Rheinisch-Westfälische Technische Hochschule Aachen
Lehrstuhl für Informatik 6
Prof. Dr.-Ing. Hermann Ney

Seminar Selected Topics in Human Language Technology and Machine Learning in WS 2022

Non-Autoregressive Automatic Speech Recognition

George Farah

Student Number 421409

24.03.2022

Adviser: Mohammad Zeineldeen

Contents

1 Introduction

In this section, we will introduce Automatic Speech Recognition (ASR) as a sequence to sequence modeling. We will also introduce the concepts of transformers and self-attention that revolutionized sequence to sequence modeling. Moreover, we will explain why the transformers architecture is considered an Autoregressive (AR) model and why they have drawbacks that led to the introduction of Non-Autoregressive (NAR) transformers. At the end of this section, we will introduce Connectionist Temporal Classification (CTC).

1.1 Automatic Speech Recognition

Automatic Speech Recognition is the process of transforming input sound or speech signal into its corresponding sequence of words (see figure (1)) [1]. ASR has a huge variety of applications such as voice dialing, voice search, and digital assistance (e.g, Siri, Cortana, and Google Assistant). [1]

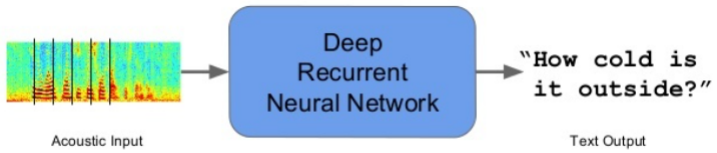

Figure 1: A basic example of ASR on input speech to predict the sentence "how cold is It outside?" [2]

To achieve better sequence to sequence modeling performance in ASR in the age of big data, neural network approaches have been implemented [1] such as Recurrent Neural Networks (RNNs) [3] and Long Short Term Memory (LSTM) Neural Networks [4].
It is important to mention that before the input speech goes into the predictive model (e.g RNNs), it has to go through preprocessing and filtering steps to extract the acoustic features from the signal[1]. One example of these representations are the Mel Frequency Cepstral Coefficients [1].

1.2 Transformers

In 2017 Vaswani et al. introduced the concept of transformers [5] which presented a new model architecture using stacked self-attention and point wise, fully connected layers for both encoder and decoder [5], (see figure (2)).
Transformers architecture has revolutionized sequence to sequence modeling and has brought great advantages in comparison to LSTMs or RNNs, [6].
This is mainly due to the introduced self-attention mechanism that relates the different positions of a single sequence and drastically enhanced the learning of long-ranged dependencies in the sequence. Also, the architecture of the self-attention layers themselves allow for parallelization in the training phase and therefore speed up the training process. [5].

1.2.1 Self-Attention

In the general sense, an attention function maps a query and pairs of key-value to an output, where the query, keys, and values are all vectors. These vectors encode the

relevance that a token has to other tokens in the sequence. This mechanism enhances the learning of the long-range dependencies in the sequence. In the original Transformer papers [5] a Multi-Head Attention Mechanism is introduced based on a "Scaled Dot-Product Attention" mechanism. The scaled dot product attention input consists of the queries and keys of dimensionality d_k and values of dimension d_v. The dot product is computed between all query, keys values, then it's divided by the square root of the dimensionality. Afterward, the softmax function is applied and multiplied by the values to obtain the weights on the values:

$$Attention(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V \tag{1}$$

Where Q,K,V are query, keys and values respectively in matrix representation, where each row of these matrices consists of the query, key, and value vectors for each character.
Multi-Head Attention utilizes the dot product attention mechanism to perform attention functions in parallel. The main idea behind it is to linearly project the queries, keys, and values h times using learned linear projections to their respective dimensionality, then the dot product attention is performed in parallel for each input token of the sequence. (See figure (3)).

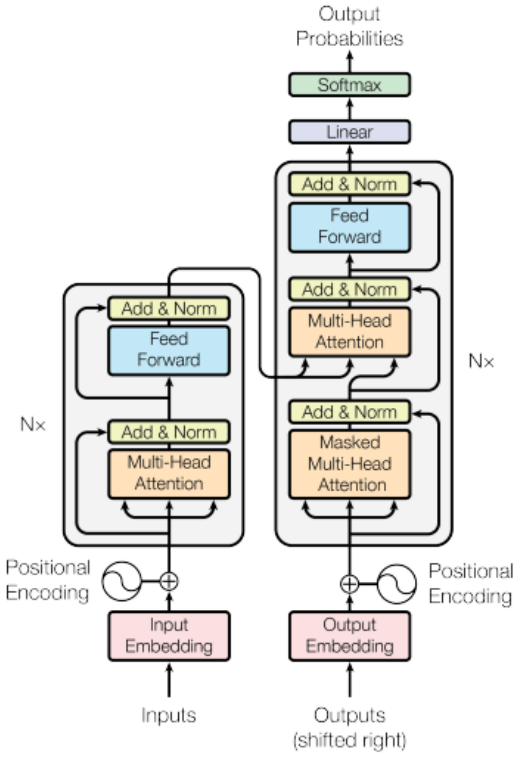

Figure 2: The Transformer - model architecture [5]

Transformers have delivered better results in performance and predictions across all applications and have become the standard in sequential modeling.[7] [8]

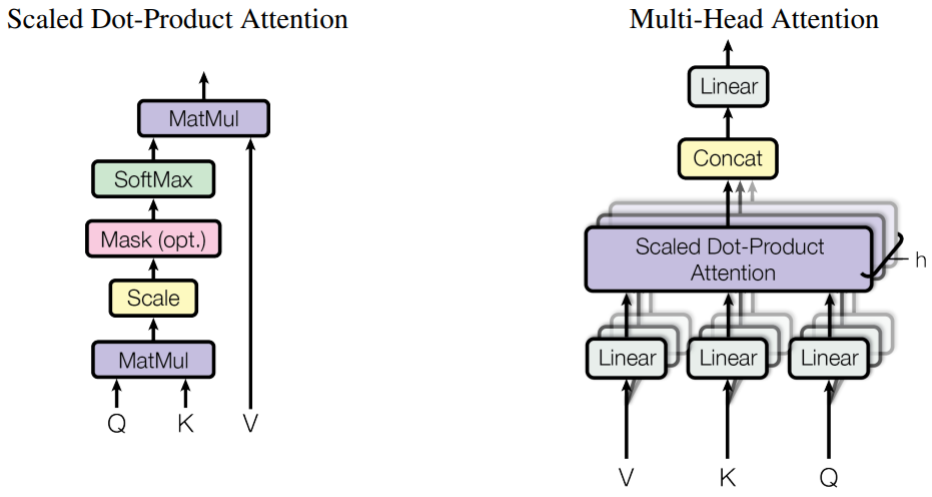

Figure 3: The Scaled Dot product and Multi-Head Attention introduced in the original transformer paper [5]

1.3 Non-Autoregressive Transformers

In the previous section we have discussed the speed-up that the transformers bring in the training phase, but what about the inference phase? As it turns out the transformers predict the sequence in an autoregressive manner, which is called autoregressive transformers (AT). That means that during inference each prediction of a token in the sequence depends on the previous tokens starting with a $<$ SOS $>$ (start of the sequence) token until an $<$ EOS $>$ (End of sequence) token is predicted.[7] (See figure (4)). This method is called in literature left to right beam search [9]

Although AT delivers good results, in some applications it is necessary to speed up the inference procedure. Due to the autoregressive procedure of inference, it is not possible to parallelize the inference to speed it up. This has led in recent years to the rise of the Non-autoregressive transformers (NAT). We can see in (1.3) a comparison between the different sequence to sequence models and how they compare in terms of parallelization. The main idea behind NAT is that the system predicts the whole sequence within a constant number of iterations which does not depend on output sequence length. The way to do this is to introduce a conditional masked language model which are encoder-decoder architectures trained with a masked model objective [11] [12]. During training, some random tokens are replaced by a special mask token, and the network is trained to predict the original token so the system makes predictions conditioned on input speech, and during inference, the network decoder can condition on any subset to predict the rest given an input speech[7] [11]. This procedure and other NAT models in ASR will be the focus of this work.

