Speech Synthesis By Unrolling Diffusion Process using Neural Network Layers

Anonymous authors
Paper under double-blind review

Abstract

This work proposes a novel setup where a neural network is trained to predict multiple steps of the reverse diffusion process in an unrolled manner, with successive layers corresponding to equally spaced steps in the diffusion schedule. Each layer progressively denoises the input during the reverse process until the final layer estimates the original input, x_0. Additionally, we introduce a new learning target by using latent variables, rather than the conventional approach of predicting the original input x_0 or source error ϵ_0. In speech synthesis, using x_0 or ϵ_0 often leads to large prediction errors in the early stages of the denoising process, causing distortion in the recovered speech. Our method mitigates this issue and, through extensive evaluation, demonstrates the generation of high-fidelity speech in competitive time, outperforming current state-of-the-art techniques. Moreover, the proposed approach generalizes well to unseen speech. Sample audio is available at `https://onexpeters.github.io/UDPNet/`.

1 Introduction

Diffusion Probability Models (DPMs) (Sohl-Dickstein et al., 2015) have gained popularity in recent years for speech synthesis tasks (Lam et al., 2022; Chen et al., 2020; Kong et al., 2020b), owing to their ability to model complex data distributions. These models rely on two key processes: the forward process, where Gaussian noise is progressively added to the data until it becomes white noise, and the reverse process, where the model aims to recover the original data distribution by denoising.

While DPMs excel at generating high-quality outputs, they typically require a large number of diffusion steps during training, which leads to an equally high number of reverse steps during sampling. This results in slow inference times, making diffusion-based speech synthesis models unsuitable for real-time or low-latency applications. For instance, models like WaveGrad (Chen et al., 2020) mitigate this issue by optimizing the noise schedule through a grid search algorithm, while BDDM (Lam et al., 2022) reduces the number of reverse steps by training a scheduling network that significantly shortens the noise schedule. However, even with these optimizations, methods like grid search remain computationally expensive and inefficient when handling large numbers of noise steps.

To address these limitations, we propose a novel approach that unrolls the reverse diffusion process into neural network layers. Instead of mapping each network layer to a single diffusion step, we reduce the number of reverse steps by setting the number of layers in the network to $N = \frac{T}{\tau}$, where $\tau > 1$ and T is the total number of forward diffusion steps. Each neural network layer is responsible for denoising the cumulative noise from multiple forward steps (τ steps), drastically reducing the required number of layers. This approach simplifies the reverse process, speeds up sampling, and lowers computational costs without sacrificing the quality of speech generation.

In contrast to previous work such as (Lam et al., 2022), which requires the optimization of two separate parameter sets, our method optimizes a single parameter set, θ, making it more efficient in terms of both training and inference time. Additionally, we introduce a new learning target: instead of predicting the original input x_0 or the noise ϵ_0, we use latent variables as the prediction target. In speech synthesis, predicting x_0 or ϵ_0 can result in high prediction errors during the early steps of the reverse diffusion process,

leading to artifacts and distortions in the synthesized speech. Latent variables, however, provide a more stable and abstract representation, reducing the error during early denoising stages and enabling smoother speech recovery.

This new approach not only leads to faster audio generation but also improves the generalization capability to unseen speech. By leveraging latent variables, we achieve high-fidelity speech synthesis while significantly reducing sampling times, making our model more suitable for real-time applications such as AI-powered voice assistants, voice cloning, and other speech-based technologies.

2 Background

2.1 Denoising diffusion probabilistic model(DDPM)

Given an observed sample x of unknown distribution, DDPM defines a forward process as:

$$q(x_{1:T}|x_0) = \prod_{i=1}^{T} q(x_t|x_{t-1}) \tag{1}$$

Here, latent variables and true data are represented as x_t with $t = 0$ being the true data. The encoder $q(x_t|x_{t-1})$ seeks to convert the data distribution into a simple tractable distribution after the T diffusion steps. $q(x_t|x_{t-1})$ models the hidden variables x_t as linear Gaussian models with mean and standard centered around its previous hierarchical latent x_{t-1}. The mean and standard deviation can be modelled as hyperparameters (Ho et al., 2020) or as learnt variables (Nichol & Dhariwal, 2021) (Kingma et al., 2021). The Gaussian encoder's mean and variance are parameterized as $u_t(x_t) = \sqrt{\alpha_t}x_{t-1}$ and $\Sigma_q(x_t) = (1-\alpha_t)I$ respectively, hence the encoder can be expressed as $q(x_t|x_{t-1}) = \mathcal{N}(x_t; \sqrt{\alpha_t}x_{t-1}, (1-\alpha_t)I)$ where α_t evolves with time t based on a fixed or learnable schedule such that the final distribution $p(x_T)$ is a standard Gaussian. Using the property of isotropic Gaussians, Ho et al. (2020) show that x_t can be derived directly on x_0 as:

$$x_t = \sqrt{\bar{\alpha}_t}x_0 + \sqrt{(1-\bar{\alpha}_t)}\epsilon_0 \tag{2}$$

where $\bar{\alpha}_t = \prod_{t=1}^{t} \alpha_t$ and $\epsilon_0 \sim \mathcal{N}(\epsilon_0; 0, I)$ hence $q(x_t|x_0) = \mathcal{N}(x_t; \sqrt{\bar{\alpha}_t}x_0, (1-\bar{\alpha}_t)I)$. The reverse process which seeks to recover the data distribution from the white noise $p(x_T)$ is modelled as:

$$p_\theta(x_{0:T}) = p(x_T) \prod_{i=1}^{T} p_\theta(x_{t-1}|x_t) \tag{3}$$

where $p(x_T) = \mathcal{N}(x_T; 0, I)$. The goal of DPM is therefore to model the reverse process $p_\theta(x_{t-1}|x_t)$ so that it can be exploited to generate new data samples. After the DPM has been optimized, a sampling procedure entails sampling Gaussian noise from $p(x_T)$ and iteratively running the denoising transitions $p_\theta(x_{t-1}|x_t)$ for T steps to generate x_0. To optimize DPM, evidence lower bound (ELBO) in equation 4 is used.

$$\begin{aligned} &\log p(x) = E_{q(x_0)}[D_{KL}(q(x_T|x_0)||p(x_T))+ \\ &\sum_{t=2}^{T} E_{q(x_t|x_0)}[D_{KL}(q(x_{t-1}|x_t, x_0)||p_\theta(x_{t-1}|x_t))]- \\ &E_{q(x_1|x_0)}[\log p_\theta(x_0|x_1)] \end{aligned} \tag{4}$$

In equation 4, the second term on the right is the denoising term that seeks to model $p_\theta(x_{t-1}|x_t)$ to match the ground truth $q(x_{t-1}|x_t, x_0)$. In (Ho et al., 2020), $q(x_{t-1}|x_t, x_0)$ is derived as:

$$\begin{aligned} q(x_{t-1}|x_t, x_0) =& \mathcal{N}(\frac{\sqrt{\alpha}(1-\bar{\alpha}_{t-1})x_t + \sqrt{\bar{\alpha}}_{t-1}(1-\alpha_t)x_0}{(1-\bar{\alpha}_t)}, \\ & \frac{(1-\alpha_t)(1-\bar{\alpha}_{t-1})}{(1-\bar{\alpha_t})}I) \end{aligned} \tag{5}$$

In order to match $p_\theta(x_{t-1}|x_t)$ to $q(x_{t-1}|x_t, x_0)$ during the reverse process, $p_\theta(x_{t-1}|x_t)$ is modeled with the same variance as that of $q(x_{t-1}|x_t, x_0)$ i.e $\Sigma_q(t) = \frac{(1-\alpha_t)(1-\bar{\alpha}_{t-1})}{(1-\bar{\alpha}_t)} I$. The mean of $p_\theta(x_{t-1}|x_t)$ is made to match that of $q(x_{t-1}|x_t, x_0)$ hence it is parameterized as:

$$u_\theta(x_t, t) = \frac{\sqrt{\alpha}(1-\bar{\alpha}_{t-1})x_t + \sqrt{\bar{\alpha}_{t-1}}(1-\alpha_t)\hat{x}_\theta(x_t, t)}{(1-\bar{\alpha}_t)} \tag{6}$$

Here, the score network $\hat{x}_\theta(x_t, t)$ is parameterized by a neural network and it seeks to predict x_0 from a noisy input x_t and time index t. Hence,

$$\begin{aligned} &p_\theta(x_{t-1}|x_t) \\ &= \mathcal{N}(\frac{\sqrt{\alpha}(1-\bar{\alpha}_{t-1})x_t + \sqrt{\bar{\alpha}_{t-1}}(1-\alpha_t)\hat{x}_\theta(x_t, t)}{(1-\bar{\alpha}_t)}, \\ &\frac{(1-\alpha_t)(1-\bar{\alpha}_{t-1})}{(1-\bar{\alpha}_t)} I) \end{aligned} \tag{7}$$

Therefore, optimizing the KL divergence between the two Gaussian distributions of $q(x_{t-1}|x_t, x_0)$ and $p_\theta(x_{t-1}|x_t)$ can be formulated as:

$$\begin{aligned} &L_{t-1} = \\ &\arg\min_\theta E_{t\sim U(2,T)} D_{KL}(q(x_{t-1}|x_t, x_0)||p_\theta(x_{t-1}|x_t)) \end{aligned} \tag{8}$$

$$\begin{aligned} L_{t-1} = &\arg\min_\theta E_{t\sim U(2,T)} D_{KL} \\ &(\mathcal{N}(x_{t-1}; \mu_q, \Sigma_q(t)||\mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \Sigma_q(t)) \end{aligned} \tag{9}$$

Here,

$$\mu_q = \frac{\sqrt{\alpha}(1-\bar{\alpha}_{t-1})x_t + \sqrt{\bar{\alpha}}_{t-1}(1-\alpha_t)x_0}{(1-\bar{\alpha}_t)}$$

Equation 9 is simplified as (see (Luo, 2022)):

$$L_{t-1} = \arg\min_\theta E_{t\sim U(2,T)}[||\hat{x}_\theta(x_t, t) - x_0||_2^2] \tag{10}$$

The loss function is composed of the neural network $\hat{x}_\theta(x_t, t)$ that is conditioned on the discrete time t and noisy input x_t to predict the original ground truth input x_0. By rearranging equation 2 as:

$$x_0 = \frac{x_t - \sqrt{1-\bar{\alpha}_t}\epsilon_0}{\sqrt{\bar{\alpha}_t}} \tag{11}$$

an equivalent optimization of modelling a neural network $\hat{\epsilon}_\theta(x_t, t)$ to predict the source noise can be derived (Ho et al., 2020).

$$L_{t-1} = \arg\min_\theta E_{t\sim U(2,T)}[||\hat{\epsilon}_\theta(x_t, t) - \epsilon_0||_2^2] \tag{12}$$

Work in (Ho et al., 2020) uses L_{t-1} as an optimization of the ELBO.

3 Related work

Deep neural network generative techniques for speech synthesis (vocoders) are either implemented using likelihood technique or generative adversarial network (Goodfellow, 2016). Likelihood methods are composed of autoregressive, VAE, flow, and diffusion-based vocoders. Autoregressive models such as (Oord et al., 2016) (Kalchbrenner et al., 2018) (Mehri et al., 2016) and (Valin & Skoglund, 2019) are models that generate speech sequentially. The models learn the joint probability over speech data by factorizing the distribution into a product of conditional probabilities over each sample. Due to their sequential nature of speech generation, autoregressive models require a large number of computations to generate a sample. This limits their ability

to be deployed in application where faster real time generation is required. However, there are models such as (Paine et al., 2016), (Hsu & Lee, 2020) and (Mehri et al., 2016) which propose techniques to speed up speech generation in autoregressive models. Another likelihood-based speech synthesis technique is the flow-based models (Rezende & Mohamed, 2015) used in (Prenger et al., 2019) (Kim et al., 2020) (Hsu & Lee, 2020). These models use a sequence of invertible mappings to transform a given probability density. During sampling, flow-based models generate data from a probability distribution through the inverse of these transforms. Flow based models implement specialized models that are is complicated to train Tan et al. (2021). Denoising diffusion probabilistic models (DDPM) have recently been exploited in speech synthesis using tools such as PriorGrad (Lee et al., 2021), WaveGrad (Chen et al., 2020), BDDM (Lam et al., 2022) and DiffWave (Kong et al., 2020b). These models exploit a neural network that learns to predict the source noise that was used in the noisification process during the forward process. Diffusion-based vocoders can generate speech with very high voice quality but are slow due to the high number of sampling steps. Tools such as BDDM (Lam et al., 2022) propose techniques to speed up speech generation while using diffusion models. Our proposed work also looks at how to speed up speech synthesis in diffusion models. Finally, GAN based models such as (Kong et al., 2020a) and (Kumar et al., 2019) exploit the training objective to make the model generate data that is indistinguishable from the training data. While GAN based models can generate high quality speech, they are difficult to train due to instability during the training process (Mescheder et al., 2018). A complete review of the vocoders can be found in (Tan et al., 2021).

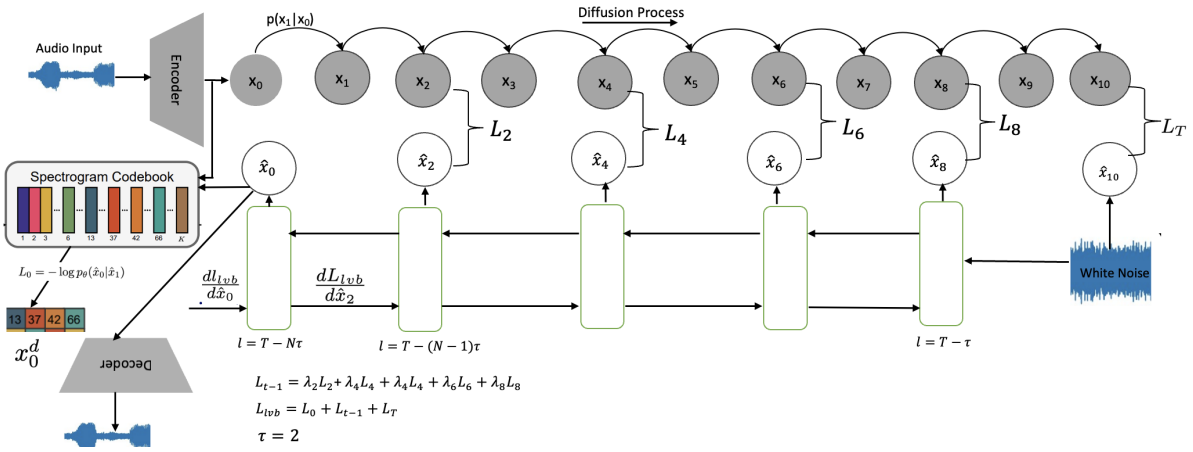

Figure 1: An overview of the unconditioned audio generation. An input audio is processed by a pre-trained model to generate x_0. x_0 is then processed by forward process to generate latent variables x_t. In the reverse process, white noise x_T is passed through the first layer of the neural network and processed through the subsequent layers. A layer is mapped to a given time step t of the forward process. If a layer l is mapped to a time step t, an error L_i is computed by establishing l_2 norm between their respective embeddings.

3.0.1 Forward Process

During the forward process, given a raw audio waveform input x, we use an encoder to generate its representation $x_0 \in \mathbb{R}^{f \times h}$, where f is the number of frames and h is the hidden dimension size. From this representation, latent variables x_t are generated based on the forward diffusion process, as described in Equation 2, where $1 \leq t \leq T$.

We also employ a discrete codebook of size K with dimension h, i.e., $\mathcal{Z} = \{z_k\}_{k=1}^{K} \in \mathbb{R}^h$, such that each row of x_0 is mapped to its closest entry in the codebook (illustrated in Figure 1). These codebook indices play a crucial role in reconstructing the signal during the reverse diffusion process.

$$z_q = \left(\arg\min_{z_k \in \mathcal{Z}} \|x_0^i - z_k\|_2^2 \; \forall i \in f \right) \in \mathbb{R}^{f \times h} \tag{13}$$

Equation 13 identifies the nearest codebook vector z_k for each row x_0^i by minimizing the squared Euclidean distance. The discrete indices, x_0^d, representing the codebook entries assigned to each row of x_0, are stored and will be used during the reverse diffusion process to recover the original signal.

3.0.2 Reverse process

The ELBO (Equation 4) used for optimizing diffusion probabilistic models consists of three key parts:

$$
\begin{aligned}
L_0 &= \mathbb{E}_{q(x_1|x_0)} \left[\log p_\theta(x_0|x_1)\right] \\
L_T &= \mathbb{E}_{q(x_0)} \, \mathrm{D_{KL}} \left(q(x_T|x_0) \| p(x_T)\right) \\
L_{t-1} &= \sum_{t=2}^{T} \mathbb{E}_{q(x_t|x_0)} \left[\mathrm{D_{KL}} \left(q(x_{t-1}|x_t, x_0) \| p_\theta(x_{t-1}|x_t)\right)\right]
\end{aligned}
$$

The total loss, based on these three parts, is defined as:

$$
L_{vlb} = L_0 + \sum_{t=1}^{T-1} L_{t-1} + L_T \tag{14}
$$

The term L_{t-1}, also known as the denoising term, is critical for teaching the model to estimate the transition $p_\theta(x_{t-1}|x_t)$, which approximates the true distribution $q(x_{t-1}|x_t, x_0)$. Minimizing the KL divergence between these two distributions ensures that the model can effectively remove noise and progressively recover the original data.

To make the denoising process more computationally feasible for our proposed layer-based recovery technique, we introduce the approximations $\hat{x}_{t-1}$ and $\hat{x}_t$ in place of the actual values. This parameterized version of L_{t-1} is given by:

$$
L_{t-1} = \sum_{t=2}^{T} \mathbb{E}_{q(x_t|x_0)} \left[\mathrm{D_{KL}} \left(q(x_{t-1}|x_t, x_0) \| p_\theta(\hat{x}_{t-1}|\hat{x}_t)\right)\right] \tag{15}
$$

L_{t-1} is minimised across different noise levels and timesteps via apply stochastic sampling by selecting random timesteps from a uniform distribution:

$$
L_{t-1} = \arg\min_\theta \mathbb{E}_{t\sim U(2,T)} \, \mathrm{D_{KL}} \left(q(x_{t-1}|x_t, x_0) \| p_\theta(\hat{x}_{t-1}|\hat{x}_t)\right) \tag{16}
$$

Hence,

$$
L_{t-1} = \arg\min_\theta \mathbb{E}_{t\sim U(2,T)} \, \mathrm{D_{KL}} \left(\mathcal{N}(x_{t-1}; \mu_q(t), \Sigma_q(t)) \| \mathcal{N}(\hat{x}_{t-1}; \hat{\mu}_\theta, \Sigma_q(t))\right) \tag{17}
$$

where $\mu_q(t)$ and $\Sigma_q(t)$ are defined as:

$$
\mu_q(t) = \frac{\sqrt{\alpha}(1-\bar{\alpha}_{t-1})x_t + \sqrt{\bar{\alpha}}_{t-1}(1-\alpha_t)x_0}{1-\bar{\alpha}_t},
$$

$$
\Sigma_q(t) = \frac{(1-\alpha_t)(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t} I.
$$

Our goal is to model $p_\theta(\hat{x}_{t-1}|\hat{x}_t)$ to have a distribution as close as possible to $q(x_{t-1}|x_t, x_0)$ (Ho et al., 2020). Therefore, we approximate $p_\theta(\hat{x}_{t-1}|\hat{x}_t)$ as a Gaussian with mean $\hat{\mu}_\theta$ and variance $\Sigma_q(t)$, where $\hat{\mu}_\theta$ is defined as:

$$\hat{\mu}_\theta = \frac{\sqrt{\alpha}(1-\bar{\alpha}_{t-1})x_\theta(\hat{x}_{t+1}, t) + \sqrt{\bar{\alpha}}_{t-1}(1-\alpha_t)x_0}{1-\bar{\alpha}_t}.$$

Here, $x_\theta(\hat{x}_{t+1}, t)$ is parameterized by a neural network that predicts x_t, given the noisy estimate $\hat{x}_{t+1}$ and the timestep t. The network learns to predict the denoised x_t at each step of the reverse process.

Using this definition of $\hat{\mu}_\theta$, the loss term L_{t-1} can be expressed as:

$$L_{t-1} = \arg\min_\theta \mathbb{E}_{t\sim U(1,T-1)} \frac{1}{2\Sigma_q(t)} \left|\left| \frac{\sqrt{\alpha}(1-\bar{\alpha}_{t-1})x_\theta(\hat{x}_{t+1}, t) + \sqrt{\bar{\alpha}}_{t-1}(1-\alpha_t)x_0}{1-\bar{\alpha}_t} - \frac{\sqrt{\alpha}(1-\bar{\alpha}_{t-1})x_t + \sqrt{\bar{\alpha}}_{t-1}(1-\alpha_t)x_0}{1-\bar{\alpha}_t} \right|\right|_2^2. \tag{18}$$

Equation 18 can be simplified as (see Appendix A for the complete derivation):

$$L_{t-1} = \arg\min_\theta \mathbb{E}t \sim U(1, T-1) \frac{\sqrt{\alpha}(1-\bar{\alpha}t-1)}{2\Sigma_q(t)(1-\bar{\alpha}t)} \left[|\hat{x}\theta(\hat{x}_{t+1}, t) - x_t|_2^2\right] \tag{19}$$

Optimizing L_{t-1} essentially boils down to learning a neural network $\hat{x}_\theta(\hat{x}_{t+1}, t)$, which is conditioned on the estimated variable $\hat{x}_{t+1}$ and a timestep t to predict x_t. This is different from the loss in Equation 10, where the neural network $\hat{x}_\theta(x_t, t)$ is conditioned on a noisy input x_t and a timestep t to predict the original noiseless input x_0.

To estimate a latent variable x_t using Equation 19, one would theoretically need a separate neural network for each timestep t. For T latent variables, this would require $T-1$ neural networks. To make this more practical, we exploit the layers of a single neural network, where each layer corresponds to a timestep. If a neural network has N layers, it effectively represents N neural networks. When $N = T-1$, there is a 1:1 mapping between each timestep $t \in [1, T-1]$ of the forward process and a neural network layer.

However, to accelerate the reverse diffusion process, we introduce a timestep skip parameter $\tau > 1$, such that $N = \frac{T}{\tau}$. By doing this, the data distribution is recovered in N steps (or layers), which is significantly fewer than T, thus speeding up the data recovery process.

The reverse process starts with white noise $x_T \sim \mathcal{N}(0, I)$, which is passed through the first layer of the neural network at timestep $l = T - \tau$. It is then processed by subsequent layers at timesteps $l = T - n\tau$ for $2 \leq n \leq N$ (see Figure 1). Each layer $l = T - n\tau$ generates an estimated representation $\hat{x}_{T-n\tau} \in \mathbb{R}^{f\times h}$, which is passed to the next layer $l + 1 = T - (n+1)\tau$ to estimate $\hat{x}_{T-(n+1)\tau} \in \mathbb{R}^{f\times h}$.

This sequential implementation implies that the timesteps t are implicitly encoded by the neural network layers, eliminating the need to explicitly condition the layers on time t. Thus, Equation 19 is implemented as:

$$L_{t-1} = \sum_{t=T-\tau}^{T-(N-1)\tau} \lambda_t \left|\left|\hat{x}_\theta^{l=t}(\hat{x}_{t+\tau}) - x_t\right|\right|_2^2 \tag{20}$$

The loss term L_{t-1} is optimized by learning a neural network $\hat{x}_\theta(\hat{x}_{t+1}, t)$, which predicts x_t conditioned on the estimated variable $\hat{x}_{t+1}$ and timestep t. Here, λ_t represents the contribution of the layer $l = t$ to the overall loss L_{t-1}. In (Ho et al., 2020), t is sampled randomly, and the expectation $E_{t,x_0,\epsilon_0}[L_{t-1}]$ (Equation 16) is used to estimate the variational lower bound L_{vlb} (Equation 18). However, the method proposed

by (Ho et al., 2020) results in samples that do not achieve competitive log-likelihoods (Nichol & Dhariwal, 2021). Log-likelihood is a key metric in generative models, driving them to capture all modes of the data distribution (Razavi et al., 2019). Inspired by this, we aim to optimize the full L_{vlb} efficiently.

To compute the loss L_0, the output $\hat{x}_{T-(N-1)\tau}$ from the layer $l = T - (N-1)\tau$ is passed to the final layer $l = T - N\tau$ of the neural network. The final predicted $\hat{x}_0$ is then given by $\hat{x}_0 = \hat{x}_\theta^{l=T-N\tau}(\hat{x}_{T-(N-1)\tau})$. This prediction is used to estimate the probability $p_\theta(\hat{x}_0 \mid \hat{x}_{T-(N-1)\tau})$, which predicts the original indices of the input x_0^d as defined by the codebook (see Figure 1). Similar to Nichol & Dhariwal (2021), we use the cumulative distribution function (CDF) of a Gaussian distribution to estimate $p_\theta(\hat{x}_0 \mid \hat{x}_{T-(N-1)\tau})$. The loss L_0 is computed as:

$$L_0 = -\log p_\theta(\hat{x}_0 \mid \hat{x}_{T-(N-1)\tau}) \tag{21}$$

The term L_T is not modeled by the neural network and does not depend on θ. It approaches zero if the forward noising process sufficiently corrupts the data distribution such that $q(x_T \mid x_0) \approx \mathcal{N}(0, I)$. This term can be computed as the KL divergence between two Gaussian distributions. Therefore, the total variational loss is defined as:

$$L_{vlb} = L_0 + L_{t-1} + L_T \tag{22}$$

However, during implementation, we ignore L_T and compute the loss as:

$$L_{vlb} = L_0 + L_{t-1} \tag{23}$$

We observed that including L_T leads to higher loss values (see Figure 2), due to the fact that the end of the forward noising process introduces too much noise, complicating the recovery of the data distribution (Nichol & Dhariwal, 2021). While (Nichol & Dhariwal, 2021) addressed this by proposing a new noise schedule, we remove the L_T term entirely from L_{vlb}.

Algorithms 1 and 2 summarize the training and sampling procedures of the proposed method.

Algorithm 1 Training Algorithm with τ, T, x_0, Codebook $\mathcal{Z}$

Initialize $N = \frac{T}{\tau}$
repeat
Initialize $L_{t-1} = 0$
Map x_0 rows to codebook indices: x_0^d
Set $\lambda_t = 0.001$
Initialize $\hat{x}_{t+\tau} = x_T \sim \mathcal{N}(0, I)$
Sample noise: $\epsilon_0 \sim \mathcal{N}(0, I)$
for $t = T - \tau$ to $T - (N-1)\tau$:
  Update loss: $L_{t-1} += \lambda_t \left\| x_\theta^{l=t}(\hat{x}_{t+\tau}) - \sqrt{\bar{\alpha}_t} x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon_0 \right\|_2^2$
  Update prediction: $\hat{x}_{t+\tau} = x_\theta^{l=t}(\hat{x}_{t+\tau})$
  Increment weight: $\lambda_t += 0.001$
  if $t = T - (N-1)\tau$:
    Predict $\hat{x}_0 = x_\theta^{l=t-\tau}(\hat{x}_{t+\tau})$
    Compute likelihood $p(\hat{x}_0 \mid \hat{x}_t)$ for restoring x_0^d
    Compute loss: $L_0 = -\log p(\hat{x}_0 \mid \hat{x}_t)$
Compute total loss: $L_{vlb} = L_0 + L_{t-1}$
Update the neural networks $x_\theta^l(.)$ to minimize L_{vlb}
until L_{vlb} converges

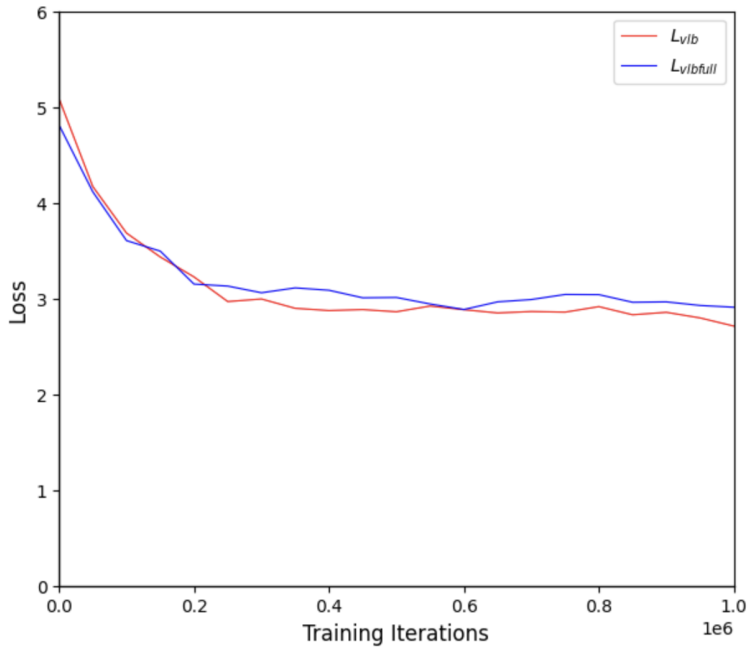

Figure 2: Learning curves comparing the full objective $L_{vlbfull} = L_0 + L_{t-1} + L_T$ and L_{vlb} on the LJSpeech dataset.

Algorithm 2 Sampling Algorithm with τ, x_t, $x_\theta^{l=t}(.)$, and $T - \tau \leq t \leq T - N\tau$

1: Initialize $\hat{x}_{t+\tau} = x_T \sim \mathcal{N}(0, I)$
2: **for** $t = T - \tau$ to $T - N\tau$:
3: Update estimate: $x_t = x_\theta^{l=t}(\hat{x}_{t+\tau})$
4: Update prediction: $\hat{x}_{t+\tau} = x_\theta^{l=t}(\hat{x}_{t+\tau})$
5: **end for**
6: Return final prediction: $x_{T-N\tau} = x_0$

3.1 Conditional Speech Generation

To enable the model to generate speech conditioned on specific acoustic features, we modify the neural network layer to incorporate these features, denoted as y. The loss function is now defined as:

$$L_{t-1} = \sum_{t=T-\tau}^{T-(N-1)\tau} \lambda_t \left\| \hat{x}_\theta^{l=t}(\hat{x}_{t+\tau}, y) - x_t \right\|_2^2 \tag{24}$$

We design the score network $\hat{x}_\theta^l(.,.)$ to process both the estimated value $\hat{x}_{t+\tau}$ and the acoustic features y. To achieve this, we use feature-wise linear modulation (FiLM) (Perez et al., 2018), as used in (Chen et al., 2020). FiLM adaptively influences the layer activations by applying an affine transformation based on the input Mel spectrogram y (see Equation 25).

$$FiLM(\hat{x}_{t+\tau}) = \gamma \odot \hat{x}_{t+\tau} + \beta \tag{25}$$

Here, both γ and $\beta \in \mathbb{R}^{f \times h}$ modulate $\hat{x}_{t+1}$ based on the Mel spectrogram y, and $\odot$ represents the Hadamard product.

