FRAMEWISE WAVEGAN: HIGH SPEED ADVISORIAL VOCODER IN TIME DOMAIN WITH VERY LOW COMPUTATIONAL COMPLEXITY

Ahmed Mustafa\textsuperscript{*} Jean-Marc Valin\textsuperscript{*} Jan Büthe\textsuperscript{*} Paris Smaragdis\textsuperscript{†} Mike Goodwin\textsuperscript{*}

\textsuperscript{*}Amazon Web Services, Palo Alto, USA
\textsuperscript{†}University of Illinois at Urbana-Champaign
{ahdmust, jmvalin, jbuethe, parsmara, mmg}@amazon.com

ABSTRACT

GAN vocoders are currently one of the state-of-the-art methods for building high-quality neural waveform generative models. However, most of their architectures require dozens of billion floating-point operations per second (GFLOPS) to generate speech waveforms in samplewise manner. This makes GAN vocoders still challenging to run on normal CPUs without accelerators or parallel computers. In this work, we propose a new architecture for GAN vocoders that mainly depends on recurrent and fully-connected networks to directly generate the time domain signal in framewise manner. This results in considerable reduction of the computational cost and enables very fast generation on both GPUs and low-complexity CPUs. Experimental results show that our Framewise WaveGAN vocoder achieves significantly higher quality than auto-regressive maximum likelihood vocoders such as LPCNet at a very low complexity of 1.2 GFLOPS. This makes GAN vocoders more practical on edge and low-power devices.

Index Terms— GAN vocoder, LPCNet, TTS, Speech synthesis, Neural speech coding

1. INTRODUCTION

The task of neural vocoding has seen rapid progress in recent years. This came with lots of applications that depend on neural vocoders for rendering high quality speech; such as TTS [1], speech enhancement [2] and neural speech coding [3, 4, 5]. Since the advent of WaveNet [6], deep generative models have become the standard way for building high quality neural vocoders, with clear outperformance over classical vocoding methods. By modelling raw waveform data distributions in the time domain, deep generative models can capture intricate representational details of the speech signal, which are not easy to track by fully-parametric methods. This results in generating high quality speech signals; but it also incurs huge amount of computations that scales rapidly with the target sampling rate. In case of WaveNet, this high computational cost becomes more challenging due to the auto-regressive (AR) sampling that makes the overall generation very slow. Later models such as WaveRNN [7] tried to address one part of this problem by using a sparse RNN that runs auto-regressively to learn the data distribution. LPCNet [8] further improved WaveRNN by using linear prediction to build lighter architectures while maintaining or even enhancing the signal quality; and currently it is one of the lowest complexity generative models available for raw audio. Further AR models [9, 10] follow the same approach but generate more than one sample at a time to speed up the inference process.

Parallel generative models propose a different way for dealing with the high computational cost. These models avoid auto-regressive generation to parallelize computations via GPUs or CPU accelerators. There are many types of parallel vocoders which are similar in their architecture but differ in the way they are trained. GAN vocoders [11, 12, 13] are more competitive in parallel generation due to their light-weight models, as compared to maximum-likelihood (ML) vocoders like WaveGlow [14], and their one-shot generation, as opposed to diffusion vocoders [15, 16] that usually require iterative denoising steps to achieve high quality signal generation. However, despite their high quality and very fast signal generation, GAN vocoders still require multiples of Giga Floating-Point Operations per Second (GFLOPS), especially when using WaveNet-based architectures. This amount of computation is challenging for low complexity devices with limited power and parallelization capabilities.

Our work aims to reduce the large computational amount of current GAN vocoders. Specifically, we report the following contributions:

• We introduce Framewise WaveGAN, a neural vocoder generating wideband speech signals in the time domain frame-by-frame instead of sample-by-sample.
• We show how to train the model reliably and stably by training in perceptual domain using a combination of multi-resolution STFT regression and adversarial losses.
• We demonstrate the quality of the proposed model that outperforms LPCNet at a very low complexity of 1.2 GFLOPS according to subjective and objective scores.

2. FRAMEWISE WAVEGAN

Most GAN vocoders can be categorized into WaveNet-based and latent-based models. In WaveNet-based models, all network parameters are used at the same rate as the target signal. That is, if we seek to generate speech signal at sampling rate \( f_s \), then every parameter in the model will do \( f_s \) multiplications plus \( f_s \) accumulