WIND: Accelerated RNN-T Decoding with Windowed Inference for Non-blank Detection

Anonymous submission to Interspeech 2025

Abstract

We propose Windowed Inference for Non-blank Detection (WIND), a novel strategy that significantly accelerates RNN-T inference without compromising model accuracy. During model inference, instead of processing frames sequentially, WIND processes multiple frames simultaneously within a window in parallel, allowing the model to quickly locate non-blank predictions during decoding, resulting in significant speed-ups. We implement WIND for greedy decoding, batched greedy decoding with label-looping techniques, and also propose a novel beam-search decoding method. Experiments on multiple datasets with different conditions show that our method, when operating in greedy modes, speeds up as much as 2.4X compared to the baseline sequential approach while maintaining identical Word Error Rate (WER) performance. Our beam-search algorithm achieves slightly better accuracy than alternative methods, with significantly improved speed. We will open-source our WIND implementation.

Index Terms: speech recognition, RNN-T, RNN-Transducers, parallel computation

1. Introduction

In the recent decade or so, End-to-end automatic speech recognition (ASR) has seen remarkable progress, with RNN-T [1] emerging as one of the dominant architectures for speech applications. Various open-source toolkits offer high-quality implementations for RNN-T models, including ESPnet [2], SpeechBrain [3] and NeMo [4] etc. Compared alternative methods like CTC [5], RNN-T achieves improved accuracy at increased computational cost. As deployment scenarios become more diverse and demanding, there is an increasing need to improve RNN-T's efficiency without compromising its recognition accuracy. A significant computational bottleneck in RNN-T inference stems from its frame-by-frame processing mechanism during decoding, where the model loops over all input frames, with majority of decoding steps predicting blank tokens. Blank predictions of RNN-T models are time-consuming but do not contribute to the final decoding outputs, marking significant waste.

In this paper, we introduce *Windowed Inference for Non-blank Detection* (WIND), a novel parallel frame processing strategy that accelerates RNN-T inference without compromising accuracy. Instead of processing frames sequentially, WIND processes multiple frames simultaneously within a configurable window, allowing the model to locate the next non-blank symbol efficiently, significantly improving the inference speed of RNN-T models with minimal memory overhead. The key contributions of this work are:

1. We propose the WIND strategy for RNN-T inference, which uses parallelized computation on a window of frames to quickly detect non-blanks.
2. Greedy (batch=1) WIND achieves up to 2.40X inference speed-up compared to the standard algorithm.
3. Batched greedy WIND achieves up to 1.26X speed-up to highly optimized label-looping methods with CUDA graphs.
4. Beam-search WIND archives significantly better speed-accurcay tradeoff than alternative methods like alsd, maes.
5. WIND can combine with alternative methods like CUDA-graphs to further optimize RNN-T inference.
6. All evaluations in this paper use publicly accessible checkpoints and public data. We will open-source our WIND implementations so that the results can be fully reproduced.

2. Related Work

Considerable work has investigated speeding up the inference of Transducer models. One way to do that is through improvement in the model architecture. For example, [6] replaced the RNN components of the model decoder with a stateless network, delivering consistent model speedup with small performance degradation. [7] proposed a multi-blank Transducer, where blank tokens can cover multiple frames, and therefore reducing the number of decoding steps; [8] proposed a Token-and-Duration Transducer, which jointly predicts labels and durations that the label covers, further reducing the number of decoding steps. [9] proposed a stochastic masking mechanism during Transducer model training, so that the model can also support non-autoregressive decoding without running the decoder. [10] proposed a way to jointly train CTC and RNN-T model, and use CTC's blank prediction to run pruning on the RNN-T decoding, achieving speedup during inference.

Another direction is to improve the RNN-T efficiency through algorithmic improvement during inference. CUDA Graphs [11] improves the efficiency of Transducer inference by reducing the CUDA launch time. Label-looping algorithm [12] synchronizes decoder operations during batched inference and brings significant speedups. Regarding beam-search for RNN-T models, in addition to the original algorithm proposed in [1], some more efficient variants, e.g. alignment-length synchronous decoding [13], modified adaptive-expansion search [14], one-step constrained search [15], token-wise beam search [16] etc, achieve faster inference by employing different model assumptions to reduce the search space.

3. Method

3.1. Background: RNN-T

An RNN-T model consists of an encoder, a decoder, and a joiner. The encoder and decoder extract high-level informa-

Algorithm 1 RNN-T Inference Algorithm

```
input: encoder output enc [T, D]
hyp = []
t = 0
while t < T do
    dec = decoder(hyp)
    joined = joiner(enc[t,:], dec[-1])
    label = argmax(joined)
    if label is not blank then
        hyp.append(label)
    else
        t += 1
return hyp
```

tion from acoustic frames and textual history, respectively, and the joiner combines the information from the two components to generate a probability distribution over the vocabulary. The vocabulary of an RNN-T model can be words, graphemes, or subwords (e.g. byte-pair encoding [17, 18]), and must include a special "blank" token. The interaction of tokens in the vocabulary and the inference process is as follows,

1. when a non-blank symbol is predicted, the symbol is added to the output; meanwhile, the model stays at the same acoustic frame t for the next decoding step, and updates its decoder representation by feeding the symbol to its decoder;
2. when a blank symbol is predicted, the model keeps its decoder representation unchanged, and increments t by one to access the next acoustic frame.

The standard inference procedure for RNN-T models is shown in Algorithm 1. The algorithm scans the encoder output from left to right in a frame-by-frame fashion. For any frame, it uses the joint network to combine its information with the decoder output computed from the current partial hypothesis and generate a probability distribution over vocabulary.

3.2. Greedy Inference with WIND Models

Algorithm 2 WIND Algorithm

```
input: encoder output enc [T, D]
hyp = []
t = 0
while t < T do
    dec = decoder(hyp)
    n = min(window-size, len(enc) - t)
    joined = joiner(enc[t:t + n,:], dec[-1]) # [n, V]
    labels = argmax(joined, dim=-1) # [n]
    if labels are all blanks then
        t += n
    else
        i = smallest index so that labels[i] != blank
        hyp.append(labels[i])
        t += i
return hyp
```

We notice that in Algorithm 1, the decoder representation would only change after a non-blank symbol is predicted; if consecutive blanks are predicted, all those steps reuse the same decoder representation. We recognize this as a potential for efficiency improvements, and propose to parallelize the computation of those frames with Algorithm 2. At any decoding step, we use the joiner to combine the information of current decoder representation, with multiple consecutive acoustic frames within a window. This allows us to locate non-blank frames faster, reducing the decoding time. The algorithm can be easily applied to batched inference, including the more efficient label-looping batched algorithms [12]. Due to space constraints, we omit the algorithm description and refer the readers to our open-source implementation for details.

3.3. A Novel Beam-search Algorithm for WIND

Algorithm 3 WIND Beam-search Algorithm

```
input: encoder output enc [T, D], window N, beam K
t2hyps = {0: [Hyp(tokens=[], score=0.0)]}
while t2hyps is not empty do
    t = min(t2hyps.keys())
    if t == T then
        break
    hyps = t2hyps.pop(t) # read and remove
    hyps = recombine_prune_prefix_search(hyps)
    n = min(N, T - t)
    windows = enc[t:t+n,:] # [n, D]
    joined = joiner(windows, dec_state(hyps)) # [B, n, V]
    compute P'_b(v, t) for all V, t, and b
    labels, jumps = topk_for_each_b(P'_b(v, t), K) # both of shape [B, K]
    for b in range(B) do
        for k in range(K) do
            new_hyp = hyps[b]
            new_token = labels[b][k]
            jump = jumps[b][k]
            if new_token is not blank then
                new_hyp.tokens.append(new_token)
                new_hyp.scores += log(P_b(new_token, jump))
                t2hyps[t + jump].append(new_hyp)
            else
                new_hyp.scores += log(P_b(blank, n-1))
                t2hyps[t + n].append(new_hyp)
return best_hyp_in(t2hyps[T])
```

