EFFICIENT CASCADED STREAMING ASR SYSTEM VIA FRAME RATE REDUCTION

Xingyu Cai, David Qiu, Shaojin Ding, Dongseong Hwang, Weiran Wang
Antoine Bruguier, Rohit Prabhavalkar, Tara Sainath, Yanzhang He

Google LLC, USA

ABSTRACT

In this paper, we explore various frame rate reduction schemes on the two-pass cascaded encoder model to improve its efficiency without scarifying the transcription quality. We conduct extensive studies on frame rate reduction strategies, left and right context window length, trade-offs in quality, latency, computation and power consumption, and performance in short- and long-form datasets. With the proposed schemes, we can lower the 2nd pass frame rate to 120 ms, half of the 1st pass's. This achieves 20% RTF reduction / 13% power saving / 19% lower final latency, without impact on the word-error-rate nor partial results' latency. If allowing partial latency increase, we can further reduce the frame rate to 180 ms or even 240 ms from the 1st pass, and obtain 45% RTF / 35% power savings, with a similar or even better (on the short-form testset) recognition accuracy.

Index Terms— On-device ASR, cascaded streaming model, frame rate reduction

1. INTRODUCTION

End-to-end (E2E) automatic speech recognition (ASR) has become popular in recent years [1, 2, 3]. There are many advantages of an E2E system. First, it does not require human annotations for the speech-text alignments, and instead learns a direct map from the speech signal to the output tokens [4]. This benefit enables the model to use word-piece [5] or other large vocabularies rather than phonemes or graphemes, and further improves the model capacity. More importantly, it often outperforms conventional systems, e.g. [6], in transcription quality [7, 8]. Additionally, it can run at a lower frame rate [9], which saves both training and inference costs. Meanwhile, without separate acoustic (AM), pronunciation (PM) and language models (LM), E2E systems are often more compact and could have smaller model sizes [10]. Interested readers can refer to these recent survey papers, [1, 11], for technical details of E2E ASR models.

Among all the benefits above, the inference efficiency and the small model size make the E2E model very suitable for on-device deployment. On-device ASR applications often require streaming recognition (i.e. allow the speech signal to be processed as it is being spoken). There are a number of challenges involved in on-device streaming ASR: 1) Due to the streaming nature, the model is typically causal, or has little right-context. Therefore, its recognition quality is often worse than full-context systems. 2) The on-device setting strictly limits the model size (typically at a maximum of a few hundreds megabytes) and the power consumption, such that heavy computational operations must be avoided as much as possible. To address these challenges, new paradigms and methods have been proposed. For example, two-pass architecture [12] brings right-context to the 2nd pass, which leads to better quality, but keeps the 1st pass causal only such that the streaming latency is not affected. To reduce the model size and satisfy on-device memory and storage requirements, quantization and sparsity training [10] are adopted. To achieve power savings, demanding model components are replaced by light-weight modules, e.g. in [13]. Lowering the frame rate [9] or skipping unnecessary frames [14], can be even more effective, as it saves the model's computation as a whole, rather than targeting a specific sub-module. Conventional hybrid systems often run at 10 ms frame length, while recent E2E streaming systems can achieve up to 80 ms frame length [15] without observing significant quality loss.

In this paper, we push the frame length to 240 ms. To the best of our knowledge, this is the first attempt to run on-device cascaded streaming ASR model on such a low rate. Our contributions are: 1. We can achieve 120 ms frame length without transcription quality loss or latency increase. 2. If allowing a slightly higher partial latency, 180 ms / 240 ms frame length can be used without / with $<$ 10% quality regression on long-form testsets, respectively. The system can save 35% power consumption. 3. When reducing frame rate, we propose to also reduce left / right-context frame number to maintain a similar context window duration. This is shown to boost WERs, reduce latency and save computation at the same time. 4. A comprehensive study on different reduction settings is conducted to evaluate quality, power and latency trade-offs. 5. We add an analysis on Voice Activity Detector (VAD) based multi-segment recognition.

2. RELATED WORK

There has been tremendous efforts to build **on-device streaming ASR** systems in the past decade [16, 17, 7, 18, 10]. It is highly desired for low-cost, real-time, privacy and reliability purposes. In [17], the authors list several key techniques to

979-8-3503-0689-7/23/$31.00

improve streaming ASR performance, including large batch-sizes, using word-piece targets, using shallow-fusion [19] for context biasing, parameter quantization, etc. To improve model robustness, [7] proposes to increase training data diversity. To reduce latency, [20] describes a technique called FastEmit to encourage the model emitting earlier than later. **Two-pass model** paradigm is proposed in [12], which applies future context in the 2nd pass, to bridge the performance gap between streaming and full-context ASR models. A number of works, e.g. in [18, 10], are continuously pushing the limit of this paradigm, by constructing more effective encoders and decoders, for the two passes.

