network with 2 previous labels as input Botros et al. [2021] and an embedding dimension of 320. The base model used fast emit Yu et al. [2021] in order to reduce the latency of the output.

We used a frame rate of 30ms, but the encoder stacks two frames and then downsamples, the effective frame rate is 60ms. Further, because the cascaded model looks ahead 15 stacked frames, it means that its output roughly corresponds to what was said about 900ms in the past. We reused the model Narayanan et al. [2019] which was trained on 400k hours of multidomain data with a one-hot domain ID Yu et al. [2021] indicating the type of utterance. All data are deidentified and the collection and handling abide by Google AI Principles aip.

4.2   Results

The results are shown on table 1. We do not report the regular WER as we verified experimentally that our algorithm indeed does not change the final results. We measure the UPWR, PWER, and partial latency on four test sets using the metrics described in section 2.3. The test model is the result of a sweep of the parameters that gives a good balance between increased quality

Test set		UPWR			PWER
		part.	trans.	all	
Dict-	Base	0.04	0.12	0.15	3.89
ation	Test	0.07	0.06	0.13	3.49
	Δ	75%	-50%	-13%	-10%
Voice	Base	0.05	0.14	0.19	5.10
Search	Test	0.08	0.09	0.16	4.99
	Δ	60%	-36%	-16%	-2%
TTS	Base	0.03	0.51	0.54	3.64
Audio	Test	0.16	0.02	0.18	2.94
book	Δ	433%	-96%	-67%	-19%
Libri-	Base	0.05	0.31	0.36	7.47
speech	Test	0.17	0.05	0.22	6.18
	Δ	240%	-84%	-39%	-17%
Tele-	Base	0.08	0.34	0.42	17.63
phony	Test	0.23	0.11	0.34	15.51
	Δ	197%	-68%	-19%	-12%

Table 1: Results of our proposed approach. As described in section 2.3, we measure UPWR for all three subsets (for only the partials, only the transition, and for all results) and PWER for our base and test models. We also show the change (Δ) either in percentage or milliseconds. The dictation test set are longer-form utterances corresponding to messages. The voice search test set are shorter queries seeking web results. We used text-to-speech (TTS) to synthesize long-form audio to produce long utterances. The librispeech test set is from Panayotov et al. [2015]. Finally, we also reported on a telephony set. All data was handled abiding by Google AI Principles aip.

and flickering ($alpha = 0.2$, $\rho_f = \infty$, $\rho_r = 0.5$, $K = 10$, $P = 25$ and $T = 1$). We can see that for all sets, the PWER is reduced significantly. The smaller reduction is for the voice search test set; this is explained by the average duration of the utterances. Since they are shorter in this test set, the 900ms delay of the cascaded model has greater impact. There is not as much time to rewrite the partial results. For the other test sets, the PWER is reduced by at least 10%.

The picture for flickering is more mixed. Because we want to improve the quality of partial results, we must, by definition, change the decoded words and because we do not want to increase the latency, we must therefore change already decoded words and thus cause flicker. We do see a large increase in the flickering of the partial results (left sub-column of the UPWR section) for all sets, with a larger increase for test sets for longer utterances. However, the UPWR for the transition from the last partial result to the final result (middle sub-column) is significantly lower. This is explained by the fact that our algorithm pulls in the transition from causal results to cascaded results earlier. Therefore, at the end of the utterance, the partial results are closer to the final results. Overall, we

see that the total UPWR (right sub-column) is lower.

As described in section 2.3, only the change in latency metric is meaningful. For all tests sets, the change was less than 10ms, well below human perception. We also ran measurements on a Pixel6a phone and the time spent in each rewriting is on average below $0.1ms$ meaning that our algorithm has very low computation requirements and power consumption.

The reader can observe the effect of the algorithm in videos sup that show the causal, cascaded, and merged partial results when decoding Librispeech Panayotov et al. [2015] utterances.

4.3 Hyperparameter sweep analysis

Figure 2: PWER and UPWER as a function of T

We plotted the PWER and the UPWR metrics according to the hyperparameter of the model for the dictation test set in figures 2, 3, and 4. Note that the parameters we chose for the results of table 1 are not the best according to the plots below. This is because we chose a set of values that had acceptable results for the many test sets. This is a product judgment call, but we found that the parameters we chose work well for almost all our test sets. This suggests that the hyperparameters were not overfitted to the dictation test set.

We see on figure 2 that as T increases (the number of words we remove from the causal partial), the PWER goes up. This is expected because if we have fewer causal words (which are usually more correct), we cannot improve the rewritten transcript as much. As T increases, we see that the UPWR for only the partials goes down (we rewrite less during decoding, thus we flicker less), and the UPWR for the transition goes up (by the time the final is shown, we have rewritten less and thus the transition has more flickering). Overall, the flickering is neutral.

Figure 3: PWER and UPWER as a function of ρ_r

Figure 4: PWER and UPWER as a function of K

We sweep the ρ_r parameter in figure 3 where a lower value means that we want a smaller edit distance. When the parameter is 0.0, we recover the value of the base model of table 1 (i.e. rewrite nothing). As it goes up, the PWER goes down because we rewrite more, but the flickering of the partial goes up (because we merge transcripts that are more different). As for the previous parameter, the transition UPWR goes down because we are closer to the final result.

We sweep the parameter K in figure 4, which means how many words we use for the edit distance hysteresis. If we set it to 0, we recover the case where $\rho_r = 1$. As K goes up, we are more strict when accepting rewrites, and thus the PWER goes up while the flickering goes down.

5   Conclusions

Until recently Bruguier et al. [2022], not much attention was paid to the quality of partial results of streaming ASR systems. It was assumed that improving final results would automatically improve partial results. Separately, more recent approaches for streaming decoding used multiple models Narayanan et al. [2021]. We presented an algorithm that doesn't require a re-architecturing of the model like in Mahadeokar et al. [2022] but instead relies on a simple, more general, rule to combine partials from multiple models. Doing so, we are able to significantly reduce partial word error rate and flickering without impacting partial latency. The computational cost of our approach is extremely low and therefore well-suited for on-device recognition. While we used a cascaded architecture to validate our approach, the proposed algorithm is applicable to any multi-decoder approach; For example, it can be applied to merging a stream of high quality but high latency partial results from a server recognizer into a stream of low-latency partial results from an on-device recognizer.

Our approach also opens a more flexible allocation of the model weights. We could re-allocate weights away from the causal model and towards the cascaded one. This could improve the quality of the final result, with limited impact on the partial results, precisely because we have rewriting.

Further work will focus on expanding the applicability of our approach. One area of further research is lowering the computational requirements of our systems by trading memory and computation away from the causal model towards the cascaded model without impacting the PWER too much. For example, the causal model could be run greedily with a single beam, or we could re-allocate the some weights to the cascaded model.

Another area of further research could be multi-stage and intermittent model. The approach described is easily expanded to 3 or more models, where the smallest model could be run more often, while the subsequent larger ones only as needed.

References

Dong Yu and Li Deng. *Automatic Speech Recognition*. Springer London, 2015.

Geoffrey Hinton, Li Deng, Dong Yu, George Dahl, Abdel-Rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara Sainath, and Brian Kingsbury. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 2012.

Peter Brown, James Spohrer, Peter Hochschild, and James Baker. Partial traceback and dynamic programming. *ICASSP*, 1982.

Ethan Selfridge, Iker Arizmendi, Peter Heeman, and Jason Williams. Stability and accuracy in incremental speech recognition. *SIGDIAL*, 2011.

Gernot Fink, Christoph Schillo, Franz Kummert, and Gerhard Sagerer. Incremental speech recognition for multimodal interfaces. *Interspeech*, 1998.

Murat Saraclar, Michael Riley, Enrico Bocchieri, and Vincent Goffin. Towards automatic closed captioning: low latency real time broadcast news transcription. *Interspeech*, 2002.

Jinyu Li, Rui Zhao, Hu Hu, and Yifan Gong. Improving RNN transducer modeling for end-to-end speech recognition. *ASRU*, 2019.

Ching-Feng Yeh, Jay Mahadeokar, Kaustubh Kalgaonkar, Yongqiang Wang, Duc Le, Kjell Schubert Mahaveer Jain, Christian Fuegen, and Michael Seltzer. Transformer-transducer: End-to-end speech recognition with self-attention. *CoRR*, 2019.

Tara 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, Minho Jin, Anjuli Kannan, Qiao Liang, Ian McGraw, Cal Peyser, Rohit Prabhavalkar, Golan Pundak, David Rybach, Yuan Shangguan, Yash Sheth, Trevor Strohman, Mirko Visontai, Yonghui Wu, Yu Zhang, and Ding Zhao. A streaming on-device end-to-end model surpassing server-side conventional model quality and latency. *ICASSP*, 2020.

Yanzhang He, Tara 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, Kanishka Rao Shuo-yiin Chang, and Alexander Gruenstein. Streaming end-to-end speech recognition for mobile device. *ICASSP*, 2019.