Given a vocabulary set $\mathcal{V}$ (including blank, represented as $\emptyset$), a window size w, and say we are processing the acoustic frame t_0. Based on the current decoder state, the RNN-T model computes a probability distribution of $P(v|t_0 + t)$ for $v \in \mathcal{V}$ and $t \in \{0, 1, ..., w - 1\}$. Now we define a new probability distribution of $P'(v, t)$ as the "probability of predicting v as the first output in the window at time $t_0 + t$". This probability is interpreted as,

1. for $v = \emptyset$, it must be predicted at the last frame of the window, and the probability is computed as product of probability of predicting blanks at *all* frames in the window;
2. for $v \neq \emptyset$, it is the probability of emitting all blanks for frames $t_0, t_0 + 1, ..., t_0 + t - 1$, and then emitting v at $t_0 + t$.

With this definition, $P'(v, t)$ can be computed as,

$$P'(v,t) = \begin{cases} 0 & v = \emptyset, t < w-1 \\ \prod_{t'=t_0}^{t_0+w-1} P(\emptyset|t') & v = \emptyset, t = w-1 \\ P(v|t_0+t) \prod_{t'=t_0}^{t_0+t-1} P(\emptyset|t'), & v \neq \emptyset \end{cases} \tag{1}$$

With $P'(v, t)$ defined, we present our WIND-based beam-search for in Algorithm 3[1]. We represent a hypothesis with a

[1]For simplicity, we omit some details regarding safeguard code to

Table 1: *Librispeech Word-Error-Rates (WER%) and decoding time (seconds) of Parakeet-RNNT-1.1b and -0.6b models at different window-sizes. Window-size=1 is the standard (baseline) algorithm against which relative speedup is measured.*

model	window	WER %	time (s)	rel. speed-up
1.1b	1	2.70	176	-
	2	2.70	122	1.44X
	4	2.70	91	1.93X
	8	2.70	83	2.12X
	16	2.70	83	2.12X
0.6b	1	3.31	161	-
	2	3.31	105	1.53X
	4	3.31	74	2.18X
	8	3.31	67	2.40X
	16	3.31	67	2.40X

tuple of (List(int), float), where the list is the sequence of tokens and float is the score. The algorithm maintains a t2hyps map, storing partial hypotheses that end at a specific time step. During processing, it takes the hypotheses associated with the smallest time steps t, processes them (more on this later). Then, for each of the hypotheses we compute the top-beam expansions in the window enc[t:t+n,:]. Note that at line 13, the topk function selects the best candidates from all $|\mathcal{V}| \times n$ possible combinations using $P'(v, t)$, allowing the algorithm to jump multiple frames during the search. For each of the chosen expansions, it updates the hypotheses and adds them t2hyps at the correct timestep for later processing. Once the Algorithm reaches the end, it returns the hypothesis with the best score.[2]

The "recombine_prune_prefix_search" function at line 8 performs the following steps,

1. it checks if there are duplicate hypotheses, and if so, combine them into one with their probabilities summed.
2. for all hypothesis pairs (A, B) where A is a prefix of B, it computes the probability of "completing A into B" by emitting extra symbols at the current frame; it then removes A and adds the extra probability mass to B. Note that this function was originally proposed in Section II-B and lines 5-7 of Algorithm 1 of [14], where readers can find more details.
3. it keeps top beam hyps in the set.

4. Experiments

We conduct our experiments using Conformer-RNNT [19, 20] implementation from the NeMo [4] toolkit. All models are public checkpoints built on 80-dimensional filter bank features extracted at 25ms frames with 10ms strides, and BPE of size 1024 as text representation. To measure inference time for all experiments, we first do a "warm-up" run, and then run three consecutive decoding runs and take the average time for those three runs with two A6000 GPUs.

avoid the loop to stay at the same time-stamp infinitely. Readers are referred to our open-sourced implementation for such details.