Conformer [21] is one of the state-of-the-art architectures designed for speech tasks. It augments the standard Transformer architecture with convolutional layers, such that local dependencies are better captured. Conformer is supported in popular open-source ASR toolkits such as ESPNet [22] and next-gen Kaldi [23]. To achieve E2E ASR training without frame-level annotations, **CTC** [24] and **RNN-T** [4] models are widely used, where the latter generally outperforms the former regarding transcription quality. Both models maximize an aggregated probability of the target label sequences given the input speech frames, but RNN-T explicitly models the conditional dependence on previous labels. In [13], the conventional LSTM prediction network in RNN-T decoder, is replaced with a simple embedding layer, such that a huge speedup is gained without quality degradation. **LAS** [25] is another popular model for E2E training, but it often requires full context thus not suitable for streaming ASR.

Frame rate reduction gains popularity as ASR model capacity dramatically increases, such that a lower resolution emphasizing on meaningful acoustic units could help recognition quality. The authors in [26] argue that we can reduce frame rate to a level of duration of a phoneme, or a word-piece token. An early attempt [9] reduces frame rate to 40 ms length, on the CTC models. A recent work in [15], tests 40 and 80 ms frame length combined with different left and right-context lengths. To achieve reduction, different sub-sampling techniques are proposed in the past a few years. [26] proposes to use convolutional layers for sub-sampling. In [17], the authors propose a time-reduction layer that concatenate consecutive frames, which is shown to have no word-error-rate (WER) loss, up to 60 ms output frame length, on the RNN-T model. **Funnel-attention** is introduced in [27], as an alternative sub-sampling technique to preserve as much information as possible. The key idea is to exploit the query-key-value scheme inside an attention layer. Let the query have a stride k; each query is still attending to all the keys, but the output will be the same length as the queries, which is down-sampled by a factor of k. Unlike a uniform sub-sampling that simply discards unused frames, the Funnel-attention can aggregate information from the unused frames through the attention mechanism. Compared to other pooling methods, the Funnel-attention extracts more information from unused frames than a simple average or max operation. Its performance gain is demonstrated in [27] for NLP tasks and in [10] for ASR tasks.

3. THE MODEL

In Figure 1, we present the diagram of the cascaded streaming ASR model. It is a two-pass architecture. The inputs are log-Mel-spectrogram features, with a frame length of ℓ milliseconds. The 1st encoder is causal only. It consumes a stack of k_1 consecutive input frames each time, and outputs one encoded feature vector. Therefore the 1st encoder's output has an equivalent output frame length $\ell_1 = k_1\ell$. The encoded feature from the 1st encoder is fed to both the 1st decoder, and the 2nd encoder. To further reduce the frame rate, we add a buffer between 1st and 2nd encoders, to accumulate k_2 frames, before sending to the 2nd encoder. As a result, the 2nd encoder consumes k_2 frames per output, and its output equivalent frame length is $\ell_2 = k_2\ell_1 = k_2k_1\ell$. The 2nd encoder is streaming as well, but not causal. It has a right context of C frames w.r.t. the output, and C frames correspond to a duration of $\Gamma = C\ell_2 = C \times k_2k_1\ell$. Similar to the 1st pass, the 2nd encoder's output is sent to the 2nd decoder. As such, the two encoder-decoder pairs form the cascaded system, where the 1st pass (1st encoder + 1st decoder) is causal only whose output frame length is $k_1\ell$, while the 2nd pass looks into C future frames with an output frame length of $k_2k_1\ell$.

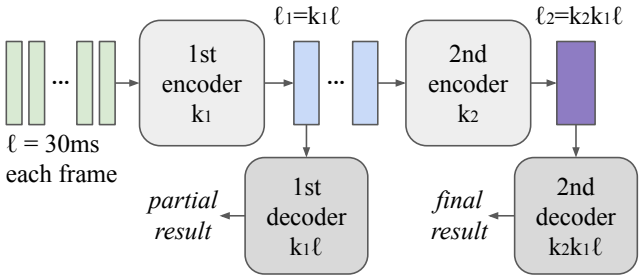

Fig. 1: Diagram of two-pass cascaded streaming ASR model.

Being causal only, the 1st pass generates hypotheses as the speech signal is ingested, and outputs the transcript from the beginning till the current time t. This unfinished transcript is called a ***partial result***. The 2nd pass runs C frames (or a delay of Γ in time) behind the 1st pass. therefore the 2nd pass is centered at $t - \Gamma$. Figure 2 illustrates the synchronization of the two passes. When the utterance ends, the 2nd pass pads fake frames (e.g. zero values, see Figure 4). After processing, the 2nd pass outputs a ***final result***, i.e. the complete utterance's transcript. Note that since the 2nd pass has future context, its ***final result*** is generally more accurate than that of the 1st pass. At a high level, when a user is speaking, ***partial result*** is emitted; upon finishing, ***final result*** is outputted or discarded, depending on the downstream applications.

In summary, k_1 and k_2 are the key hyper-parameters to control the frame rate reduction factor. The higher they are,

the longer frame length that the model uses, and the less computation needed to process the same length of the audio. The drawback is that a long frame might lose the acoustic resolution, leading to possibly worse transcription quality. Another disadvantage is that the long frame also increases the **partial latency**, which could be essential for streaming applications. In addition, the value of C (or Γ) determines the 2nd pass delay, because the 2nd pass runs C frames behind.

