A. Verification of Eq. (3)

In this section, we verify that if the alignment matrix $\boldsymbol{\alpha}$ meets the continuity and monotonicity criteria, then we have

$$0 \leq \Delta(\pi) \leq 1. \tag{1}$$

where, $\Delta\pi_i = \pi_i - \pi_{i-1}$, for $1 \leq i \leq T_2 - 1$.

We denote $P(x_i, y_j)$ the probability that the decoder frame y_j is attended on the text token x_i. At each output timestep, an alignment pair (x_{i-1}, y_{j-1}) either move forward by one step to (x_i, y_j) or stay unmoved (x_{i-1}, y_j). Then we have:

$$P(x_i, y_j|x_{i-1}, y_{j-1}) + P(x_{i-1}, y_j|x_{i-1}, y_{j-1}) = 1, \tag{2}$$

where $P(x_i, y_j|x_{i-1}, y_{j-1})$ is the conditional probability that (x_{i-1}, y_{j-1}) move forward to (x_i, y_j) given y_{j-1} is attended on x_{i-1}. For convenience we define:

$$\beta_{i,j} = P(x_i, y_j|x_i, y_{j-1}), \quad \alpha_{i,j} = P(x_i, y_j).$$

For each timestep (x_i, y_j), the previous alignment pair is either (x_i, y_{j-1}) or (x_{i-1}, y_{j-1}). thus we have:

$$\begin{aligned} P(x_i, y_j) &= P(x_i, y_j|x_i, y_{j-1}) * P(x_i, y_{j-1}) + P(x_i, y_j|x_{i-1}, y_{j-1}) * P(x_{i-1}, y_{j-1}) \\ \alpha_{i,j} &= P(x_i, y_j|x_i, y_{j-1}) * P(x_i, y_{j-1}) + (1 - P(x_{i-1}, y_j)|x_{i-1}, y_{j-1})) * P(x_{i-1}, y_{j-1}) \\ &= \beta_{i,j} * \alpha_{i,j-1} + (1 - \beta_{i-1,j}) * \alpha_{i-1,j-1}. \end{aligned} \tag{3}$$

According to the definition of IMV, we have:

$$\begin{aligned} \pi_j &= \sum_{i=0}^{T_1-1} \alpha_{i,j} * p_i = \sum_{i=0}^{T_1-1} \alpha_{i,j} * i \\ &= \sum_{i=1}^{T_1-1} \alpha_{i,j} * i + \alpha_{0,j} * 0 \\ &= \sum_{i=0}^{T_1-1} \beta_{i,j} * \alpha_{i,j-1} * i + \sum_{i=1}^{T_1-1} (1 - \beta_{i-1,j}) * \alpha_{i-1,j-1} * i \\ &= \sum_{i=0}^{T_1-1} \beta_{i,j} * \alpha_{i,j-1} * i + \sum_{i=1}^{T_1-1} (1 - \beta_{i-1,j}) * \alpha_{i-1,j-1} * (i - 1 + 1) \\ &= \sum_{i=0}^{T_1-1} \beta_{i,j} * \alpha_{i,j-1} * i + \sum_{i=1}^{T_1-1} (1 - \beta_{i-1,j}) * \alpha_{i-1,j-1} * (i - 1) + \sum_{i=1}^{T_1-1} (1 - \beta_{i-1,j}) * \alpha_{i-1,j-1} \\ &= \sum_{i=0}^{T_1-1} \beta_{i,j} * \alpha_{i,j-1} * i + \sum_{i=1}^{T_1-1} \alpha_{i-1,j-1} * (i - 1) - \sum_{i=1}^{T_1-1} \beta_{i-1,j} * \alpha_{i-1,j-1} * (i - 1) + \sum_{i=1}^{T_1-1} (1 - \beta_{i-1,j}) * \alpha_{i-1,j-1} \\ &= \sum_{i=0}^{T_1-1} \beta_{i,j} * \alpha_{i,j-1} * i + \sum_{i=0}^{T_1-2} \alpha_{i,j-1} * i - \sum_{i=0}^{T_1-2} \beta_{i,j} * \alpha_{i,j-1} * i + \sum_{i=1}^{T_1-1} (1 - \beta_{i-1,j}) * \alpha_{i-1,j-1} \\ &= \beta_{T_1-1,j} * \alpha_{T_1-1,j-1} * (T_1 - 1) + \pi_{j-1} - \alpha_{T_1-1,j-1} * (T_1 - 1) + \sum_{i=1}^{T_1-1} (1 - \beta_{i-1,j}) * \alpha_{i-1,j-1} \\ &= \pi_{j-1} + (\beta_{T_1-1,j} - 1) * \alpha_{T_1-1,j-1} * (T_1 - 1) + \sum_{i=1}^{T_1-1} (1 - \beta_{i-1,j}) * \alpha_{i-1,j-1} \end{aligned} \tag{4}$$