[2]Note, our proposed algorithm shares similarities with *token-wise beam-search* [16] in using a window (chunk) of frames as the unit of computation; other than that, the decoding algorithms of the two methods are very different. In particular, a major difference is that [16] aggregates the emissions from chunks, while in our algorithm we still attribute emissions to a specific frame within the window, so we can still generate accurate time-stamps from the beam-search.

Table 2: *Parakeet models' decoding time (seconds) comparison between the original label-looping VS WIND version of label-looping batched greedy inference. Window-size=8 for the WIND algorithm. Evaluation is done on librispeech test-other.*

model	batch	baseline	WIND	rel. speed-up
1.1b	2	142	120	1.18X
	4	91	77	1.18X
	8	66	57	1.16X
	16	54	48	1.13X
0.6b	2	105	87	1.21X
	4	68	54	1.26X
	8	48	39	1.23X
	16	38	32	1.19X

Table 3: *Parakeet-RNNT-0.6b's WER and decoding time (seconds) of different beam-search algorithms for English Librispeech test-other (up) and Slurp test (down).*

	greedy		beam=2		beam=3		beam=4	
	WER	time	WER	time	WER	time	WER	time
alsd	3.31	161	3.24	297	3.24	384	3.25	470
maes			3.29	169	3.29	185	3.28	204
beam			3.26	388	3.26	600	3.26	813
wind	3.31	67	3.25	105	3.26	114	**3.23**	120
alsd	17.93	319	17.77	622	17.74	774	17.82	949
maes			17.83	357	**17.72**	394	17.73	435
wind	17.93	139	17.80	221	**17.72**	254	17.79	283

4.1. Greedy Decoding

In Table 1, we compare WIND with standard RNN-T inference algorithms in greedy modes. We use Parakeet-RNNT-0.6b and 1.1b models (`hf.co/nvidia/parakeet-rnnt-0.6b` and `hf.co/nvidia/parakeet-rnnt-1.1b`). We report decoding results with different window-sizes, where window-size=1 represents the standard RNN-T algorithm. As we can see, in all settings the WIND algorithm achieves identical WER compared to the standard algorithm, at significantly improved speed. Larger window-size would bring larger speedups although the effect plateaus for window-size 8 and up.

4.2. Batched Greedy Results

The results of batched greedy search are shown in Table 2. Note, we omit the WER numbers since they are identical. The relative speedup is computed as the ratio between time of the standard algorithm and the WIND algorithm. Both algorithms use the label-looping [12] methods. We can see that WIND algorithm brings consistent speedup to the standard algorithm, with the effect more pronounced for relatively smaller batches.

4.3. Beam Search Results

Since beam-search algorithms can change the WER of different models, to provide a better picture of how WIND beam-search compares to alternative methods, we report our beam-search results on multiple datasets across languages, including Voxpopuli and Multilingual Librispeech for German, and Librispeech test-other and Slurp for English. For German, we use the

Table 4: *German ASR model* `stt_de_conformer_transducer_large`*'s WER and decoding time (seconds) of different beam-search algorithms on German Voxpopuli (up) and Multilingual Librispeech (down).*

	greedy		beam=2		beam=3		beam=4	
	WER	time	WER	time	WER	time	WER	time
alsd	8.85	130	8.76	606	8.63	719	8.63	871
maes			8.69	217	8.69	254	**8.64**	281
wind	8.85	61	8.72	109	**8.64**	124	**8.64**	135
alsd	3.85	437	3.85	1840	3.80	2127	3.82	2556
maes			3.87	620	3.83	718	3.80	780
wind	3.85	167	3.85	302	**3.78**	351	3.81	382

publicly accessible model from `hf.co/nvidia/stt_de_conformer_transducer_large`. Tables 3 and 4 show the results where we compare with the original beam-search (beam) [1], *alignment-length synchronous decoding* (alsd) [13] and *modified adaptive expansion search* (maes) [14]. We include "greedy" decoding results as well for context.