Arun Narayanan, Tara Sainath, Ruoming Pang, Jiahui Yu, Chung-Cheng Chiu, Rohit Prabhavalkar, Ehsan Variani, and Trevor Strohman. Cascaded encoders for unifying streaming and non-streaming ASR. *ICASPP*, 2021.

Tara N Sainath, Yanzhang He, and Arun Narayanan et al. An efficient streaming non-recurrent on-device end-to-end model with improvements to rare-word modeling. *Interspeech*, 2021.

Tara N. Sainath, Yanzhang He, Arun Narayanan, Rami Botros, Weiran Wang, David Qiu, Chung-Cheng Chiu, Rohit Prabhavalkar, Alexander Gruenstein, Anmol Gulati, Bo Li, David Rybach, Emmanuel Guzman, Ian McGraw, James Qin, Krzysztof Choromanski, Qiao Liang, Robert David, Ruoming Pang, Shuo yiin Chang, Trevor Strohman, W. Ronny Huang, Wei Han, Yonghui Wu, and Yu Zhang. Improving the latency and quality of cascaded encoders. *ICASSP*, 2022.

Jay Mahadeokar, Yangyang Shi, Ke Li, Duc Le, Jiedan Zhu, Vikas Chandra, Ozlem Kalinli, and Michael Seltzer. Streaming parallel transducer beam search with fast-slow cascaded encoders. *CoRR*, 2022.

Ke Li, Jay Mahadeokar, Jinxi Guo, Yangyang Shi, Gil Keren, Ozlem Kalinli, Michael L. Seltzer, and Duc Le. Improving fast-slow encoder based transducer with streaming deliberation. *ICASSP*, 2023.

Steve Noble, Jason White, Scott Hollier, Janina Sajka, and Joshue O'Conno. Synchronization accessibility user requirements. `https://www.w3.org/TR/saur/#caption-synchronization-thresholds`, 2022. [Online; accessed 23-July-2022].

Markus Muller, Sarah Funfer, Sebastian Stuker, and Alex Waibel. Evaluation of the kit lecture translation system. *LREC*, 2016.

Javier Iranzo-Sánchez, Adrià Giménez Pastor, Joan Albert Silvestre-Cerdà, Pau Baquero-Arnal, Jorge Civera Saiz, and Alfons Juan. Streaming cascade-based speech translation leveraged by a direct segmentation model. *EMNLP*, 2020.

Shuo-Yiin Chang, Bo Li, David Rybach, Yanzhang He, Wei Li, Tara Sainath, and Trevor Strohman. Low latency speech recognition using end-to-end prefetching. *Interspeech*, 2020.

Yuan Shangguan, Kate Knister, Yanzhang He, Ian McGraw, and Françoise Beaufays. Analyzing the quality and stability of a streaming end-to-end on-device speech recognizer. *Interspeech*, 2020.

Antoine Bruguier, David Qiu, Trevor Strohman, and Yanzhang He. Flickering reduction with partial hypothesis reranking for streaming ASR. In *SLT*, 2022.

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, 2nd edition, 2001. ISBN 0262032937.

Vassil Panayotov, Guoguo Chen, Daniel Povey, and Sanjeev Khudanpur. Librispeech: an ASR corpus based on public domain audio books. In *ICASSP*, pages 5206–5210. IEEE, 2015.

Google, Artificial Intelligence at Google: Our Principles. `https://ai.google/principles/`.

Jiahui Yu, Chung-Cheng Chiu, Bo Li, Shuo yin Chang, Tara N. Sainath, Yanzhang He, Arun Narayanan, Wei Han, Anmol Gulati, Yonghui Wu, and Ruoming Pang. Fastemit: Low-latency streaming ASR with sequence-level emission regularization. *ICASSP*, 2021.

Rami Botros, Tara N Sainath, Robert David, Emmanuel Guzman, Wei Li, and Yanzhang He. Tied & reduced RNN-T decoder. *arXiv preprint arXiv:2109.07513*, 2021.

Arun Narayanan, Rohit Prabhavalkar, Chung-Cheng Chiu, David Rybach, Tara Sainath, and Trevor Strohman. Recognizing long-form speech using streaming end-to-end models. *ASRU*, 2019.

Supplementary material. Contains three videos from utterances from Librispeech. A vertical bar shows the boundary between the text coming from the causal model and the cascaded one. This vertical border is added artificially in order to understand better the effect of the algorithm and is not present in the production code. We show an utterance where the PWER is improved greatly. In particular, we can see that in the base model, the correction only happens at the very end, when the final result comes from the cascaded model, rather than being corrected mid-decoding. We also show an example where the PWER is degraded. While the cascaded model has better quality *on average*, it sometimes performs worse than the causal model. Finally, we also show an example of high increase in flickering.