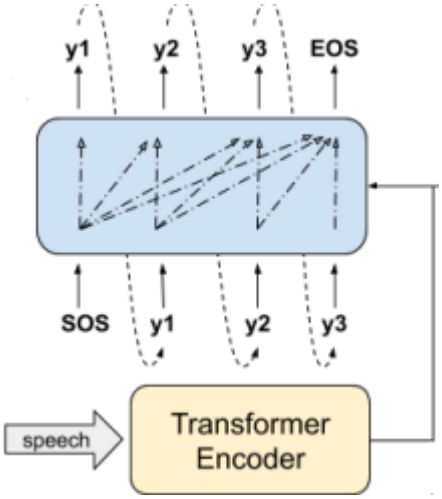

Figure 4: Inference procedure in AT ASR. When a prediction is made on input speech to predict the respective written output sequence, the predictions starts from < SOS >, and each token afterwards is predicted by conditioned on the previous tokens. [10]

Models	Training	Inference
AT RNN Based	x	x
AT CNNs based	✓	x
AT Self Attention based	✓	x
NAT	✓	✓

Table 1: Camparision between Autoregressive Models (AT) and Non-Autoregressive Models in terms of parallelization and which part could be computed in parallel [13]

1.4 Connectionist Temporal Classification

One main problem in ASR or any sequence to sequence modeling is to learn the alignment between input and output sequences [14].
Let us consider input speech sequences X as audio and output character sequences Y, the problem with alignment arises when we consider that $X = [x1, x2, ..x_T]$ and $Y = [y_1, y_2, .., y_N]$ might differ in length, the ratio of length between X and Y might differ (for example a long input speech signal might translate to fewer output sequences due to long silence in the speech or slowly spoken words), and the lack of accurate alignment in correspondence of the elements of X and Y [15]. This has led to the introduction of the Connectionist Temporal Classification (CTC) which overcomes these barriers.
The main idea behind CTC is that for any given X it produces an output distribution over all possible Y's, and we use this distribution to find the most probable alignment by summing over the probability of all possible alignments. To do so three things are required, first for each time step of the input speech sequence $X = [x1, x2, ..x_T]$ an output character is assigned, second a < BLANK > token is introduced which does not correspond to anything (it is not a space character, since space is a character on its own), third since an output character is assigned at each time step we collapse each repeating character to one in case they do not have a blank token in between them and remove the blank tokens from the output [15]. This procedure is explained in figure (5).

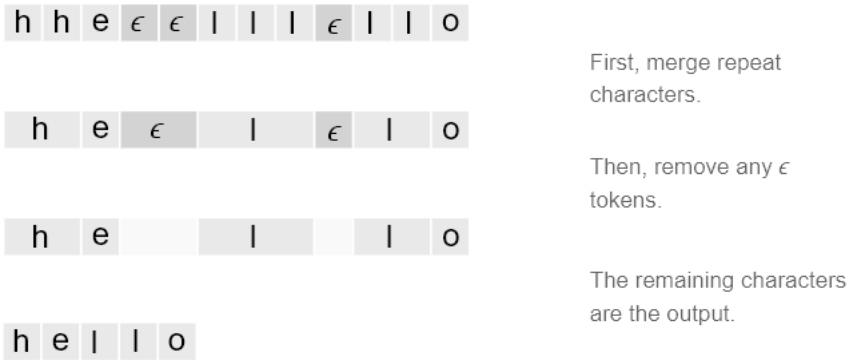

Figure 5: An example of CTC with the word Hello, the CTC approach has the advantage of avoiding the alignment problem, also introducing the blank tokens (here ϵ) allows for aligning repeated characters.

During training the network gives a probability distribution of different possible output alignments for each time step, then we compute the probabilities of different output sequences, and at the end, we sum over the aliments to get a distribution over the outputs. This procedure can be computed efficiently using dynamical programming [15] [14].

This work is organized as the following: In section 2 we will explain in detail the masked model approach and Audio-Conditional Masked Language Models to introduce the non-autoregressive method into the transformer network. In section 3 we will present another variant of the masked language model with the CTC loss, and we will study a case of the Mask predict mask CTC with spike trigger. In section 4 we will briefly present some other notable non-autoregressive ASR models. In section 5 we will present experiments, tests, and performance of the presented models and in the last section, we will conclude the results.

2 Non-Atuoregressive Automatic Speech Recognition

In the previous section we have introduced ASR, AT, and motivated the reasons to present NAT, now we want to present them in more rigorous way.
The point of end-to-end Automatic Speech Recognition (ASR) is to model the joint probability of a N-length output sequence $Y = \{y_n \in V | l = 1, ..., N\}$ given a T-length input sequence $X = \{\mathbf{x}_t \in \mathbb{R}^D | t = 1, ..., T\}$ Where y_n is the output token at position n in the vocabulary V and x_t is the D dimensional acoustic feature at a time step t [12]

2.1 Autoregressive Models

Attention-based encoder-decoder models the joint probability of Y given X by factorizing the probability based on the probabilistic chain rule:

$$P_{att}(Y|X) = \prod_{n=1}^{N} P_{att}(y_n|y_{<n}, X) = P(y_n|y_{<n}, f_t(h)) \tag{2}$$

The encoder takes speech features x_t as input and produces hidden representation h_t. The model estimates the output token y_n at each time step conditioning on previously generated tokens $y_{<n}$ and all hidden representation $\mathbf{h} = (\mathbf{h1}, \mathbf{h2}, ...)$. This also can be seen in figure (2), where the output in the decoder is conditioned on the history tokens in the decoder and hidden representation of the attention mechanism $f_t(h)$. $f_t(h)$ is a t dependent function on all hidden representations h. The attention function $f_t(h)$ which is usually considered as the weighted combination of all hidden representations:

$$f_t^{att}(\mathbf{h}) = \sum_{t'} w_{t,t'} \mathbf{h_{t'}} \tag{3}$$

With t' the enumeration of all possible hidden representations in $\mathbf{h}$ and the weight $w_{t,t'}$ is a trained parameter. [12] [7]

During training, the ground truth history tokens $y_{<n}$ are used as input to the decoder in equation (2), which lets us train the model in parallel for all the input sequences simultaneously [5], also this approach of using the ground history token has the advantage overusing the predictions makes the training easier and faster [7]. During inference, since no history tokens are used, the predicted tokens are used in an autoregressive manner, where each predicted token y_n is conditioned on the previous tokens $y_{<n}$, this has been explained in the introduction and can be seen in figure (4).