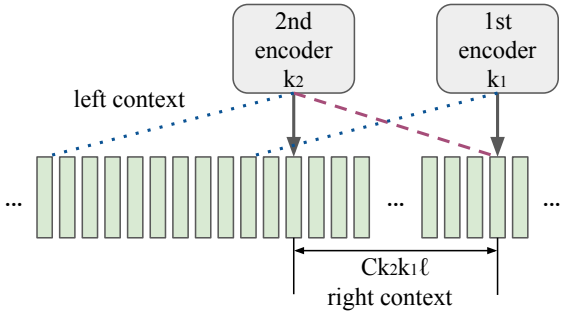

Fig. 2: The 2nd pass is running behind 1st pass by C stacked frames, an equivalent $Ck_2k_1\ell$ window length in time.

We propose two frame rate reduction strategies: 1) Increase k_1 only – both passes run at a lower frame rate. This saves computation for both passes. Another benefit is that both passes are synchronized, such that their frame-wise outputs are easier to merge (e.g. for knowledge distillation purpose). The disadvantage is that the low rate in the 1st pass delays the ***partial result*** emission, which could lead to bad user experience. Also, too much information is compressed in the early layers, such that the transcription quality may be impacted. 2) Set k_1 to be small, and increase k_2 for the 2nd pass. This setting avoids increasing k_1 to mitigate the ***partial result*** delay problem so the **partial latency** won't increase, but at a cost of less computation savings.

4. EXPERIMENTAL SETUP

4.1. Model Setup

Both passes' encoders are conformer based. Their architectures are similar: both have 6 multi-head-self-attention (MHSA) conformer layers, where each has 23 frames left contexts. Unless otherwise specified, the Funnel-attention layer (the sub-sampling layer) is placed at the first MHSA conformer layer, to maximize the computation savings (all following layers can run at a lower rate). There are 8 attention heads in each attention layer. The 1st encoder's feature dimension is 512, while the 2nd's is 640. The 1st encoder has an additional 3 layers of convolution-only conformer layers, before the 6 MHSA conformer layers. The two decoders are light-weight and RNN-T based, as introduced in [13]. Within each decoder, there is a 320-dimensional embedding prediction network, plus a 384-dimensional joint network. After the model trained with the RNN-T loss, we continue fine-tuning the model with the minimum-word-error-rate (MWER) [28] criterion. During training, FastEmit [20] is also applied.

The frontend takes 128-dimensional feature vectors from log-Mel-spectrogram (10 ms shift) as input. Contiguous 4 frames are stacked as a 512-dimensional vector, concatenated with a 16-dimensional one-hot domain-ID (indicating whether it is voice-search, dictation, or any other domain) to form a 528-dimensional input frame. The input frames are sub-sampled by a factor of 3 in the time axis. As a result, the frame fed into the 1st encoder represents a length of $\ell = 30$ ms, whose dimension is 528.

We replicate the **baseline model** in [10], which sets $k_1 = 2$ (a sub-sampling factor of 2 in the 1st pass) and $k_2 = 1$ (no further reduction in the 2nd pass). So both passes' output frame length $\ell_1 = \ell_2 = 60$ ms. The 2nd pass of the baseline model has a delay of $C = 15$ frames, an equivalent $\Gamma = 15 \times 60 = 900$ ms MHSA layers' right-context.

To deploy the model on device, we perform 8-bit post-training quantization (PTQ). In [10], PTQ is shown to have no quality gap compared to the full precision counterpart. The size of the two encoders (after PTQ) are 50 MB and 55 MB, respectively. Both decoders are 4.2 MB. Note that the training is in floating point, and all the following evaluations are using the quantized 8-bit model. The on-device benchmarking platform is an Android smartphone with 12 GB memory.

4.2. Datasets

The training data is a collection of around 1.4 million hours of English audio-text pairs. It is a multi-domain dataset, where the majority are anonymous voice-search, voice-dictation and long-form audio collections. We apply Multi-style Training (MTR) [29] and Spec-Augmentation [30] to increase the data diversity and prevent overfitting during training. To evaluate the model performance in different scenarios, the testsets are split into 3 groups that cover short to long utterances. Note that the testsets are all human transcribed. The voice-search (VS) testset has around 6k short query utterances, and the voice-dictation (VD) testset contains 20k dictation utterances. An additional 72 long-form (LF) test utterances are added to evaluate long-form recognition. Our data handling abides by *Google AI Principles* [31]. The average utterance lengths are 4.8 seconds for VS, 8.2 seconds for VD and 1032 seconds for LF. We adopt word-piece tokenization [5] for all the experiments, with a mixed-case vocabulary size of 4096.

4.3. Metrics

To evaluate the model, we define the following metrics. The main quality metric is word-error-rate: WER $= (S + I + D)/N$, where S, I, D are the number of substitutions, insertions and deletions, respectively, and N is the number of words in the reference. In the following sections, we measure the WER from 2nd pass, i.e. the ***final result***'s WER.

