Speech Synthesis By Unrolling Diffusion Process using Neural Network Layers

Peter Ochieng

Abstract

This work introduces UDPNet, a novel architecture designed to accelerate the reverse diffusion process in speech synthesis. Unlike traditional diffusion models that rely on timestep embeddings and shared network parameters, UDPNet unrolls the reverse diffusion process directly into the network architecture, with successive layers corresponding to equally spaced steps in the diffusion schedule. Each layer progressively refines the noisy input, culminating in a high-fidelity estimation of the original data, x_0. Additionally, we redefine the learning target by predicting latent variables instead of the conventional x_0 or noise ϵ_0. This shift addresses the common issue of large prediction errors in early denoising stages, effectively reducing speech distortion. Extensive evaluations on single- and multi-speaker datasets demonstrate that UDPNet consistently outperforms state-of-the-art methods in both quality and efficiency, while generalizing effectively to unseen speech. These results position UDPNet as a robust solution for real-time speech synthesis applications. Sample audio is available at `https://onexpeters.github.io/UDPNet/`.

Introduction

Diffusion Probability Models (DPMs) (Sohl-Dickstein et al., 2015) have gained significant 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 effectively. These models rely on two fundamental processes: the **forward process**, where Gaussian noise is progressively added to the data until it resembles white noise, and the **reverse process**, where the model learns to recover the original data distribution through denoising.

While DPMs excel in generating high-quality outputs, they typically require a large number of diffusion steps during training, leading to prohibitively slow inference times. 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 to shorten the noise schedule. However, even with these optimizations, sampling remains computationally expensive.

To address this, we propose a simple yet effective strategy that directly unrolls the reverse diffusion process into the architecture of a neural network. Rather than applying a shared network across all timesteps using explicit timestep embeddings (as in traditional DPMs), we map each layer of the network to a specific set of diffusion steps. Crucially, this does not introduce additional parameters, since each layer shares a common set of weights and does not independently learn new parameters for each timestep. This design leverages depth-wise allocation instead of timestep-conditioned embeddings, effectively distributing denoising refinement across layers while maintaining parameter efficiency. The number of layers N is determined by the skip parameter τ, as follows:

$$N = \frac{T}{\tau},$$

where T is the total number of forward diffusion steps. Each layer progressively refines the denoised signal, simplifying the reverse process and significantly reducing computational costs.

Beyond architectural efficiency, we further stabilize training by modifying the learning target. Instead of predicting the original input x_0 or the added noise ϵ_0, we train the model to predict latent variables. Directly

predicting x_0 or ϵ_0 often leads to large errors in the early reverse steps, introducing artifacts and distortions in synthesized speech (Zhou et al., 2023). Our proposed latent-space formulation provides a more structured representation, reducing early-stage errors and improving convergence.

Through extensive evaluations, we demonstrate that our approach not only accelerates speech generation but also generalizes effectively to unseen speech. By integrating layer-wise diffusion unrolling with a latent-variable learning target, we achieve state-of-the-art synthesis quality while significantly improving efficiency, making our method well-suited for real-time applications such as AI voice assistants and voice cloning. This makes our method particularly suitable for real-time applications such as AI-powered voice assistants and voice cloning.

1 Background

1.1 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 T diffusion steps.

The encoder models the hidden variables x_t as linear Gaussian models with mean and standard deviation centered around its previous hierarchical latent x_{t-1}. The mean and variance can be modeled as hyperparameters (Ho et al., 2020) or as learnable variables (Nichol & Dhariwal, 2021; Kingma et al., 2021). The Gaussian encoder's mean and variance are parameterized as:

$$u_t(x_t) = \sqrt{\alpha_t} x_{t-1}, \quad \Sigma_q(x_t) = (1 - \alpha_t)I \tag{2}$$

Thus, the encoder can be expressed as:

$$q(x_t|x_{t-1}) = \mathcal{N}(x_t; \sqrt{\alpha_t} x_{t-1}, (1 - \alpha_t)I) \tag{3}$$

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 from x_0 as:

$$x_t = \sqrt{\bar{\alpha}_t} x_0 + \sqrt{(1 - \bar{\alpha}_t)} \epsilon_0 \tag{4}$$

where $\bar{\alpha}_t = \prod_{t=1}^{t} \alpha_t$ and $\epsilon_0 \sim \mathcal{N}(0, I)$. Hence,

$$q(x_t|x_0) = \mathcal{N}(x_t; \sqrt{\bar{\alpha}_t} x_0, (1 - \bar{\alpha}_t)I). \tag{5}$$

The reverse process, which seeks to recover the data distribution from the white noise $p(x_T)$, is modeled as:

$$p_\theta(x_{0:T}) = p(x_T) \prod_{i=1}^{T} p_\theta(x_{t-1}|x_t) \tag{6}$$

where $p(x_T) = \mathcal{N}(x_T; 0, I)$. The goal of DPM is to model the reverse process $p_\theta(x_{t-1}|x_t)$ so that it can be used to generate new data samples.

1.2 Optimization of DPM

After the DPM has been optimized, a sampling procedure entails: 1. Sampling Gaussian noise from $p(x_T)$. 2. Iteratively running the denoising transitions $p_\theta(x_{t-1}|x_t)$ for T steps to generate x_0.

To optimize DPM, the evidence lower bound (ELBO) is used:

$$\begin{aligned}\log p(x) =& E_{q(x_0)}[D_{KL}(q(x_T|x_0)||p(x_T))+ \\ & \sum_{t=2}^{T} E_{q(x_t|x_0)}[D_{KL}(q(x_{t-1}|x_t,x_0)||p_\theta(x_{t-1}|x_t))]- \\ & E_{q(x_1|x_0)}[\log p_\theta(x_0|x_1)].\end{aligned} \tag{7}$$

In Eq. 7, 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:

$$\begin{aligned}q(x_{t-1}|x_t,x_0) =& \mathcal{N}\Bigg(\frac{\sqrt{\alpha}(1-\bar{\alpha}_{t-1})x_t+\sqrt{\bar{\alpha}}_{t-1}(1-\alpha_t)x_0}{(1-\bar{\alpha}_t)}, \\ & \frac{(1-\alpha_t)(1-\bar{\alpha}_{t-1})}{(1-\bar{\alpha_t})}I\Bigg).\end{aligned} \tag{8}$$

To ensure $p_\theta(x_{t-1}|x_t)$ matches $q(x_{t-1}|x_t,x_0)$, it is modeled with the same variance $\Sigma_q(t)$, given by:

$$\Sigma_q(t) = \frac{(1-\alpha_t)(1-\bar{\alpha}_{t-1})}{(1-\bar{\alpha_t})}I. \tag{9}$$

The mean of $p_\theta(x_{t-1}|x_t)$ is parameterized as:

$$u_\theta(x_t,t) = \frac{\sqrt{\alpha}(1-\bar{\alpha}_{t-1})x_t+\sqrt{\bar{\alpha}_{t-1}}(1-\alpha_t)\hat{x}_\theta(x_t,t)}{(1-\bar{\alpha_t})}. \tag{10}$$

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

$$p_\theta(x_{t-1}|x_t) = \mathcal{N}(\mu_\theta, \Sigma_q(t)). \tag{11}$$

Optimizing the KL divergence between $q(x_{t-1}|x_t,x_0)$ and $p_\theta(x_{t-1}|x_t)$ can be formulated as:

$$L_{t-1} = \arg\min_\theta E_{t\sim U(2,T)} D_{KL}(q(x_{t-1}|x_t,x_0)||p_\theta(x_{t-1}|x_t)). \tag{12}$$

Using standard derivations (Luo, 2022), we obtain:

$$L_{t-1} = \arg\min_\theta E_{t\sim U(2,T)} \|\hat{x}_\theta(x_t,t)-x_0\|_2^2. \tag{13}$$

By rearranging Eq. 4, we obtain:

$$x_0 = \frac{x_t-\sqrt{1-\bar{\alpha}_t}\epsilon_0}{\sqrt{\bar{\alpha_t}}}. \tag{14}$$

Thus, an equivalent optimization using a noise predictor $\hat{\epsilon}_\theta(x_t, t)$ is:

$$L_{t-1} = \arg\min_\theta E_{t\sim U(2,T)} \|\hat{\epsilon}_\theta(x_t, t) - \epsilon_0\|_2^2. \tag{15}$$

Work in (Ho et al., 2020) shows that L_{t-1} optimizes the ELBO.

2 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).

3 Speech synthesis by Unrolling diffusion process using Neural network layers

3.1 Unconditional speech generation

3.1.1 Forward Process

During the forward process, a raw audio waveform x is encoded by a pre-trained encoder into its latent representation $x_0 \in \mathbb{R}^{f\times h}$, where f denotes the number of frames and h is the hidden dimension size. This representation serves as the foundation for generating latent variables x_t through the forward diffusion process, as described in Equation 4, for $1 \leq t \leq T$.

To facilitate the reconstruction process during the reverse diffusion, a pre-trained discrete codebook of size K with dimension h is employed. The codebook, denoted as $\mathcal{Z} = \{z_k\}_{k=1}^K \in \mathbb{R}^h$, maps each row of x_0 to the closest entry in the codebook. This mapping is determined by minimizing the squared Euclidean distance, as shown in Equation 16:

$$z_q = \left(\arg\min_{z_k \in \mathcal{Z}} \|x_0^i - z_k\|_2^2 \ \forall i \in f \right) \in \mathbb{R}^{f\times h}. \tag{16}$$

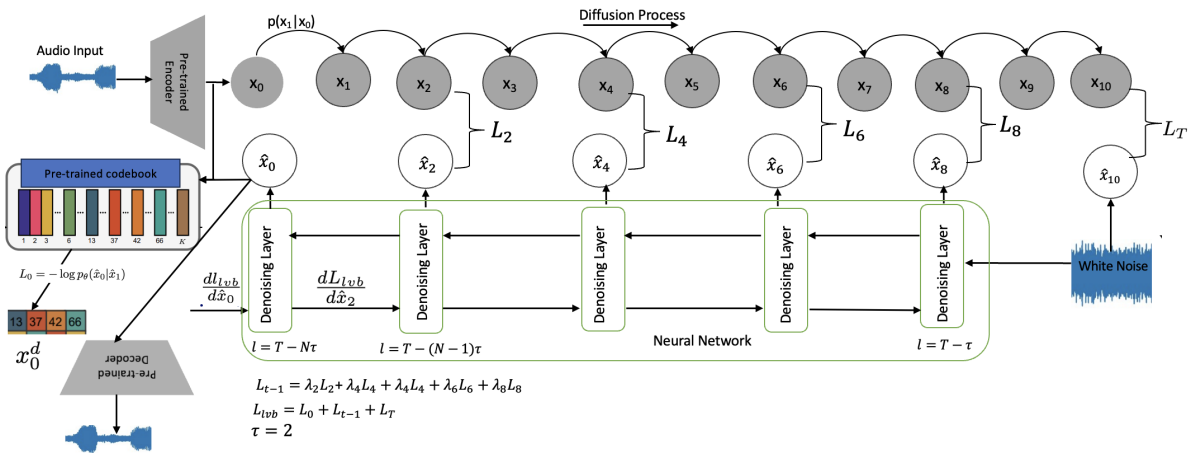

Figure 1: An overview of the unconditioned audio generation. An input audio is processed by a pre-trained model to generate x_0. x_0 is then processed by forward process to generate latent variables x_t. In the reverse process, white noise x_T is passed through the first layer of the neural network and 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.

Here, z_q represents the quantized latent representation, where each row x_0^i is replaced by the nearest codebook vector z_k. These discrete indices, x_0^d, correspond to the assigned codebook entries for each row of x_0. The stored indices play a critical role in the reverse diffusion process, enabling the recovery of the original signal from the latent representation (see Section **??**).

3.1.2 Reverse process

The ELBO (Equation 7) 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:

$$
L_{vlbfull} = L_0 + \sum_{t=1}^{T-1} L_{t-1} + L_T \tag{17}
$$

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 in our **layer-based recovery technique**, we introduce **deterministic approximations** $\hat{x}_{t-1}$ and $\hat{x}_t$ in place of the actual stochastic variables. Unlike traditional diffusion models that explicitly **sample Gaussian noise at each reverse step**, our approach **implicitly models noise transitions** via the hierarchical structure of the neural network. This removes redundant noise sampling while **preserving the core denoising dynamics**.

The parameterized version of L_{t-1} is given by:

$$L_{t-1} = \sum_{t=2}^{T} \mathbb{E}_{q(x_t|x_0)} \left[\mathrm{D_{KL}} \left(q(x_{t-1}|x_t, x_0) \| p_\theta(\hat{x}_{t-1}|\hat{x}_t) \right) \right] \tag{18}$$

This approximation significantly **reduces computational complexity** by distributing the denoising process across network layers rather than requiring per-step noise resampling. It provides three key advantages:

- **Improved Stability** – Direct estimation of $\hat{x}_{t-1}$ minimizes stochasticity, leading to smoother training dynamics.
- **Faster Inference** – Eliminating explicit noise sampling reduces redundant computations, accelerating generation speed.
- **Maintained Accuracy** – The hierarchical structure ensures that $\hat{x}_{t-1}$ and $\hat{x}_t$ retain sufficient information for precise reconstruction.

A direct comparison with the conventional **explicit noise-sampling approach** is provided in **Appendix X**, where we analyze trade-offs in **computation cost, inference speed, and output quality**.

To ensure robustness across different noise levels and timesteps, L_{t-1} is optimized via **stochastic timestep sampling**, where random timesteps are drawn from a uniform distribution:

$$L_{t-1} = \arg\min_\theta \mathbb{E}_{t\sim U(2,T)} \mathrm{D_{KL}} \left(q(x_{t-1}|x_t, x_0) \| p_\theta(\hat{x}_{t-1}|\hat{x}_t) \right) \tag{19}$$

Thus, our approach **optimally balances computational efficiency with denoising quality**, ensuring that learned approximations effectively reconstruct the target distribution:

$$L_{t-1} = \arg\min_\theta \mathbb{E}_{t\sim U(2,T)} \mathrm{D_{KL}} \left(\mathcal{N}(x_{t-1}; \mu_q(t), \Sigma_q(t)) \| \mathcal{N}(\hat{x}_{t-1}; \hat{\mu}_\theta, \Sigma_q(t)) \right) \tag{20}$$

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:

$$L_{t-1} = \arg\min_\theta \mathbb{E}_{t\sim U(1,T-1)} \frac{1}{2\Sigma_q(t)} \left\| \frac{\sqrt{\alpha}(1-\bar{\alpha}_{t-1})x_\theta(\hat{x}_{t+1}, t) + \sqrt{\bar{\alpha}}_{t-1}(1-\alpha_t)x_0}{1-\bar{\alpha}_t} - \frac{\sqrt{\alpha}(1-\bar{\alpha}_{t-1})x_t + \sqrt{\bar{\alpha}}_{t-1}(1-\alpha_t)x_0}{1-\bar{\alpha}_t} \right\|_2^2. \tag{21}$$

Equation 21 can be simplified as (see Appendix A for the complete derivation):

$$L_{t-1} = \arg\min_{\theta} \mathbb{E}_{t\sim U(1,T-1)} \frac{\sqrt{\alpha}(1-\bar{\alpha}_{t-1})}{2\Sigma_q(t)(1-\bar{\alpha}_t)} \left\|\hat{x}_\theta(\hat{x}_{t+1}, t) - x_t\right\|_2^2 \tag{22}$$

Optimizing L_{t-1} involves training a neural network $\hat{x}_\theta(\hat{x}_{t+1}, t)$ to predict x_t given the estimated variable $\hat{x}_{t+1}$ and the timestep t. This differs from the loss in Equation 13, where the network $\hat{x}_\theta(x_t, t)$ is conditioned on the noisy input x_t to predict the original noiseless input x_0.

To estimate a latent variable x_t using Equation 22, we map each timestep to a layer of a single neural network. For a network with N layers, this mapping creates an effective equivalence to N neural networks. When $N = T - 1$, each timestep $t \in [1, T-1]$ in the forward process corresponds to one neural network layer.

To accelerate the reverse diffusion process, we introduce a timestep skip parameter $\tau > 1$, reducing the number of layers to $N = \frac{T}{\tau}$. This allows the data distribution to be recovered in N steps, significantly fewer than T, speeding up data recovery.

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

In this sequential setup, the timesteps t are implicitly encoded by the neural network layers, removing the need for explicit conditioning on t. Thus, Equation 22 is implemented as:

$$L_{t-1} = \sum_{t=T-\tau}^{T-(N-1)\tau} \lambda_t \left\|\hat{x}_\theta^{l=t}(\hat{x}_{t+\tau}) - x_t\right\|_2^2 \tag{23}$$

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 15) is used to estimate the variational lower bound $L_{vlbfull}$ (Equation 17). 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:

$$L_0 = -\log p_\theta(\hat{x}_0 \mid \hat{x}_{T-(N-1)\tau}) \tag{24}$$

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:

$$L_{vlbfull} = L_0 + L_{t-1} + L_T \tag{25}$$

During implementation, we ignore L_T and compute the truncated loss as:

$$L_{vlb} = L_0 + L_{t-1}.$$

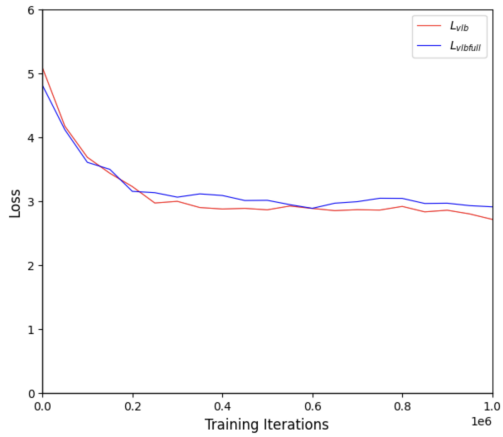

Figure 2: Learning curves comparing the full objective $L_{vlbfull}$ and L_{vlb} on the LJSpeech dataset.

While L_T is theoretically constant and does not depend on the model parameters, its inclusion introduces practical issues during training. Specifically, the KL divergence term $L_T = \mathrm{D}_{\mathrm{KL}}(q(x_T|x_0)\|p(x_T))$ reflects the mismatch between the prior $p(x_T) \sim \mathcal{N}(0, I)$ and the distribution $q(x_T|x_0)$, which depends on the forward noising process. If the noise schedule is not perfectly tuned, this mismatch can cause L_T to dominate the overall loss during optimization. This phenomenon has been observed empirically and aligns with findings in prior work (Nichol & Dhariwal, 2021).

Moreover, L_T, though constant with respect to model parameters, can distort the magnitude of the total loss, overshadowing model-dependent terms like L_0 and L_{t-1}. This distortion can hinder convergence and lead to suboptimal optimization dynamics (see Figure 2). While (Nichol & Dhariwal, 2021) proposed addressing this issue by refining the noise schedule, we chose to exclude L_T from the training loss entirely. This simplifies the implementation and allows the optimization process to focus on the model-dependent terms without sacrificing theoretical consistency during evaluation.

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

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: $x_{T-N\tau} = x_0$

3.2 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:

$$L_{t-1} = \sum_{t=T-\tau}^{T-(N-1)\tau} \lambda_t \left\| \hat{x}_\theta^{l=t}(\hat{x}_{t+\tau}, y) - x_t \right\|_2^2 \tag{26}$$

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 27).

$$FiLM(\hat{x}_{t+\tau}) = \gamma \odot \hat{x}_{t+\tau} + \beta \tag{27}$$

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:

$$L_0 = -\log p_\theta(\hat{x}_0 \mid \hat{x}_{T-(N-1)\tau}, y) \tag{28}$$

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 29, is a simplified version of the original loss in Equation 13,. This loss minimizes the difference between the predicted output $\hat{x}_\theta$ and the original input x_0, making it more computationally efficient.

$$L_{simple} = \left\| \hat{x}_\theta^{l=T-(N-1)\tau}(\hat{x}_{t+\tau}) - x_0 \right\|_2^2 \tag{29}$$

The second alternative, shown in Equation 30, 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.

$$L_{hybrid} = L_{simple} + \lambda L_{vlbfull} \tag{30}$$

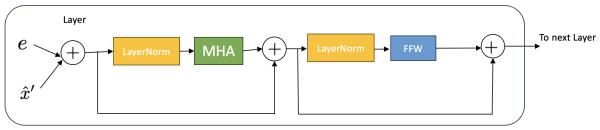

Figure 3: A single layer of the transformer model used for data recovery (see Figure 3).

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 (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 $\hat{x}'_{T-n\tau} \in \mathbb{R}^{F \times s \times V}$.

6 Evaluation

This section discusses the datasets and training parameters used to develop and evaluate the proposed technique, referred to as **UDPNet** (Unrolling Diffusion Process Network).

6.1 Dataset

To ensure comparability with existing tools and maintain alignment with trends in the speech synthesis domain, we evaluated UDPNet on two popular datasets: LJSpeech for single-speaker speech generation and VCTK for multi-speaker evaluation.

The **LJSpeech** dataset consists of 13,100 audio clips sampled at 22 kHz, totaling approximately 24 hours of single-speaker audio. Clip lengths range from 1 to 10 seconds, and all clips feature a single female speaker. Following (Chen et al., 2020), we used 12,764 utterances (23 hours) for training and 130 utterances for testing.

For multi-speaker evaluation, we used the **VCTK** dataset, which includes recordings of 109 English speakers with diverse accents, originally sampled at 48 kHz and downsampled to 22 kHz for consistency. Following (Lam et al., 2022), we used a split where 100 speakers were used for training and 9 speakers were held out for evaluation.

Feature Extraction: Mel-spectrograms were extracted from each audio clip, resulting in 128-dimensional feature vectors. The extraction process used a 50-ms Hanning window, a 12.5-ms frame shift, and a 2048-point FFT, with frequency limits of 20 Hz (lower) and 12 kHz (upper), similar to (Chen et al., 2020).

6.2 Training Parameters

UDPNet was trained on a single NVIDIA V100 GPU using the Adam optimizer. A cyclical learning rate (Smith, 2017) was employed, with the learning rate varying between $1e-4$ and $1e-1$. The batch size was set to 32, and training was performed over 1 million steps.

For **conditional speech generation**, Mel-spectrograms extracted from ground truth audio were used as conditioning features during training. During testing, Mel-spectrograms were generated by Tacotron 2 (Shen et al., 2018). To generate the FiLM parameters β and γ, we adopted the upsampling block approach proposed in Chen et al. (2020), where these parameters modulate layer activations to incorporate conditioning information.

Layer Contribution to Loss: Each neural network layer's contribution to the total loss L_{t-1} (Equation 20) was weighted using a layer-specific factor λ. The weights were initialized at $\lambda = 0.001$ for the first layer and incremented by 0.001 for each subsequent layer. This approach ensured that higher layers, which handle progressively refined denoising steps, had a greater impact on the overall loss. This weighting helps the model prioritize more challenging denoising tasks, which are typically assigned to the higher layers.

6.3 Baseline Models and Metrics

To evaluate UDPNet, we compared its performance against several state-of-the-art vocoders with publicly available implementations. The selected baseline models are:

- 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]

We assessed the models using a combination of subjective and objective metrics:

[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/

- **Mean Opinion Score (MOS):** Human evaluators rated the naturalness and quality of generated speech on a 5-point scale (1 = Bad, 5 = Excellent). Evaluators were recruited via Amazon Mechanical Turk, wore headphones, and rated 10 samples each.
- **Objective MOS Prediction:** We used three deep learning-based MOS prediction tools: SSL-MOS [9] (Cooper et al., 2022), MOSA-Net [10] (Zezario et al., 2022), and LDNet [11] (Huang et al., 2022c). These tools are widely used in the VoiceMOS challenge (Huang et al., 2022b).
- **F_0 Frame Error (FFE):** This metric measures pitch accuracy by quantifying discrepancies between the generated and ground truth audio.

Objective MOS Prediction Tools: SSL-MOS is a Wav2Vec-based model fine-tuned for MOS prediction by adding a linear layer to the Wav2Vec backbone. MOSA-Net incorporates cross-domain features, including spectrograms, raw waveforms, and features from self-supervised learning speech models, to enhance its predictions. LDNet estimates listener-specific MOS scores and averages them across all listeners for a final score.

6.4 Model Configurations

UDPNet was evaluated using different forward diffusion steps (**fsteps**) while maintaining a fixed number of 8 reverse steps. The forward steps considered were 1200, 1000, 960, 720, and 240, corresponding to skip parameters $\tau = \{150, 125, 120, 90, 30\}$, respectively. Each configuration accepted a 0.3-second audio input.

The choice of 1200 steps was motivated by our desire to explore different diffusion step configurations while ensuring a consistent number of 8 reverse steps across all models. This setup allows us to systematically analyze how the number of forward steps impacts both synthesis quality and computational efficiency.

To ensure a fair comparison with other models, such as WaveGrad, which operates with 1000 forward steps, we also evaluated UDPNet using 1000 forward steps while maintaining 8 reverse steps.

The forward noise schedule α_i was defined as a linear progression across all steps:

$$\alpha_i = Linear(\alpha_1, \alpha_N, N),$$

where N represents the total number of forward steps. For example, with 1200 forward steps, the schedule was specified as $Linear(1 \times 10^{-4}, 0.005, 1200)$.

During training, we conditioned UDPNet on Mel-spectrograms extracted from ground truth audio, while for testing, spectrograms generated by Tacotron 2 (Shen et al., 2018) were used. To enhance conditional generation, FiLM parameters β and γ were generated following the upsampling block approach proposed in Chen et al. (2020), modulating the activations of corresponding layers.

To ensure that higher layers, which handle finer denoising tasks, had a greater influence during training, each layer's contribution to the loss L_{t-1} was weighted. The weights λ were initialized at 0.001 for the first layer and incremented by 0.001 for each subsequent layer. This design ensures a progressive emphasis on higher layers, aligning with their role in refining the denoised output.

6.5 Results

6.5.1 Gradient Noise Scales of the Objective Functions

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

[9] https://github.com/nii-yamagishilab/mos-finetune-ssl
[10] https://github.com/dhimasryan/MOSA-Net-Cross-Domain
[11] https://github.com/unilight/LDNet

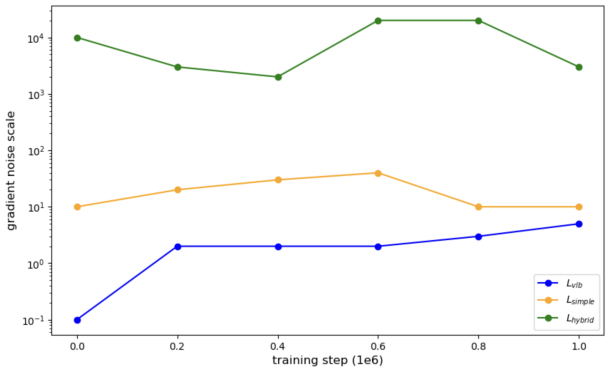

Figure 4: Gradient noise scales for the L_{vlb}, L_{hybrid}, and L_{simple} objectives on the LJSpeech dataset (see Figure 4).

As shown in Figure 4, L_{hybrid} exhibited the highest gradient noise levels. This behavior is attributed to the inclusion of the L_T term in the objective function. As noted by (Nichol & Dhariwal, 2021), L_T, which represents the KL divergence between the prior and forward noising process at the final timestep, introduces significant noise during uniform timestep sampling. This makes training less stable and impairs convergence.

In contrast, L_{vlb} demonstrated more stable gradient behavior compared to both L_{simple} and L_{hybrid}, making it the preferred choice for subsequent evaluations. Its stability ensures smoother optimization and better performance during training.

6.5.2 Single-Speaker Evaluation

To assess UDPNet's performance in conditional speech generation on a single-speaker dataset, we evaluated it using both subjective and objective metrics. Table 1 presents the subjective Mean Opinion Score (MOS) alongside objective MOS results obtained from SSL-MOS, MOSA-Net, and LDNet. Additionally, the table includes the Real-Time Factor (RTF), measuring 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. This is only 0.23 points lower than the ground truth score of 4.72, demonstrating that UDPNet can produce high-quality, natural-sounding speech closely approximating the original audio. Moreover, UDPNet (**1200, 8**) outperformed all other models in objective MOS metrics across SSL-MOS, MOSA-Net, and LDNet, confirming its superior ability to generate high-fidelity, distortion-free speech.

To ensure a fair comparison with other models that use 1000 forward steps, we evaluated UDPNet under the same condition with 1000 forward steps and 8 reverse steps (**1000, 8**). Under this configuration, UDPNet achieves a subjective MOS of 4.44, only 0.05 lower than the (**1200, 8**) configuration. However, it still outperforms other models using 1000 forward steps while also demonstrating significantly faster inference speed (RTF of 0.00389 vs. 38.2 for WaveGrad).

The slight MOS improvement in the (**1200, 8**) configuration is due to the more gradual denoising process, which better preserves fine-grained speech details and minimizes high-frequency artifacts. However, the diminishing returns beyond 1000 steps suggest that UDPNet remains highly effective even with fewer diffusion steps.

In terms of speed, all UDPNet configurations demonstrated competitive RTF values. Notably, reducing the total number of forward steps led to faster generation times. The configuration using 240 forward steps achieved the lowest RTF of 0.00182, indicating that fewer forward steps reduce the computational burden during inference, accelerating speech generation.

We also observed that increasing the number of forward steps improved audio quality, as reflected in higher subjective and objective MOS scores. This improvement is likely due to a more gradual denoising process,

which better preserves fine-grained speech details and minimizes high-frequency artifacts. Additionally, the use of latent variables x_t as targets in the objective function L_{vlb}, instead of directly predicting x_0, helps minimize prediction errors. Previous studies (Zhou et al., 2023) indicate that large prediction errors contribute to speech distortion, and their reduction in UDPNet likely explains its superior performance.

Finally, UDPNet (**1200, 8**) achieved the lowest F_0 Frame Error (FFE) rate of 2.3%, further confirming its ability to maintain pitch accuracy and reduce speech distortion compared to other state-of-the-art models.

Table 1: Evaluation results of UDPNet compared to state-of-the-art tools on the LJSpeech test dataset (see Table 1). Metrics include subjective MOS, objective MOS (SSL-MOS, MOSA-Net, and LDNet), F_0 Frame Error (FFE), and Real-Time Factor (RTF).

LJSpeech Test Dataset						
Model	**MOS(↑)**	**SSL-MOS(↑)**	**MOSA-Net(↑)**	**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: 1000, rsteps: 8)	4.44±0.12	4.37	4.31	4.40	3.3%	0.00389
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**

6.5.3 Multi-Speaker

6.5.4 Multi-Speaker Evaluation

The evaluation of UDPNet on the multi-speaker VCTK dataset demonstrates its robust generalization to unseen speakers. As shown in Table 2, UDPNet (1200, 8) achieved a subjective MOS of 4.38, closely matching the ground truth score of 4.63, and outperformed all baseline models in objective MOS metrics, including SSL-MOS, MOSA-Net, and LDNet.

While UDPNet slightly trails DiffWave in F0 Frame Error (FFE) by 0.1%, this marginal difference underscores its balanced design, which prioritizes both speech quality and efficiency. The fixed 8 reverse steps, enabled by the novel layer-timestep mapping, contribute to its superior efficiency while maintaining competitive FFE.

The UDPNet (**1000, 8**) configuration achieves a subjective MOS of 4.35, surpassing WaveGrad's 4.26 while demonstrating a significantly lower inference time (RTF 0.00389 vs. 38.2 for WaveGrad). This confirms that UDPNet remains highly effective even when configured with the same number of forward steps as WaveGrad.

Moreover, UDPNet's real-time factor (RTF) of 0.0042 in the (1200, 8) configuration demonstrates a significant speed advantage over traditional diffusion models. This efficiency makes UDPNet particularly well-suited for real-time applications, such as AI-powered voice assistants or multi-speaker transcription systems.

The layer-timestep mapping introduced in UDPNet ensures efficient denoising, contributing to its ability to generate high-quality speech while requiring fewer computational resources. For instance, the (240, 8) configuration achieves the fastest RTF of 0.00182, showcasing UDPNet's flexibility in balancing quality and efficiency through adjustable skip parameters (τ).

In summary, UDPNet (1200, 8) exemplifies the potential of its architectural novelties, achieving high fidelity, strong generalization, and competitive inference speeds in multi-speaker scenarios.

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: 1000, rsteps: 8)	4.35±0.05	4.39	4.36	4.37	3.3%	0.00389
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**

6.5.5 Unconditional Speech Generation

For unconditional speech generation, the model was trained on the multi-speaker VCTK dataset. To generate speech samples, random white noise was sampled and processed through the trained UDPNet without conditioning on any acoustic features. The results for unconditional speech generation are presented in Table 3.

For short clips, the best-performing configuration, UDPNet (fsteps: 1200, rsteps: 8), achieved a subjective MOS of 3.11. Listening to the audio samples, we observed a phenomenon where the generated clips begin with coherent and natural-sounding sentences, but the coherence diminishes as the duration increases. This issue suggests a need for future work to better understand and address the model's temporal coherence limitations.

Despite this challenge, UDPNet generates clean-sounding speech with minimal noise or artifacts. This demonstrates its ability to produce high-quality unconditioned speech while maintaining efficiency, as indicated by the competitive real-time factors (RTF) across configurations. Notably, the fastest configuration, UDPNet (fsteps: 240, rsteps: 8), achieved an RTF of 0.00162, showcasing the scalability of the proposed architecture.

Table 3: Results of the unconditioned version of UDPNet on the 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: 1000, rsteps: 8)	**3.07**±0.12	**3.15**	**3.13**	**3.18**	0.00367
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 22

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 $p_\theta(\hat{x}_{t-1}|\hat{x}_t)$:

$$L_{t-1} = \arg\min_\theta \mathbb{E}_{t\sim U(2,T)} D_{\mathrm{KL}} \left(q(x_{t-1}|x_t, x_0) \| p_\theta(\hat{x}_{t-1}|\hat{x}_t) \right) \tag{31}$$

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:

$$L_{t-1} = \arg\min_\theta \mathbb{E}_{t\sim U(2,T)} D_{\mathrm{KL}} \left(\mathcal{N}(x_{t-1}; \mu_q(t), \Sigma_q(t)) \| \mathcal{N}(\hat{x}_{t-1}; \hat{\mu}_\theta, \Sigma_q(t)) \right) \tag{32}$$

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:

$$\begin{gathered} D_{\mathrm{KL}} \left(\mathcal{N}\left(x_{t-1}; \mu_q(t), \Sigma_q(t)\right) \| \mathcal{N}\left(\hat{x}_{t-1}; \hat{\mu}_\theta, \Sigma_q(t)\right) \right) = \\ \frac{1}{2}\left[\log \frac{|\Sigma_q(t)|}{|\Sigma_q(t)|} - d + \mathrm{tr}\left(\Sigma_q(t)^{-1}\Sigma_q(t)\right) + \left(\hat{\mu}_\theta - \mu_q(t)\right)^T \Sigma_q(t)^{-1} \left(\hat{\mu}_\theta - \mu_q(t)\right) \right] \end{gathered} \tag{33}$$

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

$$D_{\mathrm{KL}} \left(\mathcal{N}\left(x_{t-1}; \mu_q(t), \Sigma_q(t)\right) \| \mathcal{N}\left(\hat{x}_{t-1}; \hat{\mu}_\theta, \Sigma_q(t)\right) \right) = \frac{1}{2}\left[\left(\hat{\mu}_\theta - \mu_q(t)\right)^T \Sigma_q(t)^{-1} \left(\hat{\mu}_\theta - \mu_q(t)\right) \right] \tag{34}$$

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

$$\frac{1}{2\Sigma_q(t)} \left[\left(\frac{\sqrt{\alpha}(1-\bar{\alpha}_{t-1})x_\theta(\hat{x}_{t+1}, t) + \sqrt{\bar{\alpha}_{t-1}}(1-\alpha_t)x_0}{1-\bar{\alpha}_t} - \frac{\sqrt{\alpha}(1-\bar{\alpha}_{t-1})x_t + \sqrt{\bar{\alpha}_{t-1}}(1-\alpha_t)x_0}{1-\bar{\alpha}_t} \right)^2 \right] \tag{35}$$

Simplifying this expression:

$$\frac{\sqrt{\alpha}(1-\bar{\alpha}_{t-1})}{2\Sigma_q(t)(1-\bar{\alpha}_t)} \left[\|x_\theta(\hat{x}_{t+1}, t) - x_t\|_2^2 \right] \tag{36}$$

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

$$L_{t-1} = \frac{\sqrt{\alpha}(1-\bar{\alpha}_{t-1})}{2\Sigma_q(t)(1-\bar{\alpha}_t)} \left[\| x_\theta(\hat{x}_{t+1}, t) - x_t \|_2^2 \right] \tag{37}$$