RAD-TTS: Parallel Flow-Based TTS with Robust Alignment Learning and Diverse Synthesis

Anonymous Authors

Abstract

This work introduces a predominantly parallel, end-to-end TTS model based on normalizing flows. It extends prior parallel approaches by additionally modeling speech rhythm as a separate generative distribution to facilitate variable token duration during inference. We further propose a robust framework for the on-line extraction of speech-text alignments – a critical yet highly unstable learning problem in end-to-end TTS frameworks. Our experiments demonstrate that our proposed techniques yield improved alignment quality, better output diversity compared to controlled baselines.

1. Introduction

While speech synthesis is most naturally modeled sequentially in a fully autoregressive manner (Wang et al., 2017; Valle et al., 2020), training and inference speeds scale poorly with the sequence length. Furthermore, a single poorly predicted audio frame in a fully autoregressive model can catastrophically impact all subsequent inference steps, prohibiting its use for inferring long sequences. Recent works (Kim et al., 2020; Miao et al., 2020; Peng et al., 2020) have presented increasingly parallel solutions to address these concerns. They first determine the duration of each phoneme in the input text, then use a generative, parallel architecture to sample and decode each mel-spectrogram frame in parallel rather than sequentially. However, parallel architectures present challenges of their own. The following work proposes RAD-TTS: a TTS framework featuring robust alignment learning and diverse synthesis. We propose a stable and unsupervised alignment-learning framework applicable to virtually any TTS framework, as well as a generative phoneme-duration model to ensure diverse output in parallel TTS architectures.

Learning unsupervised audio-text alignment is difficult, especially in parallel architectures. Some works use off-the-shelf forced aligners (Ren et al., 2020), while other opt to distill the attention from an autoregressive model (or forced aligner) into a parallel architecture in an expensive two-stage process (Peng et al., 2020; Ren et al., 2019; Łańcucki, 2020), which makes the training even more expensive. Obtaining alignments from an external aligner has severe limitations, as it requires finding or training an external aligner for every language and alphabet and requires near-perfect text normalization. Most related to our work is Glow-TTS (Kim et al., 2020), which proposes an aligning mechanism in a normalizing flow framework. Our work extends their alignment setup to be more generally applicable to arbitrary TTS frameworks, as well as with improved stability.

We further tackle the limited synthesis diversity in parallel TTS architectures. These architectures first determine the durations of each phoneme. Then, a parallel architecture maps the phonemes, replicated time-wise based on their predicted durations, to corresponding Mel-spectrogram frames. Even if the latter parallel architecture is generative, the phoneme prediction is typically deterministic. This limits diversity in output as a lot of the variability is dependent on speech rhythm. Even flow-based ones such as Glow-TTS use deterministic regression models to infer the duration. As such, we propose to use a separate generative model only for token durations. Our results demonstrate more diverse inference results in comparison to a fixed-duration baseline.

In summary, our work proposes 1) a rapidly converging alignment learning framework incorporating the forward algorithm and priors, and 2) Generative duration modeling to ensure diverse samples in parallel TTS architectures.

2. Method

We aim to construct a generative model for sampling mel-spectrograms given text and speaker information. We extend prior bipartite-flow TTS methods (Kim et al., 2020; Miao et al., 2020) with a more robust alignment learning mechanism (2.2.1) and generative duration modeling (2.3). We formulate our setup as follows: consider an audio clip of human speech represented as a mel-spectrogram tensor...
Phoneme Flow

where $J$ is the Jacobian of the invertible transformation, and $\phi(x; \phi, \xi, \zeta)$ is the Gaussian likelihood function $\mathcal{N}(x; 0, 1)$. 