The benchmarking metrics are **real-time-factor** (RTF), **final latency** (fLat) and **power consumption** (Power). RTF

and fLat are defined as:

$$\text{RTF} = T_{\text{proc}}/T_{\text{utt}}, \quad \text{fLat} = t_{\text{proc\_end}} - t_{\text{utt\_end}}$$

where T_{proc}, T_{utt} are total processing time and utterance duration, respectively. Note that RTF < 1, for real-time applications. $t_{\text{proc\_end}}$ and $t_{\text{utt\_end}}$ denotes the processing end time and utterance end time. Therefore, the fLat measures the delay from the utterance end, to when the recognition is finished and the ***final result*** is generated. To reduce variance, the device's power consumption is measured when the model recognizes a long-form (14 minutes) utterance. Since smartphone platforms' power measurement is often noisy due to operating system and other background tasks, the RTF is also a good approximation of the ASR model's relative power consumption, compared to the baseline model.

The other important latency metric is the **partial latency** (pLat). Due to the lack of ground-truth alignment information, we define a proxy to compare **partial latency** across different models. Define the pLat as an average duration between ***partial result*** emitted and the utterance beginning:

$$\text{pLat} = \frac{1}{N}\sum_{i=1}^{N}(t_{\text{partial\_emit}}[i] - t_{\text{utt\_begin}})$$

where N is the number of partials emitted in an utterance.
[NOTE] pLat itself does not reveal the token level latency w.r.t. the ground-truth alignment, but we calculate the pLat difference against the baseline setting where $k_1 = 2, k_2 = 1$. The difference reflects the latency increase or decrease.
[NOTE] pLat is just a simple approximation because we don't take ground-truth into consideration. The failure case can be, for instance, if one model has many insertion errors in the beginning. That leads to many early ***partial results*** emissions, resulting in a smaller pLat. Given that the models we compare have similar WERs, the pLat can still be a reasonable approximation to show the **partial latency** differences.
[NOTE] We report the average of the RTF, fLat and pLat. fLat and pLat are in units of ms. Power is measured in mW.

5. EVALUATION RESULTS

We vary k_1 and k_2 values to achieve different frame rate reduction, and change C to control the 2nd pass delay. The mean fLat of the baseline is 139 ms. The fLat and pLat of the other models are reported using relative increase or decrease (+/- x) against the baseline, for a better comparison.

5.1. Strategy 1: Increase k_1

The first experiment examines different k_1 values while keeping $k_2 = 1$ (no further reduction in 2nd pass). This setting maximizes computation savings. In Table 1, we present the testsets' WERs as k_1 changes. We put the corresponding output frame length ℓ_1 beside the k_1 values in the table, e.g. the baseline model is with $k_1 = 2$ and operates at output frame length of 60 ms. The model with $k_1 = 4$ operates at output frame length of 120 ms.

Table 1: Strategy 1: Vary k_1, when $k_2 = 1, C = 15$. The first column is the baseline. The higher k_1, the better VS performance and slightly worse VD / LF performance. Higher k_1 offers better RTF / Power, but worse fLat and pLat.

$k_1(\ell_1)$		2 (60ms)	4 (120ms)	6 (180ms)	8 (240ms)
WER(%)	VS	5.5	5.3	5.2	5.2
	VD	3.3	3.3	3.6	3.5
	LF	15.7	15.9	18.4	17.1
RTF		0.087	0.06	0.051	0.047
Power (mW)		281	214	223	192
fLat (ms)		+0 (139ms)	+19	+25	+27
pLat (ms)		+0	+160	+387	+433

From Table 1, a clear trend is observed that WER on VS dataset decreases when k_1 increases. This is counter-intuitive, since we often expect a worse WER in a very low rate (e.g. $k_1 = 8$, equivalently 240 ms). The VD and LF testsets results in Table 1 conform to this expectation but VS data behaves as an outlier. The reason is that, VS utterances are typically short, therefore the transcription quality is more sensitive to the right-context length of the 2nd pass. As k_1 increases, the right-context duration Γ increases. For instance, when $k_1 = 8$, $\Gamma = 3.6$ s, which is close to the utterance length (averagely 4.8 s). As a consequence, the model converges to a full-context model, resulting in better quality. On the contrary, the Γ is still small relative to a longer utterance (e.g. in VD and LF), so their WERs increase as k_1 increases.

As Table 1 shows, when $k_1 = 4$, VS result is 5% better and VD / LF quality difference is negligible compared to the baseline. The RTF gets reduced from 0.087 down to 0.06, which means already a 30% computation savings. Even at $k_1 = 8$ (240 ms frame length), the WER degradation is only 6% on VD and 9% on LF datasets. But the RTF is only 0.047, around 45% reduction from the baseline.

The cost to pay is the latency increase, on both fLat and pLat. As k_1 increases, fLat get up to 30 ms increase, which is usually acceptable. The more critical problem is on the **partial latency**. Each time, the 1st pass needs to wait until all k_1 input frames have arrived, before it can start processing. Therefore, a larger k_1 delays each ***partial result***'s emit. As shown in the Table 1, when $k_1 = 8$, the pLat is increased by around 430 ms. This amount of ***partial result*** delay significantly hurts user experience in real-time applications.