To compute L_0 for conditional generation, we first estimate the conditional probability $p_\theta(\hat{x}_0 \mid \hat{x}_{T-(N-1)\tau}, y)$, which predicts the original indices x_0^d of the input x_0 as established by the codebook. The loss L_0 is then computed as:

$$L_0 = -\log p_\theta(\hat{x}_0 \mid \hat{x}_{T-(N-1)\tau}, y) \tag{26}$$

4 Alternative Loss Functions

To improve the quality of generated speech samples, we explored alternative objective functions. These loss functions aim to balance simplicity with performance, focusing on generating clearer and more accurate speech.

The first alternative, shown in Equation 27, is a simplified version of the original loss in Equation 10. This loss minimizes the difference between the predicted output $\hat{x}_\theta$ and the original input x_0, making it more computationally efficient.

$$L_{simple} = \left\| \hat{x}_\theta^{l=T-(N-1)\tau}(\hat{x}_{t+\tau}) - x_0 \right\|_2^2 \tag{27}$$

The second alternative, shown in Equation 28, is a hybrid loss that combines the simplicity of L_{simple} with the full variational lower bound loss $L_{vlbfull}$ from (Nichol & Dhariwal, 2021). This hybrid approach aims to leverage the benefits of both simplified and complete loss functions to improve model performance across various conditions.

$$L_{hybrid} = L_{simple} + \lambda L_{vlbfull} \tag{28}$$

where $L_{vlbfull} = L_0 + L_{t-1} + L_T$.

5 Models

5.1 Encoder

The encoder (used in the forward process, see Figure 1) consists of a single layer of 256 convolutional filters with a kernel size of 16 samples and a stride of 8 samples. This configuration is chosen to capture sufficient temporal and spectral information from the input speech. The encoder generates a representation $x_0 \in \mathbb{R}^{F \times T'}$, where F is the feature dimension and T' is the time axis.

$$x_0 = ReLU(\text{conv1d}(x))$$

The latent variables $x_t \in \mathbb{R}^{F \times T'}$ are then generated from x_0 during the forward diffusion process. To reconstruct the original signal, we use a transposed convolutional layer at the end of the reverse process, which has the same stride and kernel size as the encoder to ensure symmetry.

5.2 Data Recovery Model

For data recovery, an estimate $\hat{x}_{T-(n-1)\tau} \in \mathbb{R}^{F \times T'}$ is first normalized using layer normalization. This normalized estimate is then passed through a linear layer with a dimension of F.

Next, the output is chunked into segments of size s along the T' axis with a 50% overlap to better capture temporal dependencies. The chunked output $\hat{x}'_{T-(n-1)\tau} \in \mathbb{R}^{F \times s \times V}$, where V represents the total number of chunks, is passed through a neural network layer (see Figure 3).

Each layer of this network is a transformer with 8 attention heads and a 768-dimensional feedforward network. The transformer processes the input $\hat{x}'_{T-(n-1)\tau} \in \mathbb{R}^{F \times s \times V}$ and outputs $\hat{x}'_{T-n\tau} \in \mathbb{R}^{F \times s \times V}$, which is passed

to the next layer $T-(n+1)\tau$. After processing, the final estimate $\hat{x}_{T-n\tau} \in \mathbb{R}^{F\times T'}$ is obtained by merging the last two dimensions of $\hat{x}'_{T-n\tau} \in \mathbb{R}^{F\times s\times V}$.

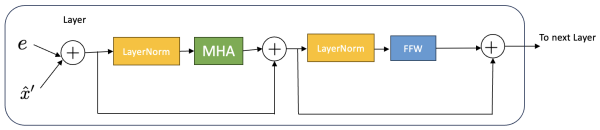

Figure 3: A single layer of the transformer model used for data recovery.

6 Evaluation

This section discusses the development and evaluation of the proposed technique, referred to as **UDPNet** (Unrolling Diffusion Process Network).

6.1 Dataset

To maintain consistency with current trends in the speech synthesis domain and allow for comparison with other existing tools, we used two popular datasets: LJSpeech for single-speaker evaluation and VCTK for multi-speaker evaluation. The LJSpeech dataset consists of 13,100 audio clips sampled at 22 kHz, with a total duration of approximately 24 hours. The clips are spoken by a single female speaker, and their lengths range from 1 to 10 seconds. For multi-speaker evaluation, we used the VCTK dataset, which is sampled at 48 kHz and includes 109 English speakers with various accents. We downsampled the VCTK dataset to 22 kHz for consistency.

Following (Chen et al., 2020), we used 12,764 utterances (approximately 23 hours of audio) from LJSpeech for training, and evaluated the model on a test set of 130 utterances. For multi-speaker evaluation, we used the data split from (Lam et al., 2022), where 100 speakers were used for training and 9 speakers were held out for evaluation.

Mel-spectrogram features were extracted from each audio sample, resulting in 128-dimensional feature vectors. Similar to (Chen et al., 2020), we used a 50-ms Hanning window, a 12.5-ms frame shift, and a 2048-point FFT with upper and lower frequency limits of 12 kHz and 20 Hz, respectively.

6.2 Training Parameters

The model was trained on a single NVIDIA V100 GPU using the Adam optimizer. We employed a cyclical learning rate (Smith, 2017) with a minimum learning rate of $1e-4$ and a maximum of $1e-1$. The batch size was set to 32, and the model was trained for 1 million steps.

For conditional audio generation, we used Mel-spectrograms extracted from ground truth audio as conditioning features during training. During testing, we used Mel-spectrograms generated by the Tacotron 2 model (Shen et al., 2018). To generate the FiLM parameters β and γ, we followed the upsampling block approach proposed in Chen et al. (2020), using these parameters to modulate the activations of the corresponding layer as described in Equation 23.

To weigh the contribution of each layer l to the overall loss L_{t-1} (see Equation 20), we initialized $\lambda = 0.001$ for the first layer and incremented it by 0.001 for each subsequent layer. This ensured that higher layers contributed more to the loss than lower layers.

6.3 Baseline Models

We compared UDPNet with other state-of-the-art vocoders, selecting models with publicly available samples for human evaluation. The baseline models used include:

- WaveNet (Oord et al., 2016) [1]
- WaveGlow (Prenger et al., 2018) [2]
- MelGAN (Kumar et al., 2019) [3]
- HiFi-GAN (Kong et al., 2020a) [4]
- WaveGrad (Chen et al., 2020) [5]
- DiffWave (Kong et al., 2020b) [6]
- BDDM (Lam et al., 2022) [7]
- FastDiff (Huang et al., 2022a) [8]

For each baseline, we selected the best-performing version based on the Mean Opinion Score (MOS) reported in the literature. **Metrics**: For subjective evaluation, we used the Mean Opinion Score (MOS) metric, a standard in speech synthesis, to assess the performance of the proposed model in comparison to baseline tools. Speech samples generated by each model were collected, along with randomly selected samples from the original audio. Each sample was presented to human evaluators one at a time for them to rate the quality and naturalness of the speech on a 5-point MOS scale: 1 (Bad), 2 (Poor), 3 (Fair), 4 (Good), and 5 (Excellent), with increments of 0.5. Each evaluator rated 10 samples. Human evaluators were recruited via Amazon Mechanical Turk, required to wear headphones, and were native English speakers.

For objective evaluation, we employed several deep learning-based MOS prediction tools, including SSL-MOS [9] (Cooper et al., 2022), MOSA-Net [10] (Zezario et al., 2022), and LDNet [11] (Huang et al., 2022c). These tools were chosen because they are adopted as baseline MOS prediction tools in the VoiceMOS challenge (Huang et al., 2022b).

SSL-MOS: A Wav2Vec-based model fine-tuned for MOS prediction by adding a linear layer on top of the Wav2Vec model. MOSA-Net: This model employs cross-domain features, such as time-frequency spectrograms, complex spectrograms, raw waveforms, and features extracted from self-supervised learning (SSL) speech models to predict MOS. LDNet: This MOS prediction tool estimates the listener-wise perceived quality of speech, where samples must be annotated with listener identity. The final MOS score is averaged across all listeners.