We can see the WIND beam-search achieves better speed-accuracy tradeoff than all alternative methods. Since the Librispeech test-other numbers show that "beam" takes significantly more time to run without performance gains, we don't include it for other experiments. Notably, WIND with beam=4 runs faster than all alternative methods with beam=2; in 3 of the 4 cases, it's faster than (Non-WIND) greedy decoding. We also see that for all datasets, WIND can achieve the best accuracy (**bold** numbers) among all methods, and in terms of a tie, WIND uses significant less time than alternative methods.

5. Analysis

5.1. Memory Footprint

Due to the decoding window used by the WIND algorithm, more GPU memory usage of the WIND algorithm is expected. However, our empirical studies reveal that the WIND algorithm does not use noticeably more memory. Note, our models consist hundreds of millions of parameters, which require Gigabytes of GPU memory for storing model weights and hidden activations; the WIND algorithm only increases memory usage by [window-size $\times$ vocabulary-size]. In our experiments, this is at most $16 \times 1025 = 16400$ floats, which amounts to around 63 Kb of memory if using float32 datatypes. Therefore, this added memory overhead is negligible.

5.2. Jump Distribution

We plot the distributions of different jump sizes for WIND inference with different window sizes, as shown in Figure 1. We see that with window-size 1 and 2, a significant probability mass is allocated to the longest possible jump; once the window-size becomes larger, the inference would still use them as needed but the the long jumps are relatively less used. The overall distribution for window-sizes 8 and 16 are not very different, which explains their similar decoding speed previously shown in Tables 1 for those larger window-sizes.

Figure 1: *Jump distributions of WIND inference with Parakeet-RNNT-1.1b model on Librispeech test-other. Bars are ordered so that smaller jumps are on the left, and the larger right, so readers viewing this on black-printed paper can infer which bar represents which jump interval without using color information.*

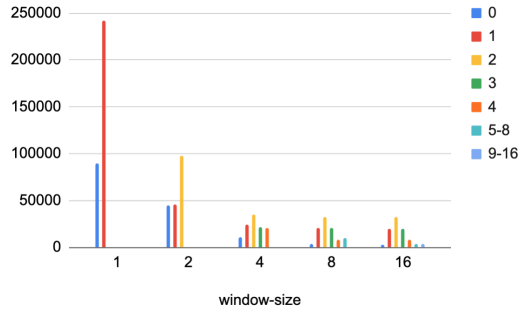

Table 5: *Decoding time of different decoding algorithms for batched greedy search, with Parakeet 1.1b RNN-T and TDT models, with and without CUDA-graph. Batch=16 for all runs. The WIND inference uses a window size of 8. "Decoder-only time" refers to the time used for inference excluding the encoder computation.*

model-decoding	total-time	decoder-only time
RNN-T	53.91	12.45
+ cuda-graph	45.94	4.23
RNN-T WIND	48.05	6.45
+ cuda-graph	43.85	2.10
TDT	48.49	6.93
+ cuda-graph	44.00	2.14

5.3. Incorporation wth CUDA-graphs

CUDA-graph [11] is a technique to speedup GPU processing by reducing GPU kernel launch time. We implement CUDA-graph optimization for label-looping batched WIND inference, and show their comparison in Table 5. To provide more context, we include results for a TDT model [8] as well, downloaded from `hf.co/nvidia/parakeet-tdt-1.1b`. We can see that while CUDA-graph optimization brings significant speedup for inference in all cases, WIND brings consistent further speedup with or without CUDA-graphs, especially when we compare decoder-only time. We also see that WIND inference with RNN-T is faster than TDT inference, indicating its strength in improving Transducer model efficiency without the need to retrain a model with a modified loss function.

6. Conclusion and Future Work

In this paper, we propose an Windowed Inference for Non-blank Detection (WIND) strategy, which helps improve decoding speed of RNN-T for both greedy, batched greedy inference; we also propose a novel WIND beam-search method that achieves better speed-accuracy tradeoff than alternative beam-search methods. In the future, we will also develop a more efficient WIND algorithm that combines beam search and batching.

7. References

[1] A. Graves, "Sequence transduction with recurrent neural networks," in *ICML*, 2012.

[2] S. Watanabe, T. Hori, S. Karita, T. Hayashi, J. Nishitoba *et al.*, "ESPnet: End-to-end speech processing toolkit," in *Interspeech*, 2018.

[3] M. Ravanelli, T. Parcollet, P. Plantinga, A. Rouhe *et al.*, "SpeechBrain: A general-purpose speech toolkit," in *Interspeech*, 2021.

[4] O. Kuchaiev, J. Li, H. Nguyen *et al.*, "Nemo: a toolkit for building ai applications using neural modules," in *NeurIPS Workshop on Systems for ML*, 2019.

[5] A. Graves, S. Fernández, F. Gomez, and J. Schmidhuber, "Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks," in *ICML*, 2006.

[6] M. Ghodsi, X. Liu, J. Apfel, R. Cabrera, and E. Weinstein, "RNN-Transducer with stateless prediction network," in *ICASSP*, 2020.

[7] H. Xu, F. Jia, S. Majumdar, S. Watanabe, and B. Ginsburg, "Multi-blank transducers for speech recognition," *arXiv:2211.03541*, 2022.

[8] H. Xu, F. Jia, S. Majumdar, H. Huang, S. Watanabe, and B. Ginsburg, "Efficient sequence transduction by jointly predicting tokens and durations," in *ICML*, 2023.

[9] H. Xu, T. M. Bartley, V. Bataev, and B. Ginsburg, "Three-in-one: Fast and accurate transducer for hybrid-autoregressive asr," *arXiv preprint arXiv:2410.02597*, 2024.

[10] Y. Yang, X. Yang, L. Guo, Z. Yao, W. Kang, F. Kuang, L. Lin, X. Chen, and D. Povey, "Blank-regularized ctc for frame skipping in neural transducer," *arXiv preprint arXiv:2305.11558*, 2023.

[11] D. Galvez, V. Bataev, H. Xu, and T. Kaldewey, "Speed of light exact greedy decoding for rnn-t speech recognition models on gpu," *arXiv:2406.03791*, 2024.

[12] V. Bataev, H. Xu, D. Galvez, V. Lavrukhin, and B. Ginsburg, "Label-looping: Highly efficient decoding for transducers," *arXiv:2406.06220*, 2024.

[13] G. Saon, Z. Tüske, and K. Audhkhasi, "Alignment-length synchronous decoding for RNN transducer," in *ICASSP*, 2020.

[14] J. Kim, Y. Lee, and E. Kim, "Accelerating rnn transducer inference via adaptive expansion search," *IEEE Signal Processing Letters*, vol. 27, pp. 2019–2023, 2020.

[15] J. Kim and Y. Lee, "Accelerating RNN transducer inference via one-step constrained beam search," *arXiv:2002.03577*, 2020.

[16] G. Keren, "A token-wise beam search algorithm for rnn-t," in *2023 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*. IEEE, 2023, pp. 1–8.

[17] R. Sennrich, B. Haddow, and A. Birch, "Neural machine translation of rare words with subword units," in *Proc. of the 54th Annual Meeting of the ACL*, 2015.

[18] T. Kudo and J. Richardson, "Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing," *arXiv:1808.06226*, 2018.

[19] A. Gulati, J. Qin, C.-C. Chiu *et al.*, "Conformer: Convolution-augmented transformer for speech recognition," in *Interspeech*, 2020.

[20] D. Rekesh, N. R. Koluguri, S. Kriman, S. Majumdar, V. Noroozi *et al.*, "Fast Conformer with linearly scalable attention for efficient speech recognition," in *Automatic Speech Recognition and Understanding Workshop (ASRU)*, 2023.