2.2. Mel Decoder Architecture

Our mel-decoder model, which we will continue to denote as $g$, models only $P_{\text{mel}}(\cdot)$ in Eq. 1, though a similar formulation is used to model $P_{\text{dur}}(\cdot)$ as we will show later on. The decoder allows us to sample latent vectors $z$ for each time step i.i.d. from a Gaussian prior, and map them to plausible-sounding mel-frames $x$. This transformation needs to be invertible to satisfy the change of variables requirement, and its behavior needs to be conditioned on text $\Phi$, speaker $\xi$, and alignment $A$ information. Following prior works (Kingma & Dhariwal, 2018; Dinh et al., 2014; 2016), we construct $g$ as a composition of invertible functions, specifically:

$$X = g(Z; \Phi, \xi, A) = g_1 \circ g_2 \cdots g_{K-1} \circ g_K(Z; \Phi, \xi, A) \quad (4)$$

$$Z = g_K^{-1} \circ g_{K-1}^{-1} \cdots g_2^{-1} \circ g_1^{-1}(X; \Phi, \xi, A) \quad (5)$$

Note that $X$ is the entire mel-spectrogram sequence for any audio clip and $Z$ is its corresponding projection into the latent domain with the same dimensions as $X$, and $z \sim \mathcal{N}(0, 1)$ for each $z \in Z$. Each compositional unit $g_k(X; \Phi, \xi, A)$ is an invertible network layer of $g$, henceforth referred to as a step of flow.

As with previous works (Kim et al., 2020; Miao et al., 2020; Prenger et al., 2019), we use a Glow-based (Kingma & Dhariwal, 2018) bipartite-flow architecture, where each step of flow is a $1 \times 1$ invertible convolution (Kingma & Dhariwal, 2018) paired with an affine coupling layer (Dinh et al., 2016). Detailed descriptions provided in the Appendix.

2.2.1. UNSUPERVISED ALIGNMENT LEARNING

We devise an unsupervised learning approach that learns the alignment between text and speech fast without any dependencies on external aligners. In practice the alignment converges rapidly toward a usable state in a few thousand iterations (roughly ten-twenty minutes into training). In order to learn to align $X$ with $\Phi$, we combine the Viterbi and forward-backward algorithms used in Hidden Markov Models (HMMs) (Rabiner, 1990) in order to learn hard and soft alignments, respectively. Let $A_{\text{soft}} \in \mathbb{R}^{N \times T}$ be the alignment matrix between text $\Phi$ and mel-frames $X$ of lengths $N$ and $T$ respectively, such that every column of $A_{\text{soft}}$ is normalized to a probability distribution. Sample alignment matrices are shown in Figure 2. Our goal is to extract monotonic, binarized alignment matrices $A_{\text{hard}}$ such that for every frame the probability mass is concentrated on a single symbol, and $\sum_{j=1}^T A_{\text{hard},i,j}$ yields a vector of durations of every symbol.
Parallel Flow-Based TTS with Robust Alignment Learning and Diverse Synthesis

**Extracting \( A_{\text{soft}} \):** Similar to Glow-TTS, the soft alignment distribution is based on the learned pairwise affinity between all text tokens \( \phi \in \Phi \) and mel-frames \( x \in X \), which is then normalized with softmax across the text dimension:

\[
D_{i,j} = \text{dist}_{L_2}(\phi_i^{\text{enc}}, x_j^{\text{enc}}), \quad (6)
\]
\[
A_{\text{soft}} = \text{softmax}(-D, \text{dim} = 0). \quad (7)
\]

Here, \( x^{\text{enc}} \) and \( \phi^{\text{enc}} \) are encoded variants of \( x \) and \( \phi \), each using a 2-3 layer 1D CNNs. We found that simple transformations with limited receptive fields produced best results. Conversely, expensive, wide receptive field transformations resulted in instabilities.

The forward-backwards algorithm maximizes the likelihood of hidden states given observations. We consider only with the forward probabilities. Text is defined as the hidden state and the mel as the observation to maximize \( P(S(\Phi) \mid X) \), considering all valid monotonic assignments to sequences \( s \in S(\Phi) \) where a specific sequence might look like: \( s : \{ s_1 = \phi_1, s_2 = \phi_2, \ldots s_T = \phi_N \} \). A monotonic sequence \( s \in S \) must be such that 1. It starts and ends at the first and last text tokens respectively; 2. It uses each text token \( \phi_n \) at least once; 3. The sequence can advance by 0 or 1 token for every advancement of mel-frame.

The likelihood of all valid monotonic alignments is:

\[
P(S(\Phi) \mid X) = \sum_{s \in S(\Phi)} \prod_{t=1}^{T} P(s_t \mid x_t). \quad (8)
\]

which is efficiently implemented using the PyTorch CTC loss module (see Appendix).

