FLICKERING REDUCTION WITH PARTIAL HYPOTHESIS RERANKING FOR STREAMING ASR

Antoine Bruguier, David Qiu, Trevor Strohman, Yanzhang He

Google LLC, USA
{tonybruguier,qdavid,strohman,yanzhanghe}@google.com

ABSTRACT

Incremental speech recognizers start displaying results while the users are still speaking. These partial results are beneficial to users who like the responsiveness of the system. However, as new partial results come in, words that were previously displayed can change or disappear. The results appear unstable and this unwanted phenomenon is called *flickering*. Typical remediation approaches can increase latency and reduce the quality of the partials results, but little work has been done to measure these effects. We first introduce two new metrics that allow us to measure the quality and latency of the partials. We propose the new, lightweight approach of reranking the partial results in favor of a more stable prefix without changing the beam search. This allows us to reduce flickering without impacting the final result. We show that we can roughly halve the amount of flickering with negligible impact on the quality and latency of the partial results.

Index Terms— Flickering, partial quality metric, partial latency metric, beam search

1. INTRODUCTION

With incremental recognition, automatic speech recognition (ASR) systems display results before the user is done speaking. Periodically, the systems shows results that are the transcription of what has been said so far. These results are called *partial results* (or partials for short) as opposed to the *final result* which corresponds to the transcription of the entire utterance, once the user is done speaking.

Displaying partial results has been a feature of many automatic speech recognizers for several decades [1, 2, 3]. Users like seeing results, one of the reasons being that they are an indication that the system is responsive [4, 5]. It is a feature of keyboard dictation systems [6]. In some applications, such as live captioning of video [7], having streaming partial results is even a required feature.

Further, partial results can be used by downstream systems in order to reduce their own latency. For example, by using partial results, machine translation systems [8, 9] can start their processing even before the speech recognition is complete. Some ASR systems for translation are even designed with low-latency in mind [10].

However, having partial results introduces the complication of unstable results. As the users continue speaking, the recognizers do not necessarily append new words to the previous partial result. Rather, words that were previously displayed can be removed or changed. The recognizers can also insert new words in the previous partial. When the words of the partial results are unstable, they change rapidly on the screen: they flicker. As an example, we uploaded two videos as supplementary material [11].

Flickering creates a poor user experience [4, 12]. The rapid change in results is distracting [13, 14]. As users speak, their attention is drawn back to previous words, thus increasing their cognitive load and frustration with the system. Having non-flickering results is recommended for accessibility [15].

Further, flickering can negate latency gains [8, 9]. Because the previously decoded words are no longer present, the previous computation of downstream systems is no longer relevant. The downstream system needs to reprocess the new partial, thus increasing latency.

Thus, flickering is an important aspect of the quality of a streaming speech recognizer. Even though most speech recognition systems are evaluated only on the quality and latency of their *final* results, partial results also play an important and under investigated role in evaluating a recognizer. We should both improve the ability to evaluate partial results' quality, latency, and stability, and devise methods to improve these metrics.

Partial results are usually generated mid-decoding by using the hypotheses of the beam search [16]. As new audio frames come in, the beam search extends its top hypotheses. Periodically, recognizers can use the current top hypothesis of the beam to create a partial result. Then, at the end of the audio, they use the top hypothesis on the beam in its final state to generate a final result. This approach is common among both in conventional [17, 18] and end-to-end systems [19, 20, 21].

At its root, flickering is a by-product of repeatedly picking the top hypothesis before the beam search is complete. Contrary to generating a final result, which happens once, the decision of picking a partial hypothesis has to be made multiple times. But as the decoding progresses, there is no guarantee that the previously picked hypothesis is a prefix of the currently picked hypothesis. This means that the previous partial result is not necessarily a prefix of the current partial result.

This is illustrated in figure 1, where we show two consecutive beam search steps. On the first step, we have three hypotheses and the one with the lowest cost is `_just,_stand`. Then, more audio comes in and at the next decoding step, the hypothesis with the lowest cost is `_just,_send,_text`, resulting in flickering because the token `_stand` disappears and is replaced with `_send`.

While modifying the beam search algorithm is at first a reasonable approach, it has numerous side effects. Since the final results are generated from the top hypothesis at the end of the beam search decoding, if we modify the beam search algorithm, we would potentially modify the final result.

For example, we could replace the beam search with a greedy search (which is equivalent to having a beam size of 1). This would effectively completely suppress the flickering, but would severely degrade the quality of both the partial and final results. Another example would be to only generate partial results from the common path of the beam search among all the hypotheses. We would only display the common path among all the hypotheses on the beam.

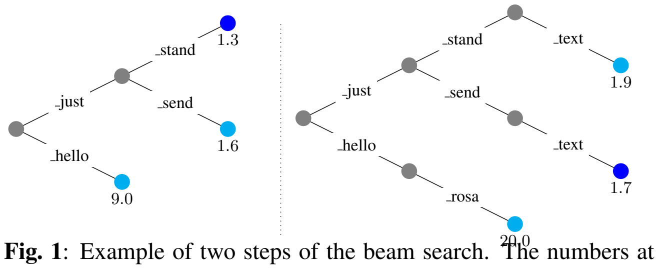


Fig. 1: Example of two steps of the beam search. The numbers at the leave nodes are the cost of the entire path from to root to the leaf.

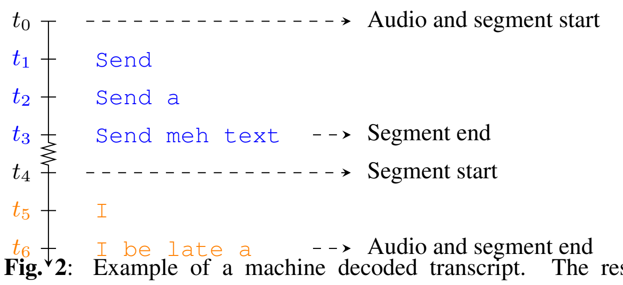


Fig. 2: Example of a machine decoded transcript. The result has two segments, $[t_0, t_3]$ and $[t_4, t_6]$, and has five partial results, “`Send`”, “`Send a`”, “`Send meh text`”, “`I`”, and “`I be late a`”. The partials are reset between the two segments, so that we stop appending new tokens and instead restart from an empty transcript.

This would also suppress flickering, but would introduce unacceptable delays in the generation of partials.

The two examples above, while extreme, do illustrate the constraints at play. We have to balance three objectives: we want partial of high quality, low flickering, and low latency. Further, when we address these objectives for the partial results, we would also prefer not degrading the final result’s quality and latency. Previously proposed flickering reduction algorithm [22, 23] also did consider latency and quality.

Thus, even before we attempt to improve the stability of partial results, we need to be able to measure the effect of our design choices on flickering, quality, and latency. While there is prior work on measuring the amount of flickering itself [22] to the best of our knowledge, there is no satisfactory metric for partial quality nor partial latency. Previous work [22, 23] did not measure the latency of all the partials, but rather the first partial. They also tried to improve the partials by increasing the period at which they were generated.

The paper is interested in both defining new metrics and once the metrics are defined, improving flickering without degrading quality nor latency.

The rest of our paper is organized as follows. In section 2, we describe the three metrics to evaluate partials. In section 3 we propose one algorithm that reduces flickering. This algorithm does not modify the beam search in any way, but rather modifies how to pick results from the output of an existing beam search algorithm. In section 4 we experimentally analyze our algorithms. Finally, we conclude in section 5.

2. METRICS

As described in section 1, we need three metrics: flickering, quality, and latency. For the flickering metric, we simply re-use the same method as [22] unmodified. However, to our knowledge, there is no satisfactory metric for either partial quality nor partial latency. Below in sections 2.2 and 2.3 we describe how these could be measured. All our metrics are performed on the decoded transcripts[1]. The quality and latency metrics of [22] were only considering the *last* partial, which we believe is not sufficient.

2.1. Flickering metrics

We briefly summarize the metrics of [22]. The gist of their approach is to count the numbers of words that flickered. They then tally the results and get the *unstable partial word ratio* (UPWR) and the *unstable partial segment ratio* (UPSR). Roughly speaking, UPWR measures the fraction of words that flicker and UPSR measures the fraction of segments that flicker. For both, the lower the value, the better. We refer the reader to section 2 of [22] for the exact description of the metrics.

2.2. Partial word error rate (PWER) metric

We need to have an aggregate metric that evaluates the quality of all the partials, not just the last emitted result. Further, this task is complicated by the fact that the decoding of the audio occurs in segments, as illustrated in figure 2. Our proposed evaluation method is in two main steps: segment attribution and segment scoring. At the end, we compute a partial word error rate (PWER) metric.

2.2.1. *Segment attribution*

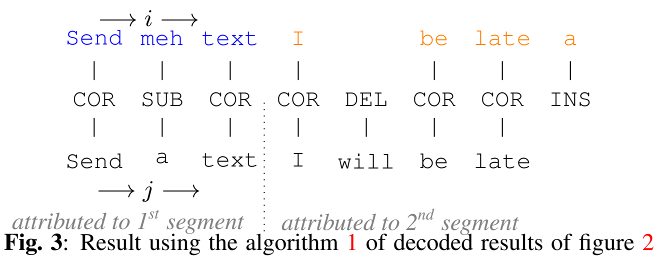
Algorithm 1 Algorithm that performs the segment attribution (we left out the creation of concatenation of the last partial of each segments, for brevity).

```
align_codes $\leftarrow$ Lev(ref, concat_last_partials)
$i \leftarrow 0$
$j \leftarrow 0$
for align_code in align_code do
    if align_code == SUB or COR then
        ref[$j$].segment $\leftarrow$ concat_last_partials[$i$].segment
        $i \leftarrow i + 1$
        $j \leftarrow j + 1$
    end if
    if align_code == INS then
        $i \leftarrow i + 1$
    end if
    if align_code == DEL then
        $i_{\text{capped}} \leftarrow \min(i, (\text{concat\_last\_partials}) - 1)$
        ref[$j$].segment $\leftarrow$ concat_last_partials[$i_{\text{capped}}$].segment
        $j \leftarrow j + 1$
    end if
end for
```

Speech recognizers often split utterances into chunks called segments, which are delimited by silence [24]. Each segment gets its own independent beam search. Our reference transcript, however, is not on a per-segment basis, but for the entire utterance. It would not be possible to have humans annotate on a per-segment basis because the segmentation is performed by a model [24] and therefore it could change from experiment to experiment.

Considering the example of figure 2, we only have the reference transcript `Send a text I will be late`. We don’t know a priori what part of the reference should be used to score each of the two segments. For example, one recognizer may decide that there is silence between the two sections and have the two segments: `Send`

[1]Even though our system [21] uses word pieces, the metrics are computed after the word pieces are reassembled into a string of characters forming words.



→ i →
Send meh text I be late a
COR SUB COR COR DEL COR COR INS
Send a text I will be late
→ j →
attributed to 1st segment
attributed to 2nd segment

Fig. 3: Result using the algorithm 1 of decoded results of figure 2 using a reference transcript `Send a text I will be late`. The algorithms determines that the first segment corresponds to the reference `Send a text` and the second segment to `I will be late`.

`meh text` and `I be late a`, but it is not immediately obvious that the first segment corresponds to the part of the reference that is `Send a text` and the second one to `I will be late`. Another recognizer might decide that there is only a single segment. Consequently, human annotation of segmentation would not be useful because segmentation itself is a result of modelling choices. Instead, the metric itself needs to decide which words of the reference transcript belong to which segment; it needs to do *segment attribution*.

The proposed algorithm for segment attribution has two steps. First, we use the last partials of each segment, concatenate them, and compute a Levenshtein [25] edit distance. Then, using the alignment results, we can attribute each entry of the reference to a segment. In short, we transfer the segment information from the partials onto the reference, thus assigning a segment information to each reference token.

The algorithm 1 inset presents the details and is illustrated by an example in figure 3 where the variable i indexes the partials on the top row, and the variable j indexes the reference on the bottom row. It essentially walks along the alignment backtrace [25], taking note of which segment the partial token is from, and assigning this segment to the reference token. The only special case is for deletion, where there is no corresponding partial token. When this occurs within a segment, we just keep use this segment. But when the deletion occurs between two segments, we can assign either the previous segment or the next one. We arbitrarily chose to assign the next one. However, for n segments, there are $n + 1$ gaps (including before the first segment and after the last segment). We need to special-case when the last token of the reference is deleted, in which case, we assign the *previous* segment.

We illustrated the output of the algorithm of inset 1 on the example of figure 2 in figure 3. The top row is the concatenation of the last decoded partials with the color indicating which segment it came from. The bottom row is the reference transcript. The middle row shows the results of the Levenshtein decisions are indicated by `COR`, `SUB`, `DEL`, and `INS` for correct, substitution, deletion, and insertion. Thus, the reference tokens `Send`, `a`, and `text` get attributed to the first segment and `I`, `will`, `be`, and `late` to the second one.

2.2.2. *Computing a partial errors for each segment*

Once segment attribution is performed, we have a reference transcript for a segment. Continuing with the example result of figure 2 and the reference attribution of figure 3, we have partials “`Send`”, “`Send a`”, and “`Send meh text`” which need to be scored against the reference “`Send a text`”. Intuitively, the first partial “`Send`” scored with reference “`Send a text`” should have no error, but if we were to compute the number of errors using the traditional metric, there would be two deletions.

One possible modification, is to compute the edit distance of the partial against all prefixes of the reference and return the minimum. Mathematically, if `Lev` is the traditional Levenshtein distance, we would replace:

$$\texttt{Lev(seg\_ref, partial)} \tag{1}$$

with

$$\min_{m} \left(\texttt{Lev(seg\_ref[}0:m\texttt{], partial)}\right) \tag{2}$$

The naive implementation would be computationally expensive. The Levenshtein algorithm is quadratic, and the additional looping would make it cubic. However, the algorithm can be modified to return the same result, but keeping it quadratic.

The traditional Levenshtein edit distance is computed using dynamic programming. If we have two strings a and b of lengths m and n, respectively, the algorithm computes an array C of dimensions $(m + 1, n + 1)$ where each cell $C(i, j)$ contains the value:

$$C(i, j) = \texttt{Lev (a[0..i], b[0..j])} \tag{3}$$

The edit distance is then read out from the bottom right cell in the array: $C(m, n)$ and the rest of the array is discarded.

In our case, we need not restrict ourselves to reading from the bottom right cell in the array. We still compute the Levenshtein distance of the partial against the *full* segment reference. Then, instead, of reading only from $C(m, n)$, we compute:

$$\min_{i\in[0,m]} C(i, n) \tag{4}$$

which will, by definition, readily give the same result as in equation (2). The length of the reference is:

$$\arg\min_{i\in[0,m]} C(i, n) \tag{5}$$

For the last partial of the segment, we do not compute the minimum, and instead revert to the traditional Levenshtein metric because we should match all the words in the reference. Thus, using the new approach, we no longer need a cubic algorithm and instead we keep the algorithm quadratic and simply add a linear step at the end.

2.2.3. *Aggregate across all segments and all utterances*

In section 2.2.1 we described a way to split the reference into segments and in section 2.2.2 we described a way to score individual partials. We need now to aggregate scores for each partials into a single score.

Several choices are possible, and the decisions can be subjective. A word that appears early in the segment might be wrong for the entire duration of the segment, or might be fixed half way through. A word that is at the end of the segment will spend less time displayed on the screen, and it might be less severe of a mistake than a early word being wrong. We would have to decide on how to weigh these possible preferences.

For simplicity, we compute the sum of the number of errors for each the partials without any weighting. Then, the length of reference is the sum of equation (5) for each partial. Thus, for each utterance, we have two values: a number of errors and a length. We then aggregate over utterances as for the traditional WER. A notable difference of the partial WER metric is that they are *not* comparable with the final WER, but the partial WER metric is still internally comparable across different ASR systems.

2.3. Partial Latency (PL) metric

The other metric to measure is the partial latency (PL) of the emitted partials. We had to make design decision about alignment of references.

First, our metric does not require force alignment. While using an external model that has good timing information would have allowed us to force-align and get timing information, this step is expensive and require an external model to align [26, 27].

Second, our metric does not use the timing of the final result to self-align. This allows us to compare an existing data set across models, as the values are consistent.

Instead, we measure all latencies from the start of the audio. Thus, we cannot compare across data sets, but we can compare across models and de-flickering algorithms within the same test set.

2.3.1. Reference token emission time determination

The proposed latency metric relies extensively on previous algorithms from section 2.2. We first reuse the segment attributions computed in section 2.2.1 without any change. We thus have a reference transcript for each segment.

For each segment, we want to compute when a reference word first appears in a partial. Intuitively, the users would consider that a word they spoke has been decoded when it first appears. If it doesn't appear (because recognition is imperfect), then the users would consider the word has been mis-transcribed when the subsequent word has been decoded. We use an algorithm similar to the one in inset 1 as shown in inset 2. It loops over all the partials and does the same modified Levenshtein distance as in 2.2.2. Then, using the alignment codes, it advances very similarly as the previous algorithms, and when it finds a correct token, it records the partial from which it came from.

Algorithm 2 Algorithms that estimates the first appearance of a word in a reference token. For brevity, we assume that the `ref[`j`].time` are initialized to $+\infty$ for all j. Note that the algorithm is very close to the trace back step of the edit distance algorithm.

```
for partial in segment_partials do
    align_codes ← VarLenLev(ref, partial)
    $i \leftarrow 0$
    $j \leftarrow 0$
    for align_code in align_codes do
        if align_code == SUB then
            $i \leftarrow i + 1$
            $j \leftarrow j + 1$
        end if
        if align_code == COR then
            ref[$j$].time ← min(partial.time, ref[$j$].time)
            $i \leftarrow i + 1$
            $j \leftarrow j + 1$
        end if
        if align_code == INS then
            $i \leftarrow i + 1$
        end if
        if align_code == DEL then
            $j \leftarrow j + 1$
        end if
    end for
end for
```

While the two algorithms are similar, they have some notable differences. One is that in algorithm 1 we use the traditional Levenshtein distance once per utterance, whereas in algorithm 2 we use the *modified* Levenshtein distance once per *partial*. Another is that in the second algorithm, we treat substitutions and correct tokens differently. We only assign a time when the token is correct.

Our algorithm is not guaranteed to assign a time to each reference token. If none of the partials contain the reference token, then the algorithm will not assign a time to this token. We fix this issue with a follow-up pass on the reference tokens. For each of the reference tokens with the time missing, we assign the time of the next reference token. We do so recursively until we find a reference token with a time or, ultimately, the time of the end of the segment. This can be performed in linear time by a traversing the tokens from the last to first, thus allowing us to compute $\hat{T}_i = T_{\texttt{first\_appearance}}(\texttt{ref\_word}_i)$ for all words in the *reference* transcript.

2.3.2. Aggregation across all segments and all utterances

From the previous section, we obtained time stamps for each reference token. We need to aggregate them into a single metric. We simply first average within an utterance, and then average over all the utterances.

As mentioned at the beginning of section 2.3, the metric returned does *not* use reference times, either from a force-alignment, or from the final results. The metric is the reference emission time delayed from the start of the utterance, and thus test sets with longer utterances on average will have larger numbers. We had made this decision for ease of use, and ability to compare algorithms *within* a data set. Thus, this partial latency metric cannot be used for a single system alone, but can only be used in relative change between two systems as the user perceived emission delay change. Our metric is:

$$\texttt{PL} = \frac{1}{N}\sum_{i}^{N}\hat{T}_i \tag{6}$$

And when comparing models:

$$\Delta = PL(\texttt{test}) - PL(\texttt{base}) \tag{7}$$

Note that if we had force-alignment information, we would have the true time for each reference word, T_i, and thus be able to subtract it from the algorithm's time, but the value of Δ would be unchanged:

$$\frac{1}{N}\left(\sum_{i}^{N}\hat{T}_i(\texttt{test}) - T_i\right) - \frac{1}{N}\left(\sum_{i}^{N}\hat{T}_i(\texttt{base}) - T_i\right) = \Delta \tag{8}$$

3. FLICKERING REDUCTION ALGORITHM

When designing a new algorithm, we had several requirements. Foremost, we did *not* want to change the final results. While changing the beam search can sometimes improve latency [28], it creates complications because it ties partial flickering and the final results. We also assumed that the existing system had been very well tuned and that it would be difficult to improve its beam search. We wanted that no matter how the partial results were modified, the final results and their WER should be unchanged. We would also prefer that any flickering improvement we propose be orthogonal to any beam search improvements, such as [29]. Thus, instead of modifying the beam search algorithm, we left it completely unchanged. We did not modify how the hypotheses on the beam are generated as the audio

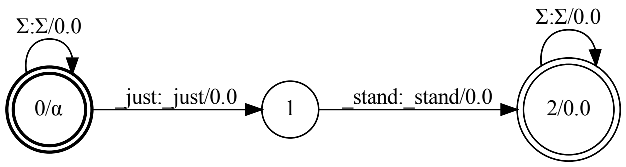


Fig. 4: Rescoring lattice to favor the prefix "`_just,_stand`". There are two sigma loops (that match any token) on states 0 and 2. Since the state 0 had exit weight α it penalizes hypotheses that do not match the prefix.

comes in. Instead, we simply changed how we selected a hypothesis from the beam to generate a partial from. At each partial generation step, we no longer always pick the current best hypothesis on the beam, instead, we might pick a different hypothesis from the beam, one that balances flickering, quality, and latency. Once we have generated a partial, the beam search proceeds unmodified. Second, we did not want to incur much of a computational cost. For example, adding an additional neural network to decide which hypothesis to use to generate a partial could be computationally expensive.

Every time we want to generate a partial, the proposed algorithm reranks the hypotheses on the beam based on whether they share a prefix with the partial that had been generated previously[2]. If a given hypothesis does share a prefix, its cost is left unchanged. If a given hypothesis does not share a prefix, its cost is increased by a penalty. We then pick the hypothesis with the lowest new cost to create the new partial. Mathematically, at every partial generation step, the previous algorithm was picking:

$$\arg\min_i \left(\text{cost}\left(\text{hyp}[i]\right)\right) \tag{9}$$

Instead, we first defined a new penalty function:

$$P(i) \triangleq \begin{cases} 0.0 & \text{if previous\_partial is a prefix of hyp}[i] \\ 1.0 & \text{otherwise} \end{cases} \tag{10}$$

And then, using this new penalty function, every time we generate a partial, the proposed algorithm instead picks:

$$\arg\min_i \left(\text{cost}\left(\text{hyp}[i]\right) + \alpha \cdot P(i)\right) \tag{11}$$

Where α is an hyperparameter of the model. If $\alpha = 0$, then we revert to the previous algorithm. If $\alpha \to \infty$ then we have maximum de-flickering. Note that even with a very large value, flickering may still occur. If at a certain decoding step, the previous partial is a prefix of *none* of the hypotheses on the beam, then all the possible hypotheses will flicker and thus we will have $P(i) = 1.0$ for all hypotheses, thus leaving the ranking unchanged. We chose this approach to eliminate the possibility of having the model be stuck.

Note that the penalty imposed in equation (11) is only for the generation of partials. The beam expansion uses the original costs and it *never* uses the penalty function from equation (10). Because the beam search algorithm is left unchanged, the final result is not changed either. This allows us to reduce flickering without impacting the final results.

Intuitively, we think the algorithm makes sense because it allows us to control the flickering and quality. If two hypotheses are very close in cost, then we aren't confident that the top one will have the best partial in term of quality, so we might as well as pick the hypothesis that does not flicker. If two hypotheses are far in cost, then it could make sense to accept some flickering because we are likely to have a better quality. Another way to think about the proposed algorithm is that we introduced some hysteresis in the update of the prefix in the partials. How partials are generated at a decoding step is no longer only a function of state of the beam. Instead, it is also a function of the previously generated partial, with a preference for not changing already decoded words.

In terms of latency, we expected our algorithm to have a small effect. The amount of computation required is quite small. We only perform prefix comparisons on the order of 10 hypotheses and only when we decide to output a partial. It could, however, increased the perceived latency nonetheless. By degrading the quality of the partials, our latency metric (section 2.3) might increase because words that are not found will have their timestamp attributed to the following word.

Other possibilities for the cost function of equation (10) are possible. Instead of a binary penalty, we could have used a distance function between the previous partial and the current hypothesis. The proposed function is however extremely cheap to compute and as we will see in section 4, it already performs well.

Going back to the example of figure 1, we see in the second step that there are three competing hypotheses:

- "`_just,_send,_text`" is the best hypothesis (cost 1.7), but it creates some flickering because it doesn't match the prefix `_just,_stand`.
- "`_just,_stand,_text`" does not flicker, but it is not the best hypothesis because its cost is 1.9.
- "`_hello,_rosa`" is neither the best hypothesis and it does flicker, so it should not be considered.

Given that the gap between the first and second hypotheses is $1.9 - 1.7 = 0.2$, if $\alpha < 0.2$, then the first hypothesis will be used to generate a partial and we will show `_just,_send,_text`, the top hypothesis, but one that flickers. Conversely, if $\alpha > 0.2$, then we will show `_just,_stand,_text`. Then, as new audio comes in, the beam search continues using the original costs, 1.7, 1.9, and 20.0 regardless of the value of α.

In practice, we rescore through FST operations on the lattice. First, given the previous partial, we build a linear FST that has a special path for the prefix. An example for the prefix "`_just,_stand`" is shown on figure 4. There are two ways to exit the FST: Either we stay in the start state and incur an exit cost, or, if allowed, we traverse the prefix section of the FST and exit without an extra cost. Then we compose the FST with the original partial hypothesis lattice, choose the path with the lowest cost as the partial result.

4. EXPERIMENTS

To evaluate the quality of our algorithm, we based our model on [30] with a streaming Conformer-Transducer. The encoder consists of 12 causal Conformer [31] layers, each with 23 frames of left context, 8-head self-attention and convolution with kernel size 15. We use an embedding prediction network [32] with 2 previous labels as input and an embedding dimension of 320. The joint network uses a single feed-forward network with 640 units. The model was trained with FastEmit to reduce the partial latency [26]. The encoder output frame rate is 60ms. We use a fixed score difference from the top hypothesis as the "beam width", which often results in up to 10 hypotheses in the beam. For simplicity, we did not use the cascaded

[2]The algorithm operates on word-pieces, contrary to the metric that uses the decoded transcript as a string of characters.

	UPWR	UPSR	PWER	PL	Δ
base	0.08	0.23	4.83	3,084	
PEI = 100ms	0.05	0.14	4.81	3,106	+22
PEI = 200ms	0.03	0.08	4.78	3,149	+65
PEI = 300ms	0.02	0.06	4.84	3,171	+87
stab. thresh. = 0.1	0.08	0.23	4.86	3,084	0
stab. thresh. = 0.2	0.01	0.04	5.10	3,246	+162
stab. thresh. = 0.3	0.01	0.03	5.19	3,271	+187
α = 0.05	0.06	0.17	4.84	3,085	+1
α = 0.1	0.05	0.04	4.83	3,086	+2
α = 0.2	0.03	0.09	4.85	3,085	+1
α = 0.5	0.02	0.04	4.89	3,086	+2
α = 1.0	0.01	0.02	4.98	3,086	+2

Table 1: Sweep over our Voice Search over the parameter α. For all experiments, the final WER was 6.2. It shows that it is verified experimentally that our algorithm does not alter final results. We then show the previous partial flickering metrics, UPWR and UPSR, and the newly introduces partial WER (PWER) and partial latency (PL in millisecond) metrics. The Δ column shows the increase in the PL metric compared to the baseline. The first section is the base results, the second and third sections are the algorithms of [22], and the last section is our proposed algorithm. For all sections, we show both the previous metric (UPWR and UPSR) and the additional metrics we introduced (PWER and PL).

encoder nor the language model. The model was trained on 400k hours of multidomain data [33], using a one-hot domain ID [21].