Different Rate Reduction Layer Positions

In the previous experiments, the Funnel-attention layer (where rate reduction happens) is placed at the 1st MHSA conformer layer in the encoder. We can also put it at other layers. It is expected that, the later layer we push the Funnel-attention layer, the less computation savings, because layers before it still run at a higher frame rate. However, a later sub-sampling layer might benefits the quality, due to less information loss from early MHSA layers.

In particular, we place the Funnel-attention at the first

and the last MHSA conformer layer (the 6th MHSA layer), within the 1st pass encoder. In the rest of the paper, we use **2x2** to represent a down-sampling factor of 2 at the beginning Funnel-attention layer, and another down-sampling factor of 2 at the ending Funnel-attention layer. Assuming input frame length $\ell = 30$ ms, starting from 2nd MHSA layer, the frame length becomes 60 ms until the last MHSA layer, and 1st encoder output frame length becomes $\ell_1 = 120$ ms. Similarly, **2x3** means down-sampling factors 2 and 3, for the begining and ending Funnel-attention layers, respectively. As a result, **2x3**'s $\ell_1 = 180$ ms.

Constraint on Left and Right Context

It is worth pointing out that as k_1 increases, since the 2nd pass looks into C future frames, the effective future context window $\Gamma = C \times k_2 k_1 \ell$ also expands. For example, if fixing $C = 15$, when $\ell = 30$ ms, as k_1 increases from 2 to 8, Γ changes from 900 ms to 3.6 s. This huge delay between two passes could be unacceptable in certain use cases. For instance, we may want to use the 2nd pass's ***partial result*** to timely correct the streaming transcription when we perform the long-form recognition. One way to mitigate this is to reduce C, such that the right-context actual duration Γ remains similar to the baseline, around 900 ms.

We evaluate on the following settings: when $\ell_1 = 120$ ms ($k_1 = 4$), let $C = 8$ (or $\Gamma = 960$ ms); when $\ell_1 = 180$ ms ($k_1 = 6$), let $C = 5$ (or $\Gamma = 900$ ms). Similarly, we can also reduce the left-context frame number, to keep the total left-context duration of the MHSA layers to be ≈ 7.68 s. The quality and efficiency comparison are shown in Figure 3, while we present latency results in Table 2. In the following, **+RC** means that we only constrain right-context to be around 900 ms, and **+LRC** means we put constraint on both left (around 7.68 s) and right-contexts (around 900 ms).

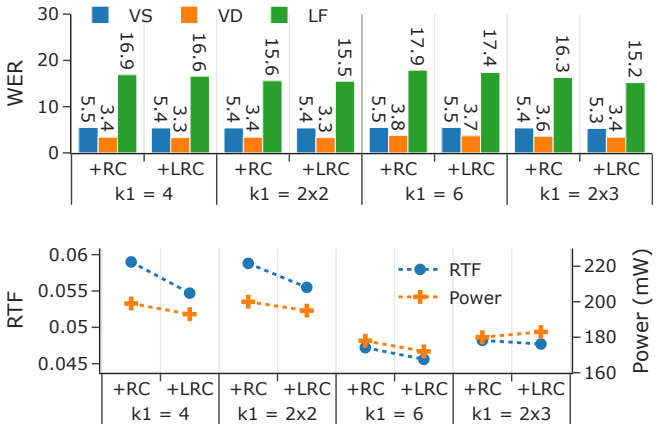

Fig. 3: 2x2 / 2x3 puts rate reduction layers at both beginning and end of the 1st encoder, and obtains better WERs than its counterpart 4 / 6. Reducing the context window, +LRC is generally better than +RC only, in both quality and efficiency.

We have two observations: 1) From Figure 3, as expected, the later we push the frame rate reduction layer, the better WERs we can obtain. 2x2's WERs are better than 4's, and 2x3's WERs are better than 6's. On the other hand, the layers before the reduction layer run at a higher rate, leading to a higher computation cost. Taking +LRC cases as the example, power consumption for $k_1 =$ 2x2 is 195 mW, versus $k_1 = 4$ at 193 mW. Similarly, 183 mW for the 2x3 setting, against 172 mW for $k_1 = 6$. 2) Figure 3 and Tabel 2 show that +RC saves computation and reduces fLat, while +LRC is even better than +RC. Reducing right-context frame number C (+RC) makes the model process fewer keys and values, leading to less computations. This speeds up the final processing when utterance ends, thus the fLat gets reduced. For instance, when $k_1 = 4$, without +RC, relative fLat is +19 ms (in Table 1, 2nd column). With +RC, it drops to -4 ms (in Table 2, 1st column). For the same reason, reducing both left and right contexts (+LRC) saves more computation. This is verified by comparing the +LRC against +RC in Figure 3's RTF / Power plot. The pLat also becomes better when +LRC, but still high due to the increase of the 1st pass frame length ℓ_1.