$T_1 - 1$ is the maximum input location, once an alignment pair reaches to maximum location (x_{T_1-1}, y_j), the next alignment pair must be (x_{T_1-1}, y_{j+1}). Thus for all $j > 1$, $\beta_{T_1-1,j} = P(x_{T_1-1}, y_j|x_{T_1-1}, y_{j-1}) = 1$. Therefore, above equation can

be written as:

$$\pi_j = \pi_{j-1} + \sum_{i=1}^{T_1-1} (1 - \beta_{i-1,j}) * \alpha_{i-1,j-1} \tag{5}$$

Because:

$$0 \leq \sum_{i=1}^{T_1-1} (1 - \beta_{i-1,j}) * \alpha_{i-1,j-1} \leq \sum_{i=1}^{T_1-1} \alpha_{i-1,j-1} \leq \sum_{i=1}^{T_1} \alpha_{i-1,j-1} = 1. \tag{6}$$

Then we get:

$$0 \leq \Delta\pi_j \leq 1, \quad \forall j \in [0, T_2 - 1]. \tag{7}$$

B. Implementation details of EfficientTTS

B.1. Implementation details of EfficientTTS families

EFTS-CNN. EFTS-CNN consists of 6 convolutions. Each convolution is interspersed with weight normalization, Leaky ReLU activation, and residual connection. We add a linear projection at the end to generate melspectrogram. The overall training objective of EFTS-CNN is a combination of aligned position loss and MSE loss of melspectrogram. We set the coefficients to 1 in our experiments.

EFTS-Flow. EFTS-Flow consists of 8 flow steps, and each flow step consists of two elemental invertible transformations: an invertible linear layer and an affine coupling layer. To reduce the cost of computing the determinant of Jacobian matrix for invertible linear layer(the cost is $\mathcal{O}(c^3)$, where c is the channels of permutation, in our case, c is the dimensionality of mel-spectrogram 80), we rearrange 80 channels into 8 groups, and perform permutation between 8 groups. To speed up training, we train the flow decoder together with the CNN decoder which is able to learn accurate alignment within 200k training steps. The overall training objective of EFTS-Flow is a combination of aligned position loss, maximum likelihood estimation (MLE) loss, and MSE loss of melspectrogram. For simplicity, we set all coefficients to 1 in our experiments.

EFTS-Wav. The dilated convolutional adversarial decoder is similar to MelGAN except: (1) The input of the generator is high dimensional hidden representations not 80-channel melspectrograms; (2) A multi-resolution STFT loss is incorporated at end of the generator. We adopt the same structure of MelGAN discriminator for adversarial training. Similar as ClariNet and EATS, we train MelGAN part by conditioning on sliced input corresponding to 0.5s audio clips, while other part is trained on the whole-length utterance. We jointly train the CNN decoder with MSE loss of melspectrogram, which allows EFTS-Wav to learn the whole-length alignment for each training step. In our experiments, we find that it's more efficient to pretrain the CNN decoder and dilated convolutional adversarial decoder using the alignment reconstructed from $\boldsymbol{\pi}$ (following Eq. (11)), and then fine-tune the whole model using alignment reconstructed from $\boldsymbol{e}$. The overall training objective of EFTS-Wav generator is a linear combination of reconstruction loss of melspectrogram, MelGAN generator loss, multi-resolution STFT loss, and aligned position loss. The coefficients of the losses are 2, 1, 10, 1 respectively.

EFTS-SMA. To evaluate the effectness of proposed monotonic approach, we also implement a EfficientTTS model called EFTS-SMA. EFTS-SMA shares the same network structure with EFTS-CNN except the IMV generator. In the IMV generator of EFTS-SMA, we compute IMV according to Eq. (2) directly. We train EFTS-SMA with a SMA loss, the coefficients of each items in SMA loss are $\lambda_0 = 5, \lambda_1 = 5, \lambda_2 = 1, \lambda_3 = 1$. The overall training objective of EFTS-SMA is a combination of aligned position loss and MSE loss of melspectrogram and SMA loss, the coefficients are 1, 1, 20 respectively.

B.2. Hyperparameters

The hyperparameters of proposed models are shown in Table 1.