**Beta-Binomial Alignment Prior:** We accelerate the alignment learning using a cigar-shaped diagonal prior that promotes the elements on a near-diagonal path formulated using a Beta-Binomial distribution. While formulated differently, it is conceptually similar to the guided attention loss used in (Tachibana et al., 2018), and we believe model training benefits from virtually any reasonable diagonal-shaped prior. It can be visualized in Fig. 2b with details given in the Appendix.

**Extracting \( A_{\text{hard}} \):** Alignments generated through duration prediction are discrete. It is therefore necessary to condition the model \( g \) on the binarized alignment matrix \( A_{\text{hard}} \) to avoid creating a train-test domain gap. We achieve this using the Viterbi algorithm while applying the same constraints for a monotonic alignment as stated above. This gives us the most likely monotonic alignment from the distribution over monotonic paths defined by \( A_{\text{soft}} \).

As the Viterbi algorithm is not differentiable, training \( g \) conditioned on \( A_{\text{hard}} \) would mean that alignment-learning attention mechanism would receive no gradients from \( g \). Similar to (Kim et al., 2020), we further enforce that \( A_{\text{soft}} \) matches \( A_{\text{hard}} \) as much as possible by minimizing their KL-divergence: \( \mathcal{L}_{\text{bin}} = A_{\text{hard}} \odot \log A_{\text{soft}} \).

### 2.3. Generative Duration Modeling \( \mathcal{L}_{\text{dur}} \)

Recent parallel TTS architectures use deterministic regression models to predict durations of lexical units. This limits the amount of diversity achievable during inference in comparison to generative autoregressive models (Valle et al., 2020), where the duration is jointly sampled with other speech characteristics. We address this with a separate normalizing flow model solely for modeling \( \mathcal{L}_{\text{dur}} \) in Eq. 1. It can be constructed using either another bi-partite flow for full parallelism, or an autoregressive flow similar to the architecture in (Valle et al., 2020).

### 2.4. Training Schedule

Our model uses a training schedule to account for the evolving reliability of extracted alignments. Let \( \mathcal{L}_{\text{align}} \) be the minimization of the log likelihood of (8). \( \mathcal{L}_{\text{mel}} \) and \( \mathcal{L}_{\text{dur}} \) minimize (3) with respect to the decoder flow and phoneme flow respectively. The training begins with the loss function \( \mathcal{L} = \mathcal{L}_{\text{mel}} + \lambda_1 \mathcal{L}_{\text{align}} \), with the corresponding changes applied at given steps:

- \([0, 6k)\): Use \( A_{\text{soft}} \) for the alignment matrix.
- \([6k, 18k)\): start using Viterbi \( A_{\text{hard}} \) instead of \( A_{\text{soft}} \).
- \([18k, \text{end})\): add binarization term \( \lambda_2 \mathcal{L}_{\text{bin}} \) to the loss.
The duration predictor shares text embeddings with the decoder flow model, but is otherwise fundamentally disjoint and converges rapidly. Thus $L_{dur}$ is applied once the decoder has converged and its weights frozen.

### 3. Experiments

We investigate the subjective quality of our model, as well as how the beta-binomial prior and proposed data augmentation scheme influence training stability and alignment quality. For the sake of comparison with prior work, we perform the bulk of our experiments training only on the LJ speech dataset (LJ) (Ito et al., 2017). We use a sampling rate of 22,050 Hz and mel-spectrograms with 80 bins using Librosa’s (McFee et al., 2015) mel filter defaults. We apply the STFT with a FFT size of 1024, window size of 1024 samples and hop size of 256 samples (≈12 ms).

We use the normalized transcriptions provided in LJ and convert all unambiguous words to phoneme, keeping ambiguous words as graphemes. We use the public WaveGlow model (NVIDIA, 2021) trained on LJ for converting mel-spectrograms to waveforms, setting $\sigma=0.9$ and denoising strength 0.01 during inference. For inference from our model, we sample $z_{dur} \sim \mathcal{N}(0, \sigma=.7)$ and $z_{mel} \sim \mathcal{N}_{\text{trunc}}(0, \sigma = \{0.5, 0.667\})$, where the latter uses a truncated Normal with truncation at $1.1\sigma$.