Table 2: Latency metrics for different reduction positions.

k_1	4		2x2		6		2x3	
	+RC	+LRC	+RC	+LRC	+RC	+LRC	+RC	+LRC
fLat	-4	**-9**	-7	**-14**	-11	**-12**	-12	**-16**
pLat	+124	**+71**	+116	**+101**	+182	**+152**	+107	**+67**

More interestingly, +LRC not only saves computation, but also boost WERs (see Figure 3's WER plot). Our hypothesis is that, the baseline model ($k_1 = 2$) already has 7.68 seconds left-context. When k_1 increases, e.g. to 6, the left-context length becomes 23 seconds. The audio 23 seconds ago, might not be helpful to the current frame. On the contrary, the long history spreads the attention weights to the less useful history frames, distracting the model from focusing on nearby important frames. This could explain the fact +LRC yields better WERs, but a complete analysis is due to future work.

To highlight, the 2x3 +LRC model obtains a slightly better WER than baseline ($k_1 = 2$) on VS (5.3 vs 5.5) and LF (15.2 vs 15.7), while similar on VD (3.4 vs 3.3). Meanwhile, it achieves 0.048 RTF (saves 45%), 183 mW power consumption (saves 35%), and -16 ms fLat (11% faster in the end). The only drawback of this candidate is the +67 ms pLat, which limits its usage on low-latency applications.

5.2. Strategy 2: Increase k_2

To satisfy the **partial latency** requirements, we propose to keep the 1st pass $k_1 = 2$, the same as the baseline, such that the ***partial results***' delay is not affected. Alternatively, we reduce the 2nd pass's frame rate by setting $k_2 = 2$ to achieve equivalent 120 ms output frame length.

Table 3 shows that, when we set $k_2 = 2$, and apply +LRC, we obtain a setting that yields the best trade-off among quality, power and latency. The same as before, +LRC (the 3rd row) is generally better than that without constraints (the 2nd

Table 3: Strategy 2: Set $k_1, k_2 = 2, 2$ ($\ell_1, \ell_2 = 60, 120$ ms). The first row is the baseline that $k_1, k_2 = 2, 1$.

	VS	VD	LF	RTF	Power	fLat	pLat
$k_2 = 1$	5.5	3.3	15.7	0.087	281	+0	+0
$k_2 = 2$	5.4	3.3	15.8	0.073	252	+0	+12
+LRC	**5.4**	3.3	**14.9**	**0.069**	**245**	**-26**	+9

row). Compared to the baseline (the 1st row), on the quality side, $k_2 = 2$ +LRC have similar (VS, VD) or better (LF) WERs. The RTF goes down from 0.087 to 0.069, and 36 mW power saving is observed. The fLat got reduced by 26 ms (nearly 20% faster in the end). The pLat remains very close to the baseline, so real-time requirements are satisfied.

5.3. Short Summary and Best of All

- For latency-sensitive scenarios, we propose $k_1 = 2, k_2 = 2$ +LRC, which has 5% WER improvement on LF, and similar WER on VS, VD. It saves 20% RTF / 13% power over the baseline. fLat decreased by 20%. pLat remains similar.
- If allowing more **partial latency** , 2x3 +LRC gives best power efficiency with 45% RTF / 35% power savings, and no sacrifice on recognition quality, at a cost of +67ms pLat.
- Limiting left and right-context (+LRC) not only reduces RTF, fLat and pLat, but also improves transcription quality.
- Later reduction layer position (e.g. 2x2, 2x3 settings) helps on WER, with some extra computation costs.

5.4. Voice Activity Detection (VAD) Segmentation

For certain use cases, e.g. voice dictation, a Voice Activity Detector (VAD) [32] could be enabled to stop ASR when long silence / pauses happen in the middle of an utterance. The benefit of VAD is to save computation and power consumption for the silence or noise periods. However, this breaks the continuous speech into segments. In order to keep the context / history for better recognition quality, upon reaching a new segment, the encoders and decoders of both passes resume from frames and hypotheses of the previous segment. Every time the cascaded system encounters a segment-end boundary, the 2nd pass needs to pad fake frames to fulfill C right contexts. But these fake frames, as the context across segments, are not seen at training time, sometimes leading to extra errors due to the discrepancy between training and inference. This segment boundary issue is illustrated in Figure 4. Possible mitigation could be: (a) training data augmentation, e.g. construct multi-segment data, with fake frames, or (b) inference-time adjustment, e.g. pad real silence frames. Nevertheless, they are beyond the scope of this paper. For simplicity, we repeat the last real frame as the padding frames.

We present a comparison between turning off (blue) and turning on VAD (+VAD, orange), in Figure 5. Note that VAD segmentation is not applied to the VS dataset, which only has one short query per utterance, so we show the WERs for VD (upper plot) and LF (bottom plot) testsets. Clearly, turning on

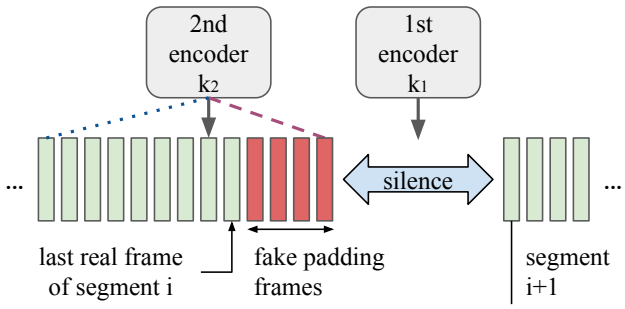

Fig. 4: At segment boundary, 2nd pass pads fake frames.