We also used the F_0 Frame Error (FFE) metric to measure the proportion of the generated speech whose pitch differs from the ground truth. Ground truth speech was required to compute FFE, which was calculated using Mel-spectrograms extracted from ground truth speeches.

Model Configurations: To train the model, we experimented with different numbers of steps in the forward process while keeping the reverse steps constant at 8. We experimented with forward steps (fsteps) of 1200, 960, 720, and 240, corresponding to skip parameters $\tau = \{150, 120, 90, 30\}$, respectively. The model accepted a 0.3-second input of audio.

For the forward process, the noise schedule α_i increased linearly from α_1 to α_N, defined as $Linear(\alpha_1, \alpha_N, N)$. For example, for 1200 forward steps, the schedule was defined as $Linear(1 \times 10^{-4}, 0.005, 1200)$.

[1] https://github.com/r9y9/wavenet_vocoder
[2] https://github.com/NVIDIA/waveglow
[3] https://github.com/descriptinc/melgan-neurips
[4] https://github.com/jik876/hifi-gan
[5] https://github.com/tencent-ailab/bddm
[6] https://github.com/tencent-ailab/bddm
[7] https://github.com/tencent-ailab/bddm
[8] https://FastDiff.github.io/
[9] https://github.com/nii-yamagishilab/mos-finetune-ssl
[10] https://github.com/dhimasryan/MOSA-Net-Cross-Domain
[11] https://github.com/unilight/LDNet

6.4 Results

6.4.1 Gradient noise scales of the objective functions

We began our evaluation by measuring the noise levels of the three proposed objective functions: L_{vlb}, L_{simple}, and L_{hybrid}. To do this, we evaluated their gradient noise scales following the methodology described in (McCandlish et al., 2018) and (Nichol & Dhariwal, 2021). The models were trained on the LJSpeech dataset for single-speaker evaluation, using 1200 forward steps and 8 reverse steps, as this configuration offered a good balance between computational efficiency and model performance.

Figure 4 shows the gradient noise scales for each objective function. Among the three, L_{hybrid} exhibited the highest noise levels, which we attribute to the inclusion of L_T in the objective function. As noted by (Nichol & Dhariwal, 2021), the term L_T, which occurs at the end of the forward noising process, introduces significant noise during uniform timestep sampling.

For the remaining sections, we report results based on using L_{vlb} as the primary training objective, as it demonstrated more stable gradient behavior compared to L_{hybrid} and L_{simple}.

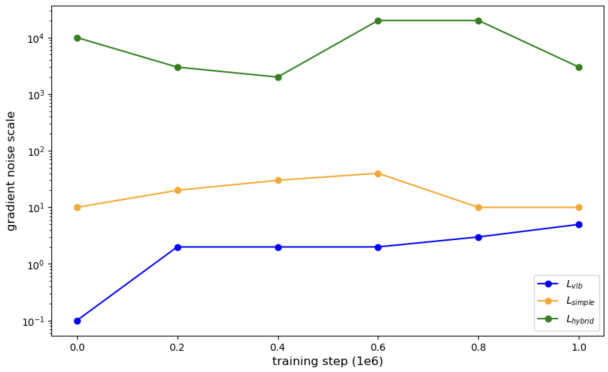

Figure 4: Gradient noise scales for the L_{vlb}, L_{hybrid}, and L_{simple} objectives on the LJSpeech dataset.

6.4.2 Single speaker

6.5 Results

6.5.1 Gradient noise scales of the objective functions

6.6 Results

For conditional speech generation on the single-speaker dataset, we evaluated the proposed technique, UDPNet, using both subjective and objective metrics. Table 1 presents the subjective Mean Opinion Score (MOS) alongside the objective MOS results obtained from SSL-MOS, MOSA-Net, and LDNet. The table also includes the real-time factor (RTF), which measures the speed of speech generation.

The best-performing configuration of UDPNet, with 1200 forward steps and 8 reverse steps (1200, 8), achieved a subjective MOS of 4.49, which is only 0.23 points lower than the ground truth score of 4.72. This demonstrates that UDPNet is capable of producing high-quality, natural-sounding speech that closely approximates the original audio. Additionally, UDPNet (1200, 8) outperformed all other models in the objective MOS metrics across SSL-MOS, MOSA-Net, and LDNet, confirming its superiority in generating speech with high fidelity and minimal distortion.

In terms of speech generation speed, all UDPNet configurations demonstrated competitive performance, with smaller step sizes (τ) leading to faster generation times. We hypothesize that as the step size decreases, each neural network layer has a smaller amount of noise to remove, which reduces the computational load and accelerates the speech generation process. This is evident from the lowest RTF of 0.00182 achieved by the configuration with 240 forward steps.

We also observed that increasing the number of forward steps resulted in higher-quality audio, as reflected in both the subjective and objective MOS scores. This improvement is likely due to the more gradual denoising process, which allows the model to better preserve the fine details of the speech signal. Furthermore, the use of latent variables x_t as targets in the objective function L_{vlb}, instead of the original input x_0, helps to reduce prediction errors. Large prediction errors have been shown to contribute to speech distortion (Zhou et al., 2023), and the reduction of these errors in UDPNet likely explains its superior performance.

Finally, in terms of F0 Frame Error (FFE), UDPNet (1200, 8) achieved the lowest error rate at 2.3%, further demonstrating its ability to maintain pitch accuracy and reduce speech distortion compared to other state-of-the-art models.

Table 1: Evaluation results of the conditioned version of the proposed method compared to state-of-the-art tools on the evaluation metrics using a single-speaker dataset.

LJSpeech Test Dataset						
Model	**MOS(↑)**	**SSL-MOS(↑)**	**MOSANet(↑)**	**LDNet(↑)**	**FFE(↓)**	**RTF (↓)**
Ground Truth	4.72±0.15	4.56	4.51	4.67	-	-
BDDM (12 steps)	4.38±0.15	4.23	4.17	4.42	3.6%	0.543
DiffWave (200 steps)	4.43±0.13	4.31	4.28	4.36	2.6%	5.9
WaveGrad (1000 steps)	4.32±0.15	4.27	4.23	4.31	2.8%	38.2
HIFI-GAN	4.26±0.14	4.19	4.13	4.27	3.3%	0.0134
MelGAN	3.49±0.12	3.33	3.27	3.42	6.7%	0.00396
WaveGlow	3.17±0.14	3.12	3.09	3.14	7.3%	0.0198
WaveNet	3.61±0.15	3.51	3.47	3.54	6.3%	318.6
UDPNet (fsteps: 1200, rsteps: 8)	**4.49**±0.12	**4.43**	**4.35**	**4.44**	**2.3%**	0.0042
UDPNet (fsteps: 960, rsteps: 8)	4.33±0.15	4.283	4.23	4.31	3.7%	0.00371
UDPNet (fsteps: 720, rsteps: 8)	4.17±0.15	4.12	4.09	4.14	4.3%	0.002912
UDPNet (fsteps: 240, rsteps: 8)	4.09±0.13	4.05	4.01	4.05	4.7%	**0.00182**

6.6.1 Multi-Speaker

The results of the proposed technique on the multi-speaker VCTK dataset are presented in Table 2. UDPNet (1200, 8) demonstrated strong generalization to unseen speakers, achieving the highest subjective MOS score of 4.38, just 0.25 points below the ground truth. It also outperformed all other models in the objective MOS metrics, including SSL-MOS, MOSA-Net, and LDNet.

While UDPNet (1200, 8) performed best in almost all objective metrics, it was slightly outperformed by DiffWave in terms of F0 Frame Error (FFE), with a margin of just 0.1%. This small difference highlights the overall robustness of UDPNet, which strikes a balance between high speech quality and competitive generation speed.

Table 2: Evaluation results of the conditioned version of the proposed method compared to state-of-the-art tools on the evaluation metrics using the multi-speaker dataset (VCTK).

VCTK Test Dataset						
Model	**MOS(↑)**	**SSL-MOS(↑)**	**MOSANet(↑)**	**LDNet(↑)**	**FFE(↓)**	**RTF(↓)**
Ground Truth	4.63±0.05	4.57	4.69	4.65	-	-
BDDM (12 steps)	4.33±0.05	4.28	4.25	4.35	4.3%	0.543
DiffWave (200 steps)	**4.38**±0.03	4.41	4.32	4.33	**3.2%**	5.9
WaveGrad (1000 steps)	4.26±0.05	4.31	4.21	4.24	3.4%	38.2
HIFI-GAN	4.19±0.14	4.12	4.16	4.18	3.9%	0.0134
MelGAN	3.33±0.05	3.27	3.24	3.37	7.7%	0.00396
WaveGlow	3.13±0.05	3.12	3.16	3.09	8.2%	0.0198
WaveNet	3.53±0.05	3.43	3.45	3.46	7.2%	318.6
UDPNet (fsteps: 1200, rsteps: 8)	**4.38**±0.12	**4.43**	**4.36**	**4.40**	3.3%	0.0042
UDPNet (fsteps: 960, rsteps: 8)	4.28±0.05	4.23	4.25	4.29	4.2%	0.00371
UDPNet (fsteps: 720, rsteps: 8)	4.12±0.05	4.11	4.13	4.08	4.6%	0.002912
UDPNet (fsteps: 240, rsteps: 8)	4.04±0.03	4.01	3.91	4.01	5.2%	**0.00182**

6.6.2 Unconditional speech generation

Here, the model was trained using multi-speaker dataset. To generate a speech sample, we sample white noise at random and process it through the trained model without conditioning it on any acoustic features. The results for unconditional speech generation are shown in table 3. For short clips, UDPNet (fsteps:1200 rsteps 8) attains the MOS score of 3.11. Listening to the audio clips, we noticed a phenomenon where the clips begin by generating coherent sounding sentences, but the coherence drops with time. We will investigate the reason for this phenomenon in our future work. However, the model can generate clean sounding speeches, i.e., almost free of noise or artefacts.

Table 3: Results of the unconditioned proposed method on multi-speaker dataset.

VCTK test-dataset					
Model	**MOS(↑)**	**SSL-MOS(↑)**	**MOSANet(↑)**	**LDNet(↑)**	**RTF (↓)**
UDPNet(fsteps:1200 rsteps 8)	**3.11**± 0.12	**3.17**	**3.16**	**3.23**	0.0038
UDPNet(fsteps:960 rsteps 8)	3.04± 0.05	3.09	3.01	3.09	0.00351
UDPNet(fsteps:720 rsteps 8)	3.02 ± 0.05	3.07	3.01	3.06	0.002812
UDPNet(fsteps:240 rsteps 8)	2.98± 0.03	3.02	3.03	3.08	**0.00162**

7 Conclusion

In this paper, we introduced UDPNet, a novel approach for accelerating speech generation in diffusion models by leveraging the structure of neural network layers. By progressively recovering the data distribution from white noise, each neural network layer performs implicit denoising. Through the use of a skip parameter τ, we effectively map neural network layers to the forward diffusion process, reducing the number of recovery steps required and improving efficiency.

Our modified objective function allows the model to balance accuracy and speed, and we further enhanced conditional speech generation by incorporating Feature-wise Linear Modulation (FiLM) to integrate acoustic features into the denoising process. Through extensive evaluations on both single-speaker and multi-speaker datasets, UDPNet demonstrated the ability to produce high-quality speech samples while maintaining competitive generation speed.

While the results are promising, future work could explore the impact of increasing the number of forward steps, investigate the coherence degradation over time in unconditional speech generation, and apply UDPNet to additional speech tasks or other generative modeling domains. Overall, UDPNet offers a significant step forward in efficient and high-quality speech generation for diffusion-based models.

References

Nanxin Chen, Yu Zhang, Heiga Zen, Ron J Weiss, Mohammad Norouzi, and William Chan. Wavegrad: Estimating gradients for waveform generation. *arXiv preprint arXiv:2009.00713*, 2020.

Erica Cooper, Wen-Chin Huang, Tomoki Toda, and Junichi Yamagishi. Generalization ability of mos prediction networks. In *ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 8442–8446. IEEE, 2022.

Ian Goodfellow. Nips 2016 tutorial: Generative adversarial networks. *arXiv preprint arXiv:1701.00160*, 2016.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020.

Po-chun Hsu and Hung-yi Lee. Wg-wavenet: Real-time high-fidelity speech synthesis without gpu. *arXiv preprint arXiv:2005.07412*, 2020.

Rongjie Huang, Max WY Lam, Jun Wang, Dan Su, Dong Yu, Yi Ren, and Zhou Zhao. Fastdiff: A fast conditional diffusion model for high-quality speech synthesis. *arXiv preprint arXiv:2204.09934*, 2022a.

Wen-Chin Huang, Erica Cooper, Yu Tsao, Hsin-Min Wang, Tomoki Toda, and Junichi Yamagishi. The voicemos challenge 2022. *arXiv preprint arXiv:2203.11389*, 2022b.

Wen-Chin Huang, Erica Cooper, Junichi Yamagishi, and Tomoki Toda. Ldnet: Unified listener dependent modeling in mos prediction for synthetic speech. In *ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 896–900. IEEE, 2022c.

Nal Kalchbrenner, Erich Elsen, Karen Simonyan, Seb Noury, Norman Casagrande, Edward Lockhart, Florian Stimberg, Aaron Oord, Sander Dieleman, and Koray Kavukcuoglu. Efficient neural audio synthesis. In *International Conference on Machine Learning*, pp. 2410–2419. PMLR, 2018.

Hyeongju Kim, Hyeonseung Lee, Woo Hyun Kang, Sung Jun Cheon, Byoung Jin Choi, and Nam Soo Kim. Wavenode: A continuous normalizing flow for speech synthesis. *arXiv preprint arXiv:2006.04598*, 2020.

Diederik Kingma, Tim Salimans, Ben Poole, and Jonathan Ho. Variational diffusion models. *Advances in neural information processing systems*, 34:21696–21707, 2021.

Jungil Kong, Jaehyeon Kim, and Jaekyoung Bae. Hifi-gan: Generative adversarial networks for efficient and high fidelity speech synthesis. *Advances in Neural Information Processing Systems*, 33:17022–17033, 2020a.

Zhifeng Kong, Wei Ping, Jiaji Huang, Kexin Zhao, and Bryan Catanzaro. Diffwave: A versatile diffusion model for audio synthesis. *arXiv preprint arXiv:2009.09761*, 2020b.

Kundan Kumar, Rithesh Kumar, Thibault De Boissiere, Lucas Gestin, Wei Zhen Teoh, Jose Sotelo, Alexandre De Brebisson, Yoshua Bengio, and Aaron C Courville. Melgan: Generative adversarial networks for conditional waveform synthesis. *Advances in neural information processing systems*, 32, 2019.

Max WY Lam, Jun Wang, Dan Su, and Dong Yu. Bddm: Bilateral denoising diffusion models for fast and high-quality speech synthesis. *arXiv preprint arXiv:2203.13508*, 2022.

Sang-gil Lee, Heeseung Kim, Chaehun Shin, Xu Tan, Chang Liu, Qi Meng, Tao Qin, Wei Chen, Sungroh Yoon, and Tie-Yan Liu. Priorgrad: Improving conditional denoising diffusion models with data-dependent adaptive prior. *arXiv preprint arXiv:2106.06406*, 2021.

Calvin Luo. Understanding diffusion models: A unified perspective. *arXiv preprint arXiv:2208.11970*, 2022.

Sam McCandlish, Jared Kaplan, Dario Amodei, and OpenAI Dota Team. An empirical model of large-batch training. *arXiv preprint arXiv:1812.06162*, 2018.

Soroush Mehri, Kundan Kumar, Ishaan Gulrajani, Rithesh Kumar, Shubham Jain, Jose Sotelo, Aaron Courville, and Yoshua Bengio. Samplernn: An unconditional end-to-end neural audio generation model. *arXiv preprint arXiv:1612.07837*, 2016.

Lars Mescheder, Andreas Geiger, and Sebastian Nowozin. Which training methods for gans do actually converge? In *International conference on machine learning*, pp. 3481–3490. PMLR, 2018.

Alexander Quinn Nichol and Prafulla Dhariwal. Improved denoising diffusion probabilistic models. In *International Conference on Machine Learning*, pp. 8162–8171. PMLR, 2021.

Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

Tom Le Paine, Pooya Khorrami, Shiyu Chang, Yang Zhang, Prajit Ramachandran, Mark A Hasegawa-Johnson, and Thomas S Huang. Fast wavenet generation algorithm. *arXiv preprint arXiv:1611.09482*, 2016.

Ethan Perez, Florian Strub, Harm De Vries, Vincent Dumoulin, and Aaron Courville. Film: Visual reasoning with a general conditioning layer. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

R Prenger, R Valle, and B Catanzaro. A flow-based generative network for speech synthesis, 2018.

Ryan Prenger, Rafael Valle, and Bryan Catanzaro. Waveglow: A flow-based generative network for speech synthesis. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 3617–3621. IEEE, 2019.

Ali Razavi, Aaron Van den Oord, and Oriol Vinyals. Generating diverse high-fidelity images with vq-vae-2. *Advances in neural information processing systems*, 32, 2019.

Danilo Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *International conference on machine learning*, pp. 1530–1538. PMLR, 2015.

Jonathan Shen, Ruoming Pang, Ron J Weiss, Mike Schuster, Navdeep Jaitly, Zongheng Yang, Zhifeng Chen, Yu Zhang, Yuxuan Wang, Rj Skerrv-Ryan, et al. Natural tts synthesis by conditioning wavenet on mel spectrogram predictions. In *2018 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pp. 4779–4783. IEEE, 2018.

Leslie N Smith. Cyclical learning rates for training neural networks. In *2017 IEEE winter conference on applications of computer vision (WACV)*, pp. 464–472. IEEE, 2017.

Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *International conference on machine learning*, pp. 2256–2265. PMLR, 2015.

Xu Tan, Tao Qin, Frank Soong, and Tie-Yan Liu. A survey on neural speech synthesis. *arXiv preprint arXiv:2106.15561*, 2021.

Jean-Marc Valin and Jan Skoglund. Lpcnet: Improving neural speech synthesis through linear prediction. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 5891–5895. IEEE, 2019.

Ryandhimas E Zezario, Szu-Wei Fu, Fei Chen, Chiou-Shann Fuh, Hsin-Min Wang, and Yu Tsao. Deep learning-based non-intrusive multi-objective speech assessment model with cross-domain features. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 31:54–70, 2022.

Rui Zhou, Wenye Zhu, and Xiaofei Li. Speech dereverberation with a reverberation time shortening target. In *ICASSP 2023-2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 1–5. IEEE, 2023.

Impact Statement

"This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

A Appendix.

B Derivation of Equation 26

We start by defining the loss function L_{t-1} as the Kullback-Leibler (KL) divergence between the true posterior distribution $q(x_{t-1}|x_t, x_0)$ and the learned model distribution $p_\theta(\hat{x}_{t-1}|\hat{x}_t)$:

$$L_{t-1} = \arg\min_\theta \mathbb{E}_{t\sim U(2,T)} D_{\mathrm{KL}}\left(q(x_{t-1}|x_t, x_0) \| p_\theta(\hat{x}_{t-1}|\hat{x}_t)\right) \tag{29}$$

Next, assuming that both $q(x_{t-1}|x_t, x_0)$ and $p_\theta(\hat{x}_{t-1}|\hat{x}_t)$ are Gaussian distributions, we rewrite the KL divergence as:

$$L_{t-1} = \arg\min_\theta \mathbb{E}_{t\sim U(2,T)} D_{\mathrm{KL}}\left(\mathcal{N}(x_{t-1}; \mu_q(t), \Sigma_q(t)) \| \mathcal{N}(\hat{x}_{t-1}; \hat{\mu}_\theta, \Sigma_q(t))\right) \tag{30}$$

where the covariance matrix $\Sigma_q(t)$ and the means $\mu_q(t)$ and $\hat{\mu}_\theta$ are defined as:

$$\Sigma_q(t) = \frac{(1-\alpha_t)(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t} I,$$

$$\mu_q(t) = \frac{\sqrt{\alpha}(1-\bar{\alpha}_{t-1})x_t + \sqrt{\bar{\alpha}_{t-1}}(1-\alpha_t)x_0}{1-\bar{\alpha}_t},$$

$$\hat{\mu}_\theta = \frac{\sqrt{\alpha}(1-\bar{\alpha}_{t-1})x_\theta(\hat{x}_{t+1}, t) + \sqrt{\bar{\alpha}_{t-1}}(1-\alpha_t)x_0}{1-\bar{\alpha}_t}.$$

The KL divergence between two Gaussians, with identical covariance matrices $\Sigma_q(t)$, simplifies as follows:

$$\begin{gathered} D_{\mathrm{KL}}\left(\mathcal{N}\left(x_{t-1}; \mu_q(t), \Sigma_q(t)\right) \| \mathcal{N}\left(\hat{x}_{t-1}; \hat{\mu}_\theta, \Sigma_q(t)\right)\right) = \\ \frac{1}{2}\left[\log\frac{|\Sigma_q(t)|}{|\Sigma_q(t)|} - d + \mathrm{tr}\left(\Sigma_q(t)^{-1}\Sigma_q(t)\right) + \left(\hat{\mu}_\theta - \mu_q(t)\right)^T \Sigma_q(t)^{-1}\left(\hat{\mu}_\theta - \mu_q(t)\right)\right] \end{gathered} \tag{31}$$

Since the covariance matrices are identical, the trace term and the log determinant term cancel out, reducing the KL divergence to:

$$D_{\mathrm{KL}}\left(\mathcal{N}\left(x_{t-1}; \mu_q(t), \Sigma_q(t)\right) \| \mathcal{N}\left(\hat{x}_{t-1}; \hat{\mu}_\theta, \Sigma_q(t)\right)\right) = \frac{1}{2}\left[\left(\hat{\mu}_\theta - \mu_q(t)\right)^T \Sigma_q(t)^{-1}\left(\hat{\mu}_\theta - \mu_q(t)\right)\right] \tag{32}$$

Substituting $\hat{\mu}_\theta$ and $\mu_q(t)$, we get:

$$\frac{1}{2\Sigma_q(t)}\left[\left(\frac{\sqrt{\alpha}(1-\bar{\alpha}_{t-1})x_\theta(\hat{x}_{t+1}, t) + \sqrt{\bar{\alpha}_{t-1}}(1-\alpha_t)x_0}{1-\bar{\alpha}_t} - \frac{\sqrt{\alpha}(1-\bar{\alpha}_{t-1})x_t + \sqrt{\bar{\alpha}_{t-1}}(1-\alpha_t)x_0}{1-\bar{\alpha}_t}\right)^2\right] \tag{33}$$

Simplifying this expression:

$$\frac{\sqrt{\alpha}(1-\bar{\alpha}_{t-1})}{2\Sigma_q(t)(1-\bar{\alpha}_t)}\left[\|x_\theta(\hat{x}_{t+1}, t) - x_t\|_2^2\right] \tag{34}$$

Thus, the final expression for the loss term L_{t-1} is:

$$L_{t-1} = \frac{\sqrt{\alpha}(1-\bar{\alpha}_{t-1})}{2\Sigma_q(t)(1-\bar{\alpha}_t)} \left[\| x_\theta(\hat{x}_{t+1}, t) - x_t \|_2^2 \right] \tag{35}$$