2.2 Non-Autoregressive ASR

To get around the problem of parallel computation in the inference stage NAT has been introduced, but to do so, we need to introduce some adjustments to the decoder in the training phase. The main idea is to replace the tokens $y_{<n}$ with some other information that are not dependent on other previous tokens so the computation could be performed in parallel. This idea originated from the context of Machine Translation in papers like [11],[16] and was extended to Automatic Speech Recognition by papers like [7] and [12]. In this section we will present the non-autoregressive methods presented in [7] and in the next section we will discuss other extensions and methods based on CTC presented in papers like [12] and [17]
One approach to introduce non-autoregressivness into the model is to remove $y_{<n}$ so

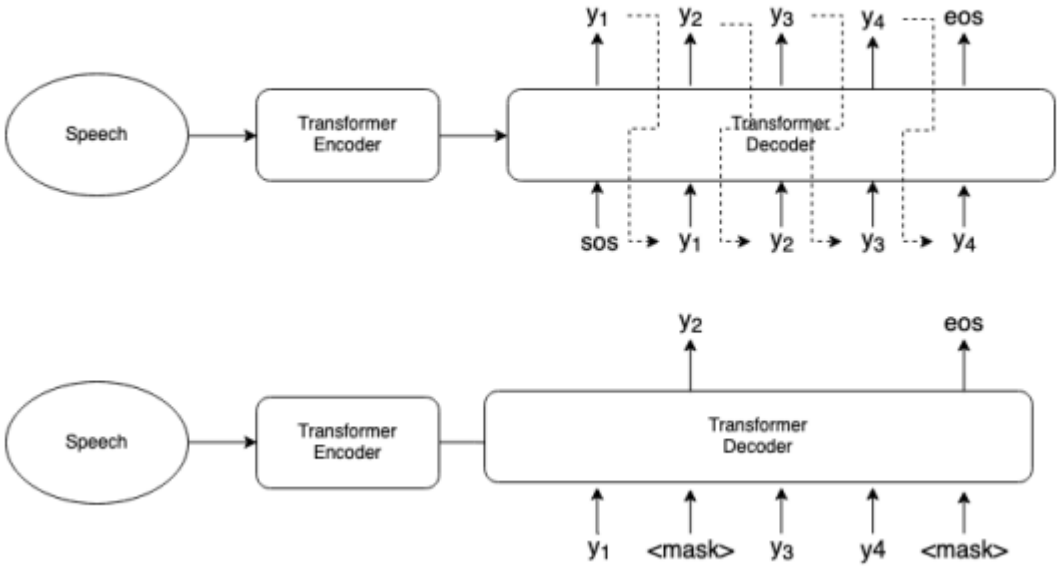

Figure 6: Comparison between Autoregressive transformer (upper figure) and Non-Autoregressive transformer (lower figure). The dashed lines in the upper figure represent the inference procedure, and it can be seen that each prediction is conditioned on the previous ones as in figure (4). The solid lines represent the training phase. In the NAT random ground truth history tokens are replaced with masked tokens and predictions in the training phase are made on them. In the AT only ground history tokens are used for training. We can also see that the changes between AT and NAT are only in the decoder, the encoder stays the same. [7]

parallel computation of $P(\mathbf{y}|\mathbf{h})$ can be factorized as the product of $P(\mathbf{y_n}|\mathbf{h})$, but this approach does not always deliver good results since the conditional independence might be too strong for ASR. Instead of that, Audio-Conditional Masked Language Model (A-CMLM) is introduced. [11] [7]

2.2.1 Audio-Conditional Masked Language Model

A-CMLM consists of introducing new masked tokens < MASK > into the decoding and training phases so the multi-iteration process can be implemented in the inference phase and therefore the possibility of parallelization. The goal of this procedure is to omit the conditioning on previous history tokens in the posterior as in equation (2) and replace it with conditioning on the masked tokens. [7].
Let N_M and N_U be the sets of masked and unmasked tokens so the posterior of the masked token is conditioned the unmasked tokens and the input speech becomes:

$$P(\mathbf{y}_{N_M}|\mathbf{y}_{N_U}, \mathbf{x}) = \prod_{n \in N_M} P_{dec}(\mathbf{y}_n|\mathbf{y}_{N_U}, f_t(\mathbf{h})) \tag{4}$$

Where $f_t(\mathbf{h})$ is the hidden representation (including the input speech and attention mechanism). During training, some random tokens (according to a uniform distribution) are

replaced by the special < MASK > token, and the network is tuned to predict the original unmasked tokens based on the input speech and attention mechanism [7].An important assumption here that allows the parallel computation is that give unmasked tokens, the predictions of masked tokens are conditionally independent so they can be estimated simultaneously computed as a product in equation (4) [7].

2.2.2 The Mismatch Problem and Audio-Factorized Masked Language Model

During training we can see from equation (4), that we needed to condition on the set of unmasked ground history token N_U to predict the masked tokens in the set N_M. However, during inference, those tokens are not provided. That means the model makes the prediction during inference without any context. This creates a mismatch between training and inference which can be large in long utterances. To reduce the effect of this possible mismatch, a method has been presented by [7] by reformulating the sets N_M and N_U and using the factorizations properties of conditional probabilities.
Let $Z_i \subset [0, 1, .., N-1]$ be a length L sequence of indices such that:

$$\begin{aligned} & Z_0 = \emptyset \\ & Z_L = [0, 1, ..., N-1] \\ \forall i \quad & Z_i \subset Z_{i+1} \end{aligned} \tag{5}$$

The posterior of training and inference becomes:

$$P(\mathbf{y}|\mathbf{h}) = \prod_{n=1}^{N} \prod_{Z_n \cap \overline{Z}_{n-1}} P_{dec}(\mathbf{y_i}|\mathbf{y}_{Z_{n-1}}, f_t\mathbf{h}) \tag{6}$$

Where $Z_n \cap \overline{Z}_{n-1}$ are the indices for decoding iteration n. This can be clearer with an example:

$$\begin{aligned} Z_0 &= \emptyset \\ Z_1 &= 0 \\ Z_2 &= 0 \\ Z_3 &= 0, 1, 2 \\ Z_4 &= 0, 1, 2, 3 \end{aligned} \tag{7}$$

Two special cases are important to notice here: It follows from the equations that $Z_n \cap \overline{Z}_{n-1} = n - 1$ and plugging this into equation (6) delivers equation (2). Also the AR model is a special case of the NAR model when $N = L$ and $Z_n = [0, 1, ..., n]$

During training, just like explained in the last section, random tokens are replaced with masked tokens in the decoder input. However, here all posteriors from the previous iteration $n - 1$ are sorted and the most confident one is chosen, this, in turn, is used to match with the inference case. The size of Z_n also has to be sampled from a uniform distribution between 0 and N. In the paper [7] it was suggested to set $N = 2$ which

would speed up the training and partially solve the problem of mismatch, we get now the following objective by solving for $N = 2$ in equation (6):

$$P(\mathbf{y}|\mathbf{h}) = \prod_{i \notin Z_1} P_{dec}(\mathbf{y}_i|\mathbf{y}_{t \in Z_1}, f_t(\mathbf{h})) * \prod_{j \in Z_1} P_{dec}(\mathbf{y}_j|f_t(\mathbf{h})) \tag{8}$$

Now if we compare the two equations (4) and (8), we can see that using the A-CMLM posterior accounts only for the first term, here when $Z1 = N_U$. In this case here in equation (8), we get an extra term that only concerns the probabilities of the unmasked tokens conditioned on the input speech, which help to "tune" the posterior and solves partially the problem of the mismatch. During inference, we still require further factorization in the decoding phase.

A pseudo code for the algorithm in this subsection was presented in the paper [7]. This method was called in the paper "Audio-Factorized Masked Language Model":

Algorithm 1: Minibatch forward pass

input : minibatch size n, dataset D, encoder network f_{enc}, decoder network f_{dec}
output : Posterior P
Sample $x = \mathbf{x}_1, ..., \mathbf{x}_n, y = \mathbf{y}_1, ..., \mathbf{y}_n$ from D;
$\mathbf{h} = f_{enc}(\mathbf{x})$;
Assign ⟨MASK⟩to all elements in $\hat{\mathbf{y}}^0$;
$P(\mathbf{y}^1|\mathbf{h}) = f_{dec}(\hat{\mathbf{y}}^0, \mathbf{h})$;
$mask = zeros(n, max\_length)$;
Assign ⟨MASK⟩to all elements in $\hat{\mathbf{y}}^1$;
for i=1,...,n **do**
  $probs = P_i(\mathbf{y}^1|\mathbf{h})$;
  $indices = argsort(probs.max(-1))$;
  $Z \sim Uniform(1, length(probs))$;
  $mask[i, indices[Z:]] = 1$;
  $\hat{\mathbf{y}}^1[i, indices[Z:]] = y[i, indices[Z:]]$;
end
$P(\mathbf{y}^2|\mathbf{h}) = f_{dec}(\hat{\mathbf{y}}^1, \mathbf{h})$;
$P = mask * P(\mathbf{y}^1|\mathbf{h}) + (1 - mask) * P(\mathbf{y}^2|\mathbf{h})$;

2.3 Decoding

In the last two subsections, we have discussed two training methods of Non-Autoregressive models. Now we want to discuss the decoding and how it can be done in parallel. In the next subsections, two different decoding strategies are going to be discussed of NAT differ from the left to right beam search that is usually applied in AT.

2.3.1 Easy First

This method was originally proposed by [18] and extended to the ASR context by [7]. The main idea is to predict the most obvious tokens and then gradually predict the less obvious ones within a predefined number of iterations.
In the first iteration, the decoder is fed masked tokens for all n (we will discuss the length of the sequence later.) Now after the first iteration we get decoding results $P(\mathbf{y}_n^1|.)$ we use the probabilities to keep the most confident ones and update them for the next iterations to make predictions on them:

$$\hat{\mathbf{y}}_n^1 = \begin{cases} \arg\max_V P(\mathbf{y}_n^1|.) & t \in \text{largest}_C(max_V P(\mathbf{y}^1|.)) \\ \hat{\mathbf{y}}_n^0 & \text{otherwise} \end{cases} \tag{9}$$

Where V is the vocabulary, $C = \frac{N}{K}$ is the largest number of predictions we keep, N is the sequence length, K is the fixed total number of iterations. This means we start with a sequence with a predefined length N filled with masked tokens, then within a predefined number of iterations, we make predictions. At each iteration, we replace masked tokens with the most confident predictions. This method erases the need for previous prediction to predict the next sequence, and therefore brings non-autoregressive processes into the picture and allows us to perform the computation in parallel.
In algorithm Algorithm 1 we used the ground truth tokens $y[i, indices[Z;]]$,but now we can substitute it with the predictions $\arg\max_V P(\mathbf{y}_n^1|.)$

2.3.2 Mask Predict

Another method for decoding with a fixed number of iterations was also suggested originally in the context of machine translation in [11] and extended to ASR in [7].
Similar to the last section we start with $\hat{\mathbf{y}}_n^0 =<$ MASK $>$ and in each iteration k, we check the the posterior probability for the most probable token at each output and use the probability as a confidence score to replace any less confident previous token. That means in the first iteration we replace masked tokens, but in the next iterations, if a previously predicted token is less confident than a currently predicted token, the previous token will be replaced with a masked token. (see figure 2). So we get for the k-th iteration:

$$\hat{\mathbf{y}}_n^k = \begin{cases} <\text{MASK}> & n \in \text{smallest}_C(\max_V P(\mathbf{y}_n^k|.)) \\ \arg\max_V P(\mathbf{y}_n^k|.) & \text{otherwise} \end{cases} \tag{10}$$

The number of masked tokens is $\lceil N * (1 - k/K) \rceil$
For example, for K=10 we mask 90% in the first iteration, 80% in the second iteration, and so on. In this method, we also need to update the confidence scores since masked tokens could re-emerge in the sequence, so the probabilities become:

$$P(\mathbf{y}_n^k|.) = \begin{cases} P(\mathbf{y}_n^k|.) & \hat{\mathbf{y}}_n^{k-1} =<\text{MASK}> \\ P(\mathbf{y}_n^{k-1}|.) & \text{otherwise} \end{cases} \tag{11}$$

2.3.3 Comparison and Example

Since easy fist takes the most confident predictions "for granted", it is considered a more conservative and common method. However, easy predict sometimes deliver better results since we can change previous predictions. For instance, if a prediction in easy first had false results in the first iteration, it could affect the whole sequence predictions afterward. Both methods are done in a fixed number of iterations which allows for parallelization, and allows us to get rid of the more traditional left to right search beam decoding. [7]

A clear example was presented in the paper of Chen 2021 [7] and an illustration from that paper can be found in figure (7).
In part (a) easy first is presented and part (b) shows mask predict. For both, we start with masked tokens in the first iteration, we have in total $K = 3$ iterations and a predefined sequence length $N = 5$. In easy first from the equation presented in the last section $\lceil N * (1 - k/K) \rceil = \lceil \frac{5}{3} \lceil = 2$, we get that 2 tokens need to be predicted in the first iteration, the the most confident tokens are "so" and "< EOS > tokens, so they are kept. Afterward, at each iteration, 2 tokens are predicted until the whole sequence is predicted.
A similar process is done with mask predict, in the first iteration the same two tokens are predicted "so" and "< EOS >. However, in the second iteration we get other tokens that have higher confidence that the token "so" and therefore it is replaced again by a masked token.

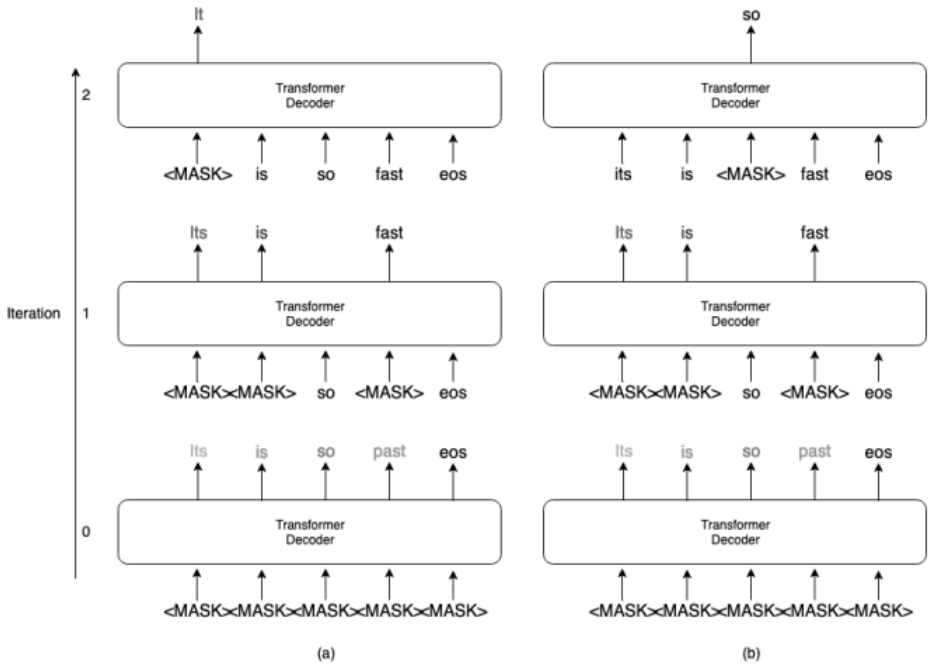

Figure 7: Illustration of the decoding example To predict the whole sequence with K = 3 iterations. Part (a) shows easy first process and part (b) shows mask predict. The example is explained in section (2.3.3). [7]

2.4 Sequence Length Problem

For both cases of easy fist and mask predict, we have predefined the sequence length N and required the prediction of $<$ EOS $>$ token. However, this method does not always deliver good results, especially if we are making the predictions with a language with a huge number of characters (for example using a Latin language corpus). [7] This is because that if we select a large sequence length redundant computations might occur and decoding speed will be reduced. And using a shorter sequence length might lead to the deletion of correctly predicted tokens. [17] Therefore, other methods are presented to solve this problem using the CTC loss presented in the introduction.

3 Connectionist Temporal Classification (CTC) with Mask Predict

Before we see results and experiments of the Mask Predict method in ASR that was presented in the last section, we are going to explore another variant of the mask predict with CTC.

3.1 Connectionist temporal classification

In this subsection we will give a short mathematical introduction of the CTC loss after a visual and conceptual introduction was given in section (1.4), more can be found here [14] and here [11].
As mentioned in the introduction, CTC loss predicts sequence alignment between input sequence X and output sequence Y by introducing special $<$ BLANK $>$ tokens, creating an output a possible alignment at each time step t, selecting the most probable output from the possible alignments, and then collapsing the predictions by ignoring all blank tokens and repeated letters that are not separated by a blank token.
The alignment $A = \{a_t \in V \cup \{<\text{BLANK}\} | t = 1, .., T\}$ is predicted with the conditional independence assumption between the output tokens as follows:

$$P_{ctc}(A|X) = \prod_{t=1}^{T} P_{ctc}(a_t|X) \tag{12}$$

Now if we sum over all possible alignments, CTC models the joint probability of Y given X as follows:

$$P_{ctc}(Y|X) = \sum_{A \in \beta^{-1}(Y)} P_{ctc}(A|X) \tag{13}$$

Where $\beta^{-1}(Y)$ returns all possible alignments compatible with Y. The summation of the probabilities for all of the alignments is usually computed using dynamic programming which ensures efficient and fast computations.
In the context of autoregressive ASR, the objective of the joint CTC-attention model is defined by combining equations (2) and (13) [12]:

$$L_{AR} = \lambda \log P_{ctc}(Y|X) + (1 - \lambda) \log P_{att}(Y|X) \tag{14}$$

Where λ is a tunable parameter.

3.2 Non Autoregressive ASR with joint CTC-CMLM

In section 2.2 we have presented the Audio-Conditional Masked Language Model. However, as we will see in upcoming chapters, when assessing its performance, it does not always deliver good results, and the autoregressive models can perform better. This is due to the problem of the sequence length. Therefore it was suggested in [12] to use CTC loss to jointly train the model using CTC loss and CMLM loss.

$$L_{NAR} = \lambda log P_{ctc}(Y|X) + (1 - \lambda) log P_{cmlm}(Y_{N_M}|Y_{N_U}, X) \tag{15}$$

3.3 Decoding Strategy

Using CTC omits the need of predefining the sequence length during inference. The main idea is to use the initialization of the target sequence and the CTC outputs as the initial sequence for decoding. This is done by using the output of the CTC, replacing the least confident outputs with masked tokens and then redo "easy first" decoding procedure. The CTC needs to be found using a greedy algorithm so it can be done in parallel, and a certain threshold is set to The CTC to know what percentage of the least confident tokens need to be masked (see figure (8)). [12].
All possible alignments of the predictions of the CTC are $\hat{Y} = \{\hat{y}_t \in \beta(A)|t = 1, ..., T'\}$, we take a greedy approach to find all possible alignments so we could later compute the decoding in parallel. The errors caused by the conditional independence assumption are expected to be corrected using CMLM decoder, see figure (8)
The posterior probability of $\hat{y}_t$ is approximately calculated using the frame-level CTC probabilities as follows:

$$\hat{P}(\hat{y}_t|X) = \max_j(P(a_j \in A_t|X) \tag{16}$$

Where $A_t = \{a_j\}_j$ is the consecutive same alignment that corresponds to the aggregated token $\hat{y}_n$. Then some output tokens are predicted (masked-out) depending on the confidence intervals as follows:

$$\hat{Y}_{N_M} = \{y_n \in \hat{Y}|\hat{P}(y_n|X) < P_{thres}\} \tag{17}$$

Where P_{thres} is a predefined probability threshold that determines whether a target token is masked or not.

Finally, another decoding method is just using the previously explained "easy first" method with CTC outputs and the decisions at each iteration are made according to equation (9).

3.4 Case Study: Spike Triggered NAT

An interesting work [17] introduces a new method using transformer with CTC loss and spike triggered (ST) mechanism to catch the blank tokens and speed up the performance of the model.

3.4.1 The Sequence Length Problem revisited

The CTC method discussed in last chapter solves the problem of the predefined sequence length. However, using CTC error might cause the model to generate duplicate tokens and a large number of blanks during inference that it slows down the prediction process which defies the original goal of the NAR approach.