VAD (orange) causes a drop of quality. When applying +LRC (green) if VAD is on, we can observe WER improvements. We hypothesize that limiting the context window reduces the number of padding frames, thus preventing the model from spreading attention weights on the fake frames at the segment boundaries. Needless to say +LRC also saves computations as shown before.

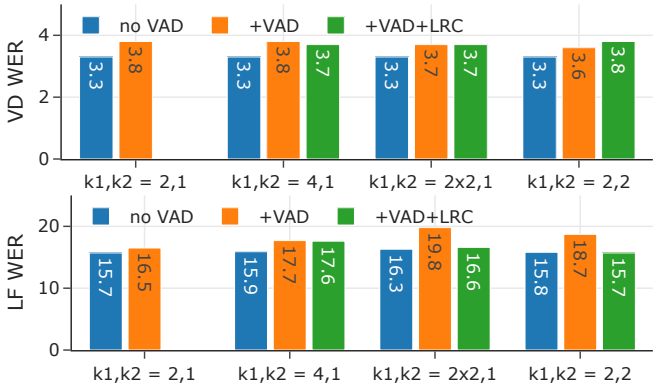

Fig. 5: Turning on VAD (orange) hurts WERs on VD (upper) / LF (bottom) testsets, for different k_1, k_2 settings. Constraint on left and right context window length (+LRC, green, not applicable when $k_1, k_2 = 2, 1$) recovers WERs slightly.

6. CONCLUSIONS AND FUTURE WORK

In this paper, we systematically investigate frame rate reduction settings in the on-device cascaded streaming ASR system. Particularly, we propose a low-latency candidate that reduces 2nd pass's frame rate to 120 ms. It achieves better quality and efficiency at the same time, without hurting streaming latency. We propose another low-power candidate whose frame length is 180 ms starting from 1st pass. Though this setting increases partial latency, it saves 35% power without quality regression. We also have the following observations: 1. Up to 240 ms frame length, no significant quality drop is observed. 2. Limiting left and right-context window duration, generally helps on all three aspects of quality, power and latency. 3. Turning on VAD introduces a segment boundary issue, leading to worse quality. Limiting the left and right-context, again can help. We defer a deeper analysis on why reducing context window is beneficial, to the future work.

7. REFERENCES

[1] Jinyu Li, "Recent advances in end-to-end automatic speech recognition," *APSIPA Transactions on Signal and Information Processing*, vol. 11, no. 1, 2022.

[2] Dong Wang, Xiaodong Wang, and Shaohe Lv, "An overview of end-to-end automatic speech recognition," *Symmetry*, vol. 11, no. 8, pp. 1018, 2019.

[3] Jinyu Li, Yu Wu, Yashesh Gaur, Chengyi Wang, Rui Zhao, and Shujie Liu, "On the comparison of popular end-to-end models for large scale speech recognition," *arXiv preprint arXiv:2005.14327*, 2020.

[4] Alex Graves, "Sequence transduction with recurrent neural networks," *arXiv preprint arXiv:1211.3711*, 2012.

[5] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al., "Google's neural machine translation system: Bridging the gap between human and machine translation," *arXiv preprint arXiv:1609.08144*, 2016.

[6] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al., "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal processing magazine*, vol. 29, no. 6, pp. 82–97, 2012.

[7] Tara N Sainath, Yanzhang He, Bo Li, Arun Narayanan, Ruoming Pang, Antoine Bruguier, Shuo-yiin Chang, Wei Li, Raziel Alvarez, Zhifeng Chen, et al., "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)*. IEEE, 2020, pp. 6059–6063.

[8] Jinyu Li, Rui Zhao, Zhong Meng, Yanqing Liu, Wenning Wei, Sarangarajan Parthasarathy, Vadim Mazalov, Zhenghao Wang, Lei He, Sheng Zhao, et al., "Developing rnn-t models surpassing high-performance hybrid models with customization capability," *arXiv preprint arXiv:2007.15188*, 2020.

[9] Golan Pundak and Tara N Sainath, "Lower frame rate neural network acoustic models," *Interspeech 2016*, pp. 22–26, 2016.

[10] Shaojin Ding, Weiran Wang, Ding Zhao, Tara N Sainath, Yanzhang He, Robert David, Rami Botros, Xin Wang, Rina Panigrahy, Qiao Liang, et al., "A unified cascaded encoder asr model for dynamic model sizes," *arXiv preprint arXiv:2204.06164*, 2022.

[11] Rohit Prabhavalkar, Takaaki Hori, Tara N Sainath, Ralf Schlüter, and Shinji Watanabe, "End-to-end speech recognition: A survey," *arXiv preprint arXiv:2303.03329*, 2023.