#### 3.1. Speech Quality (MOS)

We subjectively compare RAD-TTS samples with Glow-TTS by collecting Mean Opinion Scores (MOS) on utterances from the LJ speech dataset (Ito et al., 2017). Glow-TTS-Blank denotes a variant of Glow-TTS with blank tokens interspersed between all phonemes. RAD-TTS models are trained on LJ with and without the Beta-Binomial attention prior. Lastly, we consider RAD-TTS-internal, which incorporates additional internal data featuring 2 additional female speakers.

We crowd-sourced MOS tests on Amazon Mechanical Turk (Table 1). Raters first pass a sinusoid-counting hearing test for eligibility. They then rate 10 randomly assigned samples from the LJ Test set. Pleasantness and naturalness are rated on a five-point scale. Over 480 scores were collected per model and each rater was exposed to exactly one model.

The beta-binomial attention prior demonstrably improves the baseline and achieves MOS similar to Glow-TTS-Blank. Further improvements in RAD-TTS-Internal demonstrate that the model scales further with additional data.

#### 3.2. Phoneme Duration Distribution

We compare samples and phoneme duration predictions from a deterministic model (Glow-TTS), a test-time dropout model (DP-Dropout-0.5) inspired by its use for output variability in Tacotron2 (Shen et al., 2018), a bi-partite normalizing flow (DPFP) and an autoregressive normalizing flow (DPFA). We compare the empirical duration distribution over the word Climate in the phrase Climate change knows no borders by sampling each model 100 times. We resample whenever the minimum duration is smaller than 25 ms and the maximum duration is larger than 300 ms.

Figure 3 plots the distribution of phoneme durations, with the proposed architectures producing the most variability. The deterministic baseline cannot synthesize speech with variable prosody. The DP-Dropout-0.5 model produces limited variance and is not as intuitively adjustable as is the $\sigma$ scaling in proper generative modeling.

### 4. Conclusion

We resolve the output diversity issue in parallel TTS architectures by proposing a dedicated generative flow for phoneme duration modeling. Our overall model is competitive with the similar Glow-TTS in speech quality and proposes improvements to the Glow/TTS unsupervised alignment framework through the use of simple unsupervised diagonal priors and the HMM forward-sum loss.

### Table 1.

<table>
<thead>
<tr>
<th>Model</th>
<th>Prior</th>
<th>MOS</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAD-TTS ($\sigma=0.667$)</td>
<td>✓</td>
<td>3.40 ± 0.10</td>
</tr>
<tr>
<td>RAD-TTS ($\sigma=0.667$)</td>
<td>✓</td>
<td>3.49 ± 0.10</td>
</tr>
<tr>
<td>RAD-TTS-Internal ($\sigma=0.667$)</td>
<td>✓</td>
<td>3.97 ± 0.06</td>
</tr>
<tr>
<td>Ground Truth (LJ)</td>
<td></td>
<td>4.11 ± 0.07</td>
</tr>
<tr>
<td>Glow-TTS-Blank ($\sigma=0.667$)</td>
<td></td>
<td>3.79 ± 0.09</td>
</tr>
<tr>
<td>Glow-TTS ($\sigma=0.667$)</td>
<td></td>
<td>3.56 ± 0.10</td>
</tr>
</tbody>
</table>

Figure 3. Phoneme-level duration distributions for the word Climate with 95% confidence intervals obtained from 100 samples collected from different models conditioned on the phrase Climate change knows no borders. Explicit generative models (shades of green and blue) provide high diversity in speech rhythm by adjusting $\sigma$, whereas test-time dropout (yellow) provides limited variability.
Parallel Flow-Based TTS with Robust Alignment Learning and Diverse Synthesis

References


A. Appendix

A.1. Affine Coupling Layer

Coupling layers (Dinh et al., 2014; 2016; Kingma & Dhariwal, 2018) are a family of invertible transformations wherein one split of the input data is used to infer scale and translation parameters to affine transform the rest, as described in Algorithm 1.