Therefore [17] suggests a new model using the same previously explained CTC approach but with a probability threshold on the CTC outputs that helps predict the output. The main idea behind it is that we use the CTC to predict the length of the output sequence as it was explained in the previous chapter, but the CTC generates spike-like label posterior probabilities and the number of spikes reflects the length of the output sequence. The model is trained with a joint CTC loss and cross entropy loss. (see figure

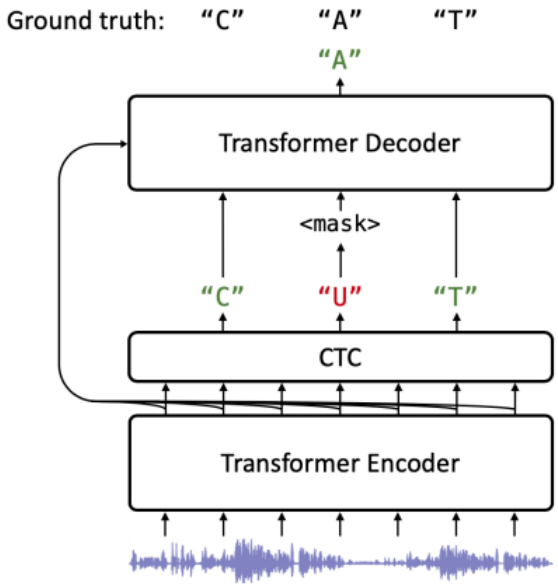

Figure 8: Overview of Mask CTC predicting "CAT" based on CTC outputs. The model is trained with the joint CTC and mask-predict objectives. During inference, the target sequence is initialized with the greedy CTC outputs and low-confidence tokens are masked based on the CTC probabilities. The masked low-confidence tokens are predicted by conditioning on the higher confidence tokens.

(9)) During inference, this model counts the number of spikes, which is an operation that requires usually little computational power. It does not depend on one of the masked tokens methods mentioned in the last chapter, but rather a beam search with a width of 5, this is a greedy algorithm that allows the model to be non-autoregressive [17]. The spike-triggered transformer adopts the encode states corresponding to the positions of spikes as the input of the decoder, and at the end the network uses the CTC loss as auxiliary loss which speeds up the training and convergence.

3.4.2 Training the spike-triggered Model

As mentioned, The ST-NAT can predict the length of the target sequence accurately, by counting the number of spikes produced by the CTC module. When the probability that the CTC generates a non-blank token is greater than some predefined threshold β, the corresponding trigger position is recorded:

$$POS(i) = \begin{cases} \text{triggered} & 1 - p_b \geq \beta \\ \text{ignored} & 1 - p_b < \beta \end{cases} \tag{18}$$

Where $POS(i)$ is the i-th position of the encoder output state, and p_b is the blank probability predicted by the CTC module.
The ST-NAT model inserts the end of sentence $<$ EOS $>$ token into the target sequence, to guarantee the model is still able to generate a correct sequence in case the predicted length T' is larger than the target length T. The model uses the CTC loss as an auxiliary

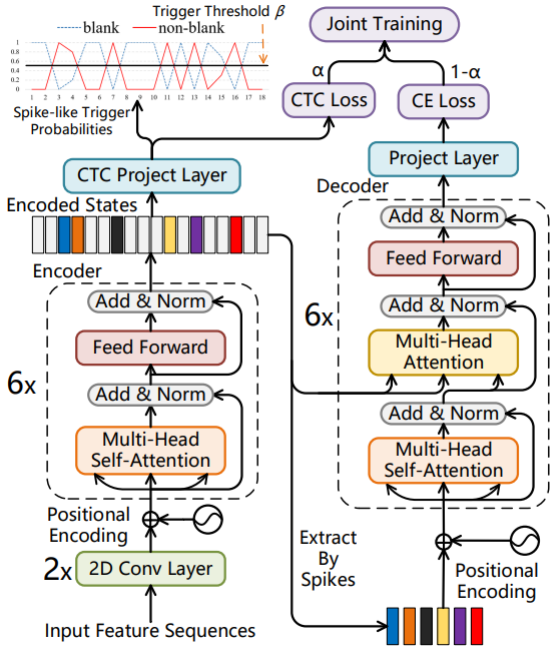

Figure 9: The spike-triggered non-autoregressive transformer consists of an encoder, a decoder, and a CTC module. The encoder processes the input feature sequences into encoded states sequence. The CTC computes spike like posteriors from encoded states. And then the decoder extract encoded states sequence from the corresponding position of spikes as the input. The whole system is trained jointly. [17]

loss function to optimize the model

$$L = \begin{cases} \alpha L_{CTC} + (1-\alpha) L_{CE} & T' \geq T \\ L_{CTC} & T' < T \end{cases} \tag{19}$$

Where L_{CE} is the cross-entropy loss function, L_{CTC} is the CTC loss and α is the weight of the joint loss function. As we can see in the equation, if T' the predicted target length is smaller than T the real target length the model only uses the CTC loss function does not need to be tuned with the cross-entropy to take into account the case that the real sequence length is larger than the predicted one. [17]

3.4.3 Inference of the ST-NAT

This model does not use the mask predict method with fixed length masked tokens as it was explained earlier, but rather uses the CTC spike and then utilizes the encoded states corresponding to the CTC spike as the input of the decoder by choosing token with the highest probability at each position. We also assume that the triggered encode state sequence contains some prior information of the target words, which makes the decoding process more purposeful than guessing from the empty sequence as it was explained in chapter 2 and figure (7). This lets us substitue the masked token method with a greedy

beam search with depth five, which allows us to do the computation in parallel.
NAT cannot model the temporal dependencies between the output labels which in general prevents the improvement of the output labels, which in general is a weakness to the performance of any NAT model. In the work [17] a transformer-based language model is also introduced to include a context in the decoding, and makeup partially to this weakness. However, the authors did not explicitly mention how they included non-atuoregressive inference for the language model. The joint decoding process can be described as:

$$\hat{y} = \arg\max_y(\log P(y|x) + \lambda \log P_{LM}(y|x)) \tag{20}$$

Where $\hat{y}$ is the predicted sequence and P_{LM} is the probability of the language model that we mentioned earlier and λ is the weight of the language model.

4 Other Models

Before starting presenting results and experiments of the three models that we have introduced in this work, we would like to present some other works that also deliver interesting approaches and interesting results that are worth mentioning, but we will only give introductions to them and avoid giving complete mathematical descriptions or presenting results of the experiments later.

4.1 Listen Attentively, and Spell Once

The "Listen Attentively, and Spell Once" (LASO) [19] model tries to deal with the problem of the language semantic and the dependencies between tokens that sometimes limits the performance of NAT models in speech recognition. The authors suggest a self-attention mechanism as the basic block to build three modules of LAS: encoder, the position dependent summarizer (PDS) , and the decoder.
The PDS summarizes the semantic at each position from the high level representation and bridges the sequence length gap between speech and token sequence, so the decoder can capture token-level semantic and then predict tokens.