[12] Tara N Sainath, Ruoming Pang, David Rybach, Yanzhang He, Rohit Prabhavalkar, Wei Li, Mirkó Visontai, Qiao Liang, Trevor Strohman, Yonghui Wu, et al., "Two-pass end-to-end speech recognition," *arXiv preprint arXiv:1908.10992*, 2019.

[13] 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.

[14] Yongqiang Wang, Zhehuai Chen, Chengjian Zheng, Yu Zhang, Wei Han, and Parisa Haghani, "Accelerating rnn-t training and inference using ctc guidance," in *ICASSP 2023-2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2023, pp. 1–5.

[15] Wenyong Huang, Wenchao Hu, Yu Ting Yeung, and Xiao Chen, "Conv-transformer transducer: Low latency, low frame rate, streamable end-to-end speech recognition," *arXiv preprint arXiv:2008.05750*, 2020.

[16] Mohammed Kyari Mustafa, Tony Allen, and Kofi Appiah, "A comparative review of dynamic neural networks and hidden markov model methods for mobile on-device speech recognition," *Neural Computing and Applications*, vol. 31, pp. 891–899, 2019.

[17] Yanzhang He, Tara N Sainath, Rohit Prabhavalkar, Ian McGraw, Raziel Alvarez, Ding Zhao, David Rybach, Anjuli Kannan, Yonghui Wu, Ruoming Pang, et al., "Streaming end-to-end speech recognition for mobile devices," in *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2019, pp. 6381–6385.

[18] Arun Narayanan, Tara N Sainath, Ruoming Pang, Jiahui Yu, Chung-Cheng Chiu, Rohit Prabhavalkar, Ehsan Variani, and Trevor Strohman, "Cascaded encoders for unifying streaming and non-streaming asr," in *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2021, pp. 5629–5633.

[19] Ian Williams, Anjuli Kannan, Petar S Aleksic, David Rybach, and Tara N Sainath, "Contextual speech recognition in end-to-end neural network systems using beam search.," in *Interspeech*, 2018, pp. 2227–2231.

[20] Bo Li, Anmol Gulati, Jiahui Yu, Tara N Sainath, Chung-Cheng Chiu, Arun Narayanan, Shuo-Yiin Chang, Ruoming Pang, Yanzhang He, James Qin, et al., "A better and faster end-to-end model for streaming asr," in *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2021, pp. 5634–5638.

[21] 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.

[22] Pengcheng Guo, Florian Boyer, Xuankai Chang, Tomoki Hayashi, Yosuke Higuchi, Hirofumi Inaguma, Naoyuki Kamo, Chenda Li, Daniel Garcia-Romero, Jiatong Shi, et al., "Recent developments on espnet toolkit boosted by conformer," in *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2021, pp. 5874–5878.

[23] Daniel Povey, Piotr Zelasko, and Sanjeev Khudanpur, "Speech recognition with next-generation kaldi (k2, lhotse, icefall)," *Interspeech: tutorials*, 2021.

[24] Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber, "Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks," in *Proceedings of the 23rd international conference on Machine learning*, 2006, pp. 369–376.

[25] William Chan, Navdeep Jaitly, Quoc Le, and Oriol Vinyals, "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)*. IEEE, 2016, pp. 4960–4964.

[26] Md Akmal Haidar, Chao Xing, and Mehdi Rezagholizadeh, "Transformer-based asr incorporating time-reduction layer and fine-tuning with self-knowledge distillation," *arXiv preprint arXiv:2103.09903*, 2021.

[27] Zihang Dai, Guokun Lai, Yiming Yang, and Quoc Le, "Funnel-transformer: Filtering out sequential redundancy for efficient language processing," *Advances in neural information processing systems*, vol. 33, pp. 4271–4282, 2020.

[28] Rohit Prabhavalkar, Tara N Sainath, Yonghui Wu, Patrick Nguyen, Zhifeng Chen, Chung-Cheng Chiu, and Anjuli Kannan, "Minimum word error rate training for attention-based sequence-to-sequence models," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2018, pp. 4839–4843.

[29] Chanwoo Kim, Ananya Misra, Kean Chin, Thad Hughes, Arun Narayanan, Tara N Sainath, and Michiel Bacchiani, "Generation of large-scale simulated utterances in virtual rooms to train deep-neural networks for far-field speech recognition in google home," *Proc. Interspeech 2017*, pp. 379–383, 2017.

[30] Daniel S Park, William Chan, Yu Zhang, Chung-Cheng Chiu, Barret Zoph, Ekin D Cubuk, and Quoc V Le, "Specaugment: A simple data augmentation method for automatic speech recognition," *arXiv preprint arXiv:1904.08779*, 2019.

[31] Google, "Artificial intelligence at google: Our principles," 2018.

[32] Shaojin Ding, Rajeev Rikhye, Qiao Liang, Yanzhang He, Quan Wang, Arun Narayanan, Tom O'Malley, and Ian McGraw, "Personal vad 2.0: Optimizing personal voice activity detection for on-device speech recognition," *arXiv preprint arXiv:2204.03793*, 2022.