Algorithm 1 Affine Coupling Layer $f^{-1}_{coupling}(x; context)$

**input** $x, context$

**output** $s, t, x'$

$x_a, x_b = SPLIT(x)$

$(s, t) = f_{param}(x_a, context)$

$x'_b = s \circ x_b + t$

$x' = CONCAT(x_a, x'_b)$

Here $f_{param}()$ is an arbitrary function that predicts the affine parameters $s, b$ to be applied to the remaining channels $x_b$. In addition to the inputs $x_a$, the parameter prediction is critically conditioned on the context, which is a
We use the beta-binomial distribution which is discrete with a finite support, and probability mass function
\[
f_B(k;\alpha,\beta) = \binom{N}{k} \frac{B(k+\alpha)B(N-k+\beta)}{B(\alpha,\beta)}
\]
for \(k = \{0, \ldots, N\}\), where \(\alpha\) and \(\beta\) are hyperparameters of the beta function \(B(\cdot,\cdot)\). We use the beta-binomial distribution to construct a 2D cigar-shaped prior over the diagonal of \(A_{\text{soft}}\), which widens in the center of the matrix, and narrows towards the corners. For every column of \(A_{\text{soft}}\) representing \(P(\Phi \mid X = x_t)\), we incorporate the prior to attain the posterior:
\[
P_{\text{posterior}}(\Phi = \phi_k \mid X = x_t) = P(\Phi = \phi_k \mid X = x_t) \odot f_B(k, \omega t, \omega(T - t + 1))
\]
for \(k = \{0, \ldots, N\}\), where \(\omega\) is the scaling factor controlling the width: lower the \(\omega\), wider is the width of the prior.

**A.5. CTC Formulation**

**Generic HMM Formulation:** Let \(\lambda = (A, B, \pi)\) be the parameters of an HMM: \(A\) the state transition matrix, \(B\) the emission probability matrix, and \(\pi\) the vector of initial state probabilities. Denote by \(b_t(j)\) the emission probability of symbol \(j\) in the \(t\)th state.

For an observation sequence \(O = \{O_1, \ldots, O_n\}\), the probability of such sequence can be computed recursively (Rabiner, 1990)
\[
\alpha_1(i) = \pi_i b_1(O_1) \quad 1 \leq i \leq N,
\]
\[
\alpha_{t+1}(j) = \sum_{i=1}^{N} \alpha_t(i) a_{ij} b_j(O_{t+1})
\]
for \(1 \leq t \leq T - 1, \quad 1 \leq j \leq N.
\]

The final probability is the sum of final forward variables over all states
\[
P(O \mid \lambda) = \sum_{i=1}^{N} \alpha_T(i).
\]

CTC imposes several constraints on this general formulation. Let
\[
\text{The quick brown fox}
\]
be a sample transcription of an input audio signal. In CTC, it is being interleaved with blank tokens \(\emptyset\), giving raise to the sequence of observations
\[
O = \{\emptyset, \ T, \ \emptyset, \ h, \ \emptyset, \ e, \ldots, \ e, \ \emptyset, \ o, \ \emptyset, \ x, \ \emptyset\}.
\]

In CTC, initial state probabilities \(\pi\) are not modeled. The state-transition matrix \(A\) is sparse and allows only a handful of monotonic transitions from neighboring states, giving
Parallel Flow-Based TTS with Robust Alignment Learning and Diverse Synthesis

raise to the recursive formulation of Eq. (15) in CTC

\[
\alpha_t(s) = \begin{cases} 
\left[ \alpha_{t-1}(s) + \alpha_{t-1}(s-2) \right] b_t(O_s) & \text{if } O_s = \emptyset \text{ or } O_s = O_{s-2}, \\
\left[ \alpha_{t-1}(s) + \alpha_{t-1}(s-2) + \alpha_{t-1}(s-1) \right] b_t(O_s) & \text{otherwise}.
\end{cases}
\]

(17)

It tells that a transition can occur only to the same \(\alpha_{t-1}(s)\), or a subsequent state \(\alpha_{t-1}(s-1)\), and blank states can be skipped altogether \(\alpha_{t-1}(s-2)\). The condition \(O_s = O_{s-2}\) ensures that when there are repeated letters (e.g., in the word `Anna`), the blank state has to be visited in order to later differentiate between both occurrences of the repeated letter.

When using CTC, the speech recognition model outputs an emission probability matrix \(B \in \mathbb{R}^{|V| \times T}\), where \(|V|\) denotes the size of the vocabulary with the blank symbol (and so the number of states), and \(T\) is the number of outputs proportional to the number of input audio frames. Every column in \(B\) is normalized to a probability distribution.