In table 1, we compare the results of different systems. The baseline is the vanilla ASR system without any de-flickering approach, which shows a partial at every 60ms as the model output frame rate. As a comparison, we also re-implemented the two proposed de-flickering algorithms of [22]: The first approach is based on increasing the partial emission interval (PEI), which is the time we set between showing consecutive partial results from the ASR recognizer. Naturally the larger we set the interval, the more stable the partials are. The second approach uses a logistic regression approach from [23] to estimate the stability of partial results. Partial words that have a stability score lower than a predefined threshold are withheld from showing on the screen. Both approaches rely on delaying the partials. As a result, we can see in table 1 that both approaches can decrease flickering but at the cost of significantly higher latency, compared to the baseline (see the Δ column). We see that the increase in the PL metric for the proposed algorithm is negligible (about 2ms), while prior algorithms could increase latency to noticeable values of up to 187ms.

On the other hand, our approach (with a sweep of α in table 1) is much less punitive when it comes to latency, and still can achieve similar or more flickering reduction, compared to the two approaches above. This is a confirmation of the design approach that we had chosen of not delaying the partials. As we increase α, we see minimal impact on latency and steady reduction on flickering, while the partial WER goes up slightly as expected. In general, the partial quality regression caused by our approach is on par or lower than the two approaches above.

Overall, it seemed that setting $\alpha = 0.2$ provided a good balance between quality, latency, and flickering. While α might have been increased for the set in question, we wanted a robust choice under a wide set of conditions, and thus decided against a more aggressive setting. We thus fixed $\alpha = 0.2$ and computed our new metrics both

Test set		WER	UPWR	UPSR	PWER	PL	Δ
Voice	base	6.2	0.08	0.23	4.83	3,084	
search	α=0.2	6.2	0.03	0.09	4.85	3,085	+1
Smart	base	9.6	0.08	0.20	4.82	2,581	
speaker	α=0.2	9.6	0.04	0.09	4.88	2,581	0
Phone	base	5.8	0.07	0.24	4.08	4,532	
assistant	α=0.2	5.8	0.03	0.10	4.10	4,531	-1
Dict-	base	5.7	0.29	0.60	4.56	2,707	
ation	α=0.2	5.7	0.24	0.43	4.56	2,707	0

Table 2: Effect of the de-flickering algorithm on partial flickering, quality, and latency. The "WER" column is the *final*'s WER. It verifies experimentally that our algorithm does not alter final results.

with and without the proposed algorithm on several test sets. The results are shown on table 2. The difference in PL value across test sets for a fixed model is simply a reflection of a different average utterance length. We also see that the PL metric does not materially change due to the proposed de-flickering algorithm. We are able to roughly divide by two the amount of flickering with negligible impact on both quality and latency. These results appear to be robust across test sets. Thus, the choice of $\alpha = 0.2$ seems to work well under a wide range of conditions. Note that this choice is dependent on the underlying model that generates hypothesis costs.

Additionally, we demonstrate the visual effect of the proposed algorithm compared with the baseline and the increased partial emission interval approach on a Librispeech [34] utterance in our supplementary video [11], where the proposed approach reduces flickering without hurting the quality and latency of the partials as expected, even though our model wasn't trained on Librispeech.

5. CONCLUSIONS

Flickering is a result of the beam search and has some negative effects, both on the users and for subsequent systems that process the output of ASR systems. Flickering can be reduced, but it typically involves degrading the quality and latency of the partials.

In this paper, we first presented two new metrics that allow us to measure the quality and latency of the partials. To our knowledge, this is the first of such metrics for partials. Then, thanks to these new metrics, we were able to devise algorithms to improve them. By noticing the root cause of the flickering, we devised a better way to pick a hypothesis from the beam to generate a partial. By introducing some hysteresis, we were able to divide the the amount of flickering by roughly half with very little impact on quality and latency. Our algorithm is very lightweight, and has no noticeable impact on the amount of computation required. It does not require training of a new model, and is applicable to both conventional and end-to-end recognizers. The rest of the behavior of the model is left unchanged. In particular, because the beam search still behaves the same way, the final result is unchanged.