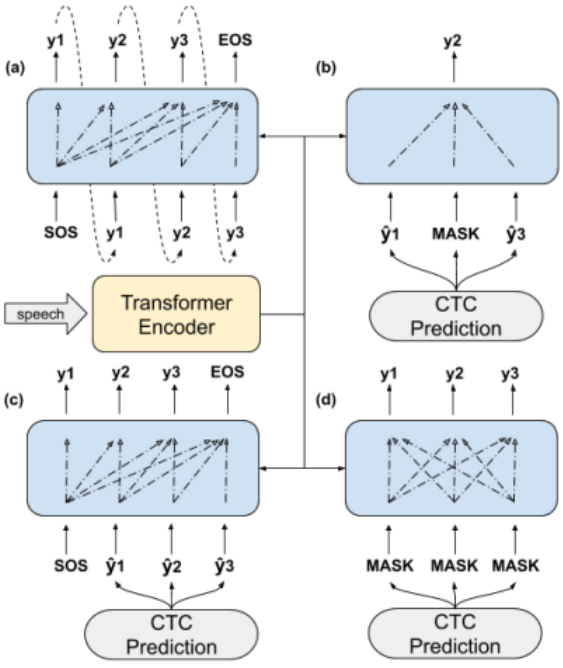

Figure 10: Comparison between AR and NAR transformer during inference. The blue boxes indicate Transformer Decoder and the dash-dotted lines inside them indicate the dependency between tokens. (a) AR Transformer. The dashed line limits the emission of yt by waiting for the end of generating yt1. (b) MP-CTC From section 3.1 (c) proposed NAR Transformer in section 4.2. (d) upgraded version of LASO Section 4.1. [10]

4.2 NAT transfomer with CTC-Enhanced Decoder Input in ASR

One drawback of NAT is that sometimes the predicted CTC tokens with high confidence are not correct and the masked tokens with low confidence are not always wrong, this

might lead to performance issues due to wrong predictions. Another drawback is that in general, the decoder is more sensitive to target side information rather than source side information. Therefore the authors in [10] aim to solve these issues by enhancing the decoder inputs of the model to reduce the difficulty of the task that the decoder needs to handle. They do so by feeding all the greedy CTC predictions to the decoder during inference. During training, the ground truth tokens are fed into the decoder as usual.

4.3 Insertion Based Modeling for End-to-End ASR

Another work that also tries to address the sequence length problem is [20] which suggests a method called insertion method inspired form applications in machine translation that utilizes joint modeling of CTC and insertion-based token sequence generation. The insertion method is an alternative to the mask-predict models that were presented in chapter 2. They marginalize over all different permutations of an input sequence and then the posterior probability is factorized with sum and product rules. In the decoding phase, the decoder utilizes the relative position representation at each step n and the final output factorizes a word prediction and positional prediction for the output probabilities.
In this work [20] the authors also present other decoding strategies for using the insertion-based model using CTC joint modeling output. In the experiments in the paper they receive results comparable to the autoregressive models.

5 Results and Experiments

In this section we will present the most relevant results and experiments of the models that were presented in sections 2 and 3 from the models: Mask Predict Models, CTC with Mask Predict and spike-triggered CTC. We will also compare them to the traditional autoregressive models.

5.1 Results of Listen and Fill in Missing letter form section 2

In the work of Listen and Fill in Missing Letter (Mask Predict) form chapter two [7], the authors have avoided using Latin Corpus since the words consist of a lot of letters and they don't deliver good results due to the sequence length pre definiton problem that we have mentioned multiple times before. Instead they use the Open Source Mandarin speech corpus (AISHELL) [21] and the Corpus of Spontaneous Japanese (CSJ) [22] because of the properties of the Japanese and Mandarin in the way of the composite writing of words and large number of characters that decreases the dependencies between them.

System	Dev CER	Test CER	RTF
Baseline Transformer (AR)	**6.0**	**6.7**	1.44
Easy first(K=1)	6.8	7.6	**0.22**
Easy first(K=3)	6.4	7.1	**0.22**
Mask-predict(K=1)	6.8	7.6	**0.22**
Mask-predict(K=3)	6.4	7.2	0.24
A-FMLM(K=1)	6.2	**6.7**	0.28
A-FMLM(K=2)	6.2	6.8	**0.22**

Table 2: Comparison between baseline Transformer, CMLM with Easy First and Mask Predict, and A-FMLM on AISHELL Corpus. CER is the Character Error Rate, K is the number of iterations, RTF is the real time factor. We can see that the AT outperforms the NAT models, however the other models still deliver reasonable performance with A-FMLM delivering the best results. All NAT deliver great speed up, up to 7 times. [7]

The authors use the network ESPnet for all experiments [23], for the non-autoregressive baseline a transformer system from [24] has been used. The authors use for the experiments encoder that includes 12 transformer blocks with convolutional layers at the beginning of downsampling. The decoder consists of 6 transformer blocks where each transformer 4 heads are used for attention.

We can see in the table (2) that the network on the AISHELL corpus with AT delivers the best performance. However, the NAT models deliver up to 7 times the speed up with a reasonable performance. As to the CSJ Corpus it delivers almost identical results in terms of speed up and performance. [7]

5.2 MASK-CTC results

In this section we will discuss the results of the proposed methods from section 3. Unlike the FMLM and CMLM methods, the Mask-CTC model actually can deliver reasonable performance on Latin alphabet, they do they experiments on the Wall Street Journal 81

hours corpus (WSJ) [25] and Italian Voxforge 16 hours corpus [26]. They do their experiments also on the CSJ corpus [22].

For the experimental setup the authors use the following architecture: Transformer self-attention layers with 4 attention heads, 256 hidden unites and 2048 feed-forward inner dimension size. The encoder consists of 12 self-attention layers with convolutional layers and for downsamplimg, the decoder has 6 self attention layers. With mask predict objective, the convergence for training the Mask-CTC model requires about 200-500 epochs while the autregressive model needs about 50-100 epochs. For the Mask-CTC model, they trained with 10-30 epochs and the P_{thres} from equation (17) they use a threshold value 0.999 for WSJ.

System	Iterations	dev93 WER	eval93 WER	RTF
AR CTC-attention+beam search	N	**13.5**	**10.9**	4.62
Mask CTC ($P_{thres} = 0$)	1	16.3	12.9	**0.03**
Mask CTC	1	15.7	12.5	0.04
Mask CTC	10	15.5	12.1	0.07
Mask CTC	#mask	15.4	12.1	0.13

Table 3: Word error rate (WERs) for WSJ on AT and NAT that were presented in sections 3.1-3.3 from the paper [12]. P_{thres}=0 means probability threshold form equation (17) is zero and the decoder output takes the CTC output without masking process, this models delivers highest speed up, up to 154 times. For all other NAT experiments $P_{thres} = 0.999$. #mask means that decoding takes as many iterations as there are masked tokens, it also delivers best results for all NAT models, it delivers also up to 13 times speed up from AT.

In [12] experiments are done on multiple corpus. However here we are going to concentrate only on the case of WSJ corpus, as they show almost identical behavior to the WSJ experiments.
In Table (3) is the WER of the WSJ corpus with NAT and AT. The AT model is based CTC loss and beam search model from [27] and it delivers the best performance. However, the NAT mask CTC approach with CTC outputs without masking (P_{thres}=0 from equation (17)) delivers a 156 times speed up from the NAT model. The mask CTC with iterations equal to the masked tokens delivers the best performance between the NAT with a 35 times speedup than the AT.

5.3 Spike Triggered NAT experiments

In this section we will present the results of the ST-NAT from chapter 3.4. The authors of the paper[17] only use the Japanese AISHELL corpus. The experiments were done on various spike threshold values as in equation (18) and various values α for the weighted CTC as in equation (19). However, the best values were $\alpha = 0.6$ and $\beta = 0.3$ and the models shown here are trained with these values.

The experimental setup for the model is as the following: the model consists of 6 encoder blocks and 6 decoder blocks. There are 4 heads in multi head attention. The output size of the multi-head attention and the feed-forward layers are 320. The model is trained for 80 epochs and for the last 20 epochs the parameters are averaged.