Calculating \(A_{\text{soft}}\): In RAD-TTS, the alignment matrix \(A_{\text{soft}} \in \mathbb{R}^{N \times T}\) can be interpreted similarly to \(B\), as a series of column-wise emission probability distributions over possible \(N\) input symbol (states) for each of \(T\) audio frames. To ensure monotonicity of alignment, we build a surrogate transcript with a sequence of unique states

\[\{s_1, s_2, \ldots, s_N\}\]

which, upon passing to CTC loss, is being converted to

\[O = \{\emptyset, s_1, \emptyset, s_2, \ldots, \emptyset, s_N, \emptyset\} .\]

As in CTC, we do not model initial probabilities \(\pi\). In order to account for the blank symbol state, we add a row to \(A_{\text{soft}}\)

\[
\begin{bmatrix}
A_{\text{soft}} \\
\epsilon
\end{bmatrix}
\]

where \(\epsilon\) denotes a constant blank probability for every frame. We set to a tiny, non-zero value, to prevent numerical instability. The recursive formulation from Eq. (17) can now be broken down into two cases:

1. \(O_t = \emptyset\) (odd positions in the transcript denoted \(2k+1\)).

   From \(b_t(\emptyset) = 0\) it follows that \(\alpha_t(2k+1) = 0\).

2. \(O_t \neq \emptyset\) (even positions in the transcript denoted \(2k\)).

The surrogate sequence has unique states and \(O_t \neq \emptyset\). It follows that \(O_t \neq O_{t-2}\), and the first case of Eq. (17) is not possible. Because \(\alpha_{t-1}(2k-1) = 0\), we have

\[
\alpha_t(2k) = [\alpha_{t-1}(2k) + \alpha_{t-1}(2k - 2)] b_t(O_s) .
\]

Thus, with negligible blank probability and under the imposed monotonicity constraint, Eq. (17) effectively reduces to Eq. (15) from the original formulation for calculating forward variables.

In practice, setting the blank emission probability \(\epsilon\) to be roughly the value of the largest of the initial activations significantly improves convergence rates. The reasoning behind this is that it relaxes the monotonic constraint, allowing the objective function to construct paths while optionally skipping over some text tokens, notably ones that have not been sufficiently trained on during early iterations. As training proceeds, the probabilities of the skipped text token increases, despite the existence of the blank tokens, allowing us to extract clean monotonic alignments.
Example Implementation:

```python
class ForwardSumLoss(torch.nn.Module):
    def __init__(self, blank_logprob=-1):
        super(ForwardSumLoss, self).__init__()
        self.log_softmax = torch.nn.LogSoftmax(dim=3)
        self.blank_logprob = blank_logprob
        self.CTCLoss = nn.CTCLoss(zero_infinity=True)

    def forward(self, attn_logprob, text_lens, mel_lens):
        # The CTC loss module assumes the existence of a blank token
        # that can be optionally inserted anywhere in the sequence for
        # a fixed probability.
        # A row must be added to the attention matrix to account for this
        attn_logprob_pd = F.pad(input=attn_logprob,
                                pad=(1, 0, 0, 0, 0, 0, 0, 0),
                                value=self.blank_logprob)
        cost_total = 0.0
        for bid in range(attn_logprob.shape[0]):
            # construct the target sequence. Every
            # text token is mapped to a unique sequence number,
            # thereby ensuring the monotonicity constraint
            target_seq = torch.arange(1, text_lens[bid]+1)
            target_seq=target_seq.unsqueeze(0)
            curr_logprob = attn_logprob_pd[bid].permute(1, 0, 2)
            curr_log_prob = curr_logprob[:mel_lens[bid],:,:text_lens[bid]+1]
            curr_logprob = self.log_softmax(curr_logprob[None])[0]
            cost = self.CTCLoss(curr_logprob,
                                target_seq,
                                input_lengths=mel_lens[bid:bid+1],
                                target_lengths=text_lens[bid:bid+1])
            cost_total += cost
        # average cost over batch
        cost_total = cost_total/attn_logprob.shape[0]
        return cost_total
```