Future work plans to build on the new metrics. Thanks to our significantly improved measurement of partial results, we can safely explore more complicated architectures. In particular, we plan to expand our work to models with two-pass parallel streaming beam search. While some models use non-causal cascaded encoders with right context for only the final result [35, 30], others [36] modify the beam search to occasionally use a larger causal model. Measured by our new metrics, we can explore an approach to merge partials from a non-causal decoder to maximize the partial quality while minimizing flickering and partial latency.

6. REFERENCES

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

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

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

[4] Gregory Aist, James Allen, Ellen Campana, Carlos Gomez Gallo, Scott Stoness, Mary Swift, and Michael Tanenhaus, "Incremental dialogue system faster than and preferred to its non-incremental counterpart," *DECALOG*, 2007.

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

[6] 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.

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

[8] Naveen Arivazhagan, Colin Cherry, I Te, Wolfgang Macherey, Pallavi Baljekar, and George Foster, "Re-translation strategies for long form, simultaneous, spoken language translation," *ICASSP*, 2020.

[9] 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.

[10] Orion Weller, Matthias Sperber, Christian Gollan, and Joris Kluivers, "Streaming models for joint speech recognition and translation," *EACL*, 2021.

[11] "Supplementary material," Contains two videos from the same utterance from Librispeech. The file `libri_normal_speed_with_audio.mp4` is the playback at regular speed with the audio. Timing information is provided as second and frame number (with 30 frame per seconds). The top row is at the base, the middle row is our proposed algorithm with $\alpha = 0.2$, and the bottom row the comparison with partial delay with PEI = 300ms. The file `libri_slow_mo_silent.mp4` is the exact same information slowed down 10 times and without the audio.

[12] Timo Baumann, Michaela Atterer, and David Schlangen, "Assessing and improving the performance of speech recognition for incremental systems," *NAACL*, 2009.

[13] Moritz Stolte and Ulrich Ansorge, "Automatic capture of attention by flicker," *Attention, Perception, and Psychophysics*, 2021.

[14] Steven A. Hillyard, Hermann Hinrichs, Claus Tempelmann, Stephen T. Morgan, Jonathan C. Hansen, Henning Scheich, and Hans-Jochen Heinze, "Combining steady-state visual evoked potentials and fMRI to localize brain activity during selective attention," *Human Brain Mapping*, 1997.

[15] 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].

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

[17] Geoffrey Zweig, Chengzhu Yu, Jasha Droppo, and Andreas Stolcke, "Advances in all-neural speech recognition," *ICASSP*, 2017.

[18] 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.

[19] Jinyu Li, Rui Zhao, Hu Hu, and Yifan Gong, "Improving rnn transducer modeling for end-to-end speech recognition," *ASRU*, 2019.

[20] 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.

[21] 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.

[22] 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.

[23] Ian McGraw and Alex Gruenstein, "Estimating word-stability during incremental speech recognition," *Thirteenth Annual Conference of the International Speech Communication Association,*, 2012.

[24] Matt Shannon, Gabor Simko, Shuo-yiin Chang, and Carolina Parada, "Improved end-of-query detection for streaming speech recognition," *Interspeech*, 2017.

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

[26] Jiahui Yu, Chung-Cheng Chiu, Bo Li, Shuo yiin 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," *CASSP*, 2021.

[27] Bo Li, Anmol Gulati, Jiahui Yu, Tara Sainath, Chung-Cheng Chiu, Arun Narayanan, Shuo-Yiin Chang, Ruoming Pang,

Yanzhang He, James Qin, Wei Han, Qiao Liang, Yu Zhang, Trevor Strohman, and Yonghui Wu, "A better and faster end-to-end model for streaming ASR," *ICASSP*, 2021.

[28] Mahaveer Jain, Kjell Schubert, Jay Mahadeokar, Ching-Feng Yeh, Kaustubh Kalgaonkar, Anuroop Sriram, Christian Fuegen, and Michael Seltzer, "RNN-T for latency controlled ASR with improved beam search," *CoRR*, 2019.

[29] Jiwei Li, Will Monroe, and Dan Jurafsky, "A simple, fast diverse decoding algorithm for neural generation," *Arxiv*, 2016.

[30] 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.

[31] Anmol Gulati, James Qin, Chung-Cheng Chiu, Niki Parmar, Yu Zhang, Jiahui Yu, Wei Han, Shibo Wang, Zhengdong Zhang, Yonghui Wu, et al., "Conformer: Convolution-augmented transformer for speech recognition," *arXiv preprint arXiv:2005.08100*, 2020.

[32] 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.

[33] 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.

[34] Vassil Panayotov, Guoguo Chen, Daniel Povey, and Sanjeev Khudanpur, "Librispeech: an asr corpus based on public domain audio books," in *2015 IEEE international conference on acoustics, speech and signal processing (ICASSP)*. IEEE, 2015, pp. 5206–5210.

[35] 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.

[36] 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.