System	Dev CER	Test CER	RTF
Speech Transformer AR	6.57	7.37	0.0504
NAT-masked	7.16	8.03	0.0058
ST-NAT	6.88	7.67	**0.0056**
ST-NAT+LM	**6.39**	**7.02**	0.0292

Table 4: Character error rate (CER) of AISHELL corpus with the ST-NAT model from section 3.4 with and without a language model (LM) compared to the masked language model from chapter 2 [7], and AT model from [28]. The ST model deliver up to 10 times the speed up of the AT model with a comparable performance, it also outperforms the masked model performance. The ST model with language model delivers the best performance.

In table (4) we present the experiment results on the AISHELL corpus.The ST model deliver the highest speed up even in comparison with the masked language model from chapter 2. The ST model with language model delivers the best performance even in comparison with the AT models tested in the paper [17].

6 Conclusion

In this work, we have presented the various non-autoregressive models in the context of automatic speech recognition. There are several proposed methods to introduce this NAT approach into transformers with the aim to speed up the inference process of ASR. Audio-Conditional Masked Language model, Audio-Factorized Masked Language model [7], CTC Mask Predict model[12] and Spike Triggered CTC [17] model have all been presented, they all present the NAR approach from different angles.
In chapter 2 A-CMLM and A-FMLM introduce non-autoregressive models into ASR by using masked tokens in the decoder to train the model and predict the output sequence within a fixed number of steps. The models however do not always deliver good results with Latin corpus and there is the problem of predefining the output sequence length. The model performs reasonably on Japanese and Chinese corpus and delivers about 30 times the speed up from AT models.
In chapter 3 a method to jointly use CTC loss with A-CMLM was presented. The Mask-Predict CTC approach deals better with the problem of sequence length by utilizing CTC loss and jointly training and decoding the model with A-CMLM. This approach was also able to train and infer the model on the English language WSJ corpus and deliver a speedup up to 156 times.
In section 3.4 we have presented a model with CTC loss and spike-triggered mechanism to set a threshold to capture blank tokens. This is trained jointly with CTC with spike-triggered output into the decoder and a cross-entropy loss for the decoder. The model delivers better performance and speedup than the models presented in chapter 2.
In general, we can say that the non-autoregressive models bring a great speedup but introducing the ability of parallelization in the decoding. However, their perform less that autoregressive models.

References

[1] Jinyu Li, Li Deng, Reinhold Haeb-Umbach, and Yifan Gong. Chapter 1 - introduction. In Jinyu Li, Li Deng, Reinhold Haeb-Umbach, and Yifan Gong, editors, *Robust Automatic Speech Recognition*, pages 1–7. Academic Press, Oxford, 2016.

[2] Poodar Chu. Speech recognition with lstm in tensorflow, 2017.

[3] M I Jordan. Serial order: a parallel distributed processing approach. technical report, june 1985-march 1986.

[4] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 11 1997.

[5] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

[6] George Lawton. Transformer neural networks are shaking up ai. `https://www.techtarget.com/searchenterpriseai/feature/Transformer-neural-networks-are-shaking-up-AI`, 2021.

[7] Nanxin Chen, Shinji Watanabe, Jesus Villalba, Piotr Zelasko, and Najim Dehak. Non-autoregressive transformer for speech recognition. *IEEE Signal Processing Letters*, 28:121–125, 2021.

[8] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Huggingface's transformers: State-of-the-art natural language processing, 2020.

[9] Markus Freitag and Yaser Al-Onaizan. Beam search strategies for neural machine translation. *Proceedings of the First Workshop on Neural Machine Translation*, 2017.

[10] Xingchen Song, Zhiyong Wu, Yiheng Huang, Chao Weng, Dan Su, and Helen Meng. Non-autoregressive transformer asr with ctc-enhanced decoder input, 2021.

[11] Marjan Ghazvininejad, Omer Levy, Yinhan Liu, and Luke Zettlemoyer. Mask-predict: Parallel decoding of conditional masked language models, 2019.

[12] Yosuke Higuchi, Shinji Watanabe, Nanxin Chen, Tetsuji Ogawa, and Tetsunori Kobayashi. Mask ctc: Non-autoregressive end-to-end asr with ctc and mask predict, 2020.

[13] Junliang Guo, Xu Tan, Di He, Tao Qin, Linli Xu, and Tie-Yan Liu. Non-autoregressive neural machine translation with enhanced decoder input. *CoRR*, abs/1812.09664, 2018.

[14] 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*, ICML '06, page 369–376. Association for Computing Machinery, 2006.

[15] Awni Hannun. Sequence modeling with ctc. `https://distill.pub/2017/ctc/`, 2017.

[16] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

[17] Zhengkun Tian, Jiangyan Yi, Jianhua Tao, Ye Bai, Shuai Zhang, and Zhengqi Wen. Spike-triggered non-autoregressive transformer for end-to-end speech recognition, 2020.

[18] Yoav Goldberg and Michael Elhadad. An efficient algorithm for easy-first non-directional dependency parsing. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 742–750, Los Angeles, California, June 2010. Association for Computational Linguistics.

[19] Ye Bai, Jiangyan Yi, Jianhua Tao, Zhengkun Tian, Zhengqi Wen, and Shuai Zhang. Listen attentively, and spell once: Whole sentence generation via a non-autoregressive architecture for low-latency speech recognition, 2020.

[20] Yuya Fujita, Shinji Watanabe, Motoi Omachi, and Xuankai Chan. Insertion-based modeling for end-to-end automatic speech recognition, 2020.

[21] Hui Bu, Jiayu Du, Xingyu Na, Bengu Wu, and Hao Zheng. Aishell-1: An open-source mandarin speech corpus and a speech recognition baseline, 2017.

[22] Kikuo Maekawa. Corpus of spontaneous japanese : Its design and evaluation. 2003.

[23] Shinji Watanabe, Takaaki Hori, Shigeki Karita, Tomoki Hayashi, Jiro Nishitoba, Yuya Unno, Nelson Enrique Yalta Soplin, Jahn Heymann, Matthew Wiesner, Nanxin Chen, Adithya Renduchintala, and Tsubasa Ochiai. Espnet: End-to-end speech processing toolkit, 2018.

[24] Shigeki Karita, Nanxin Chen, Tomoki Hayashi, Takaaki Hori, Hirofumi Inaguma, Ziyan Jiang, Masao Someki, Nelson Enrique Yalta Soplin, Ryuichi Yamamoto, Xiaofei Wang, Shinji Watanabe, Takenori Yoshimura, and Wangyou Zhang. A comparative study on transformer vs RNN in speech applications. *CoRR*, abs/1909.06317, 2019.

[25] Douglas B. Paul and Janet M. Baker. The design for the Wall Street Journal-based CSR corpus. In *Speech and Natural Language: Proceedings of a Workshop Held at Harriman, New York, February 23-26, 1992*, 1992.

[26] Voxforge. `"http://www.voxforge.org"`.

[27] Suyoun Kim, Takaaki Hori, and Shinji Watanabe. Joint ctc-attention based end-to-end speech recognition using multi-task learning, 2016.

[28] Linhao Dong, Shuang Xu, and Bo Xu. Speech-transformer: A no-recurrence sequence-to-sequence model for speech recognition. *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5884–5888, 2018.