B.3. EfficientTTS pseudocode

In Fig. 1 we present the pseudocode of EfficientTTS modules, including generating IMV from alignment matrix, extracting aligned positions from IMV, and reconstructing alignment matrix from aligned positions.

Table 1. Hyperparameters of EfficientTTS models

Modules	EFTS-CNN	EFTS-Flow	EFTS-Wav
Text-Encoder	EmbeddingDimension = 512, FFTLayers = 4, HiddenDimension = 512, AttentionHeads = 2,		
Mel-Encoder	Conv1DLayers = 4, kernelSize = 5, Dilation = [1,2,2,3], FilterSize = 512,		
IMV generator	-		
Aligned position predictor	$\sigma^{-2} = 0.5$ (in Eq. (17)), Conv1DLayers = 3, kernelSize =[3,3,1], FilterSize = [128, 32, 1]		
Alignment reconstruction	$\sigma^{-2} = 0.2$ (in Eq. (20)),		
Decoder	Conv1DLayers = 6 kernelSize = 5 Dilation = [1, 2, 2, 2, 1, 1] FilterSize = 512	FlowSteps = 8, InvertibleLinearChannels = 8, AffineCouplingkernelSize = 5, AffineCouplingFilterSize = 512, AffineCouplingConv1dLayers = 2, Dilation = [1,1]	UpsamplingScales = [8, 8, 2, 2], kernelSize = 7

B.4. Training details

Dataset preprocess. For both the DataBaker dataset and LJ-Speech dataset, we use phoneme sequences for training. Specifically, for DataBaker dataset, we first convert Chinese characters into pinyin sequences and further convert pinyin sequences to phoneme sequences according to fixed rules. While for LJ-Speech dataset, we use phonemizer [1] to convert text sequences to phoneme sequences. We pad a special silence token at the beginning and end of each phoneme sequence to get a more accurate alignment. We generate the 80-channel mel-spectrograms from 22.05kHZ waveforms according to an open source implementation [2]. The FFT size is set to 1024, hop length is 256, and window size is 1024. In our experiments, we find the robustness of non-autoregressive models are sensitive to the lengths of audio clips of training data. Therefore, we perform audio augmentation through random concatenating several audio clips to increase the average length of training audio clips.

Training settings. EfficientTTS is memory efficient, thus allows us training with large batch size. We train all models on a single Tesla V100 GPU. For EFTS-CNN and EFTS-Flow, we use the Adam optimizer with a batch size of 96 and a learning rate 1×10^{-3}, $\beta_1 = 0.9$ and $\beta_2 = 0.97$. It takes 60k steps for training EFTS-CNN on DataBaker dataset until converge, and 250k training steps for EFTS-Flow. For EFTS-Wav, we pretrain the CNN decoder and dilated convolutional adversarial decoder for 100k steps, and fine-tune the whole model for 560k training steps until converge. we use Adam optimizer with a batch size of 48. We use a learning rate of 1×10^{-4} for the generator and 5×10^{-5} for the discriminator.

C. Analysis of Melspectrograms

Speech rate. EfficientTTS is able to produce high quality speech with different speech rate by multiplying the aligned position with a positive scalar. Fig. 2 shows melspectrograms with different speech rate of the same utterance. The scalar factors are $1.2, 1.0, 0.8$ respectively. As can be seen, the generated melspectrograms are similar with each other, which means proposed models are able to speed up or slow down generated speech without changing the pitch and degrading the speech quality. We also attach several audio samples in our demo page.

Speech variation EFTS-Flow is a flow-based model, which is able to produce speech in diversity by inputting with different latent variable z. Fig. 3 shows the different melspectrograms by changing the temperature of z. Our results show that EFTS-Flow can produce high quality speech with different variations.

[1] https://github.com/bootphon/phonemizer
[2] https://github.com/r9y9/wavenet_vocoder

Algorithm 1 Tacotron2 with HMA

Initialize:
$\pi = 0$
$p = range(0, T_1)$
for $i = 0$ **to** $T_2 - 1$ **do**
$\alpha_i = SoftAttention(x_i, h)$
$\pi' = \alpha_i \cdot p$
$\Delta\pi = Clamp(\pi' - \pi, min = 0, max = 1)$
$\pi = \pi + \Delta\pi$
$\alpha_i = softmax(-\sigma^{-2} * (\pi - p)^2)$
$out = DecoderRNN(\alpha_i, h)$
end for

D. Experiments on Tacotron 2

D.1. Implementation of Tacotron2-SMA and Tacotron2-HMA

We implement Tacotron2-SMA by adding a SMA loss during training. The detailed implementation of Tacotron2-HMA is shown in Alg. 1. For each output timestep, we compute IMV π following Eq. (2). A $Clamp(\cdot)$ operation is used to limit $0 \leq \Delta\pi \leq 1$. We reconstruct the alignment α according to Eq. (11), and use the newly generated α for further computation.

D.2. Experiments results

We visualized the alignment matrix of different models in Fig. 4. As can be seen, the alignment matrix of Tacotron 2 is noisy, which often leads to mispronunciations, in contrast, both Tacotron2-SMA and Tacotron2-HMA learn a clean and smooth alignment.

```python
def generate_index_vector(text_mask):
    """Create index vector of text squence.
    Args:
        text_mask: text_mask: mask of text-squence. [B,T1]
    returns:
         index vector of text squence. [B,T1]
    """
    p = torch.arange(0,text_mask.size(-1)).repeat(text_mask.size(0),1).cuda().float()
    return p * text_mask

def imv_generator(alpha, p, mel_mask, text_length, use_hma=True, bi_direction_cum=True):
    """Compute imv from alignment matrix alpha.
    Args:
      alpha:  scaled dot attention [B,T1,T2]
      p: index vector, output of generate_index_vector. [B,T1]
      mel_mask: mask of mel-spectgram [B, T2]
      text_length: lengths of input text-squence [B]
      use_hma: whether to use HMA, default True
      bi_direction_cum: whether to use bi-directional cumulative sum operation, default True.
    returns:
      Index mapping vector (IMV) [B,T2]
    """
    imv_dummy = torch.bmm(alpha.transpose(1,2), p.unsqueeze(-1)).squeeze(-1)  #[B,T2]
    if not use_hma:
        return imv_dummy
    delta_imv = torch.relu(imv_dummy[:,1:] - imv_dummy[:,:-1]) # [B, T2-1]
    delta_imv = torch.cat([torch.zeros(alpha.size(0),1).type_as(alpha),delta_imv],-1)* mel_mask.float() # [B, T2-1] -> [B, T2]
    imv_f = torch.cumsum(delta_imv,-1)
    if bi_direction_cum :
        imv_b = torch.flip(torch.cumsum(torch.flip(delta_imv, [-1]),-1), [-1])
        imv = imv_f - imv_b
    else:
        imv = imv_f
    last_imv = torch.clamp(imv[:,-1].unsqueeze(1),min=1e-8)  # get last element of imv
    first_imv = imv[:,0].unsqueeze(1) # get first element of imv
    imv = (imv- first_imv) / (last_imv - first_imv) * (text_length.float().unsqueeze(1) -1)
    return imv * mel_mask.float()


def get_aligned_positions(imv, p, mel_mask, text_mask, sigma=0.5):
    """ compute aligned positions from imv
    Args:
        imv: index mapping vector #[B,T2]
        p: index vector, output of generate_index_vector. [B,T1]
        sigma: a scalar, default 0.5
        mel_mask: mask of mel-spectgram [B, T2]
    returns:
        Aligned positions [B,T1]
    """
    energies = -1 * ((imv.unsqueeze(1) - p.unsqueeze(-1))**2) * sigma
    energies = energies.masked_fill(~(mel_mask.unsqueeze(1).repeat(1,energies.size(1),1)),-float('inf'))
    beta =  torch.softmax(energies,dim=2)
    q = torch.arange(0,mel_mask.size(-1)).unsqueeze(0).repeat(imv.size(0),1).cuda().float()
    q = q * mel_mask.float() # generate index vector of target sequence.
    return torch.bmm(beta,q.unsqueeze(-1))*text_mask.unsqueeze(-1)


def reconstruct_align_from_aligned_postions(e, delta=0.1,mel_mask=None, text_mask=None):
    """ reconstruct alignment matrix from aligned positions
    Args:
        e: aligned positions [B,T1]
        delta: a scalar, default 0.1
        mel_mask: mask of mel-spectgram [B, T2], None if inference and B==1
        text_mask: mask of text-squence, None if B==1
    returns:
        alignment matrix [B,T1,T2]
    """
    if mel_mask is None:  # inference phase:
        max_length = torch.round(e[:,-1] + (e[:,-1] - e[:,-2]))
    else:
        max_length = mel_mask.size(-1)
    q = torch.arange(0,max_length).unsqueeze(0).repeat(e.size(0),1).cuda().float()
    if mel_mask is not None:
        q = q*mel_mask.float()
    energies = -1 * delta *(q.unsqueeze(1) - e.unsqueeze(-1))**2
    if text_mask is not None:
        energies = energies.masked_fill(~(text_mask.unsqueeze(-1).repeat(1,1,max_length)),-float('inf'))
    return torch.softmax(energies, dim=1)
```

Figure 1. EfficientTTS pseudocode

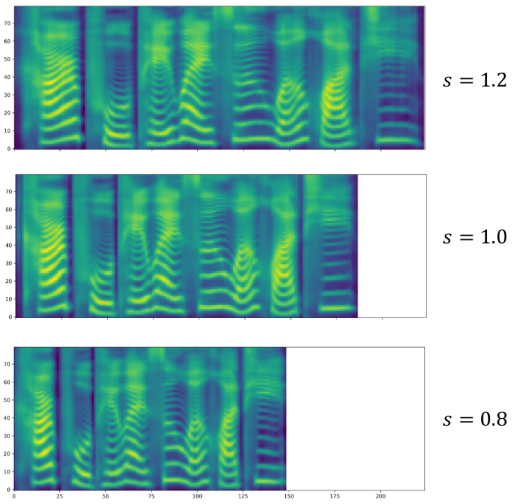

Figure 2. Melspectrograms with different speech rate by multiplying the aligned position by a positive scalar.

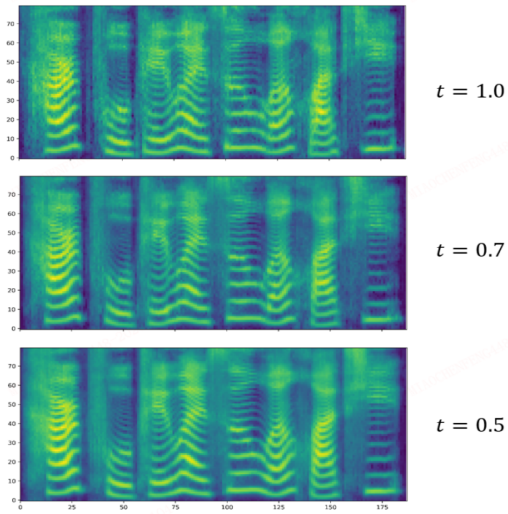

Figure 3. Melspectrograms with different speech variations generated by EFTS-Flow.

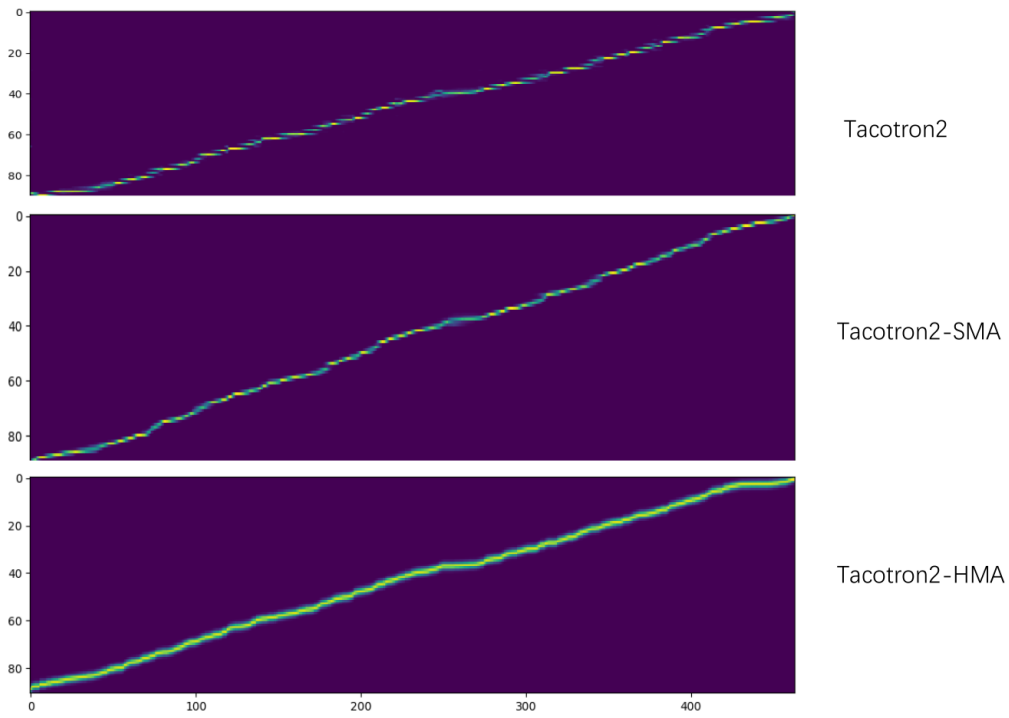

Figure 4. Alignments of Tacotron 2 with different settings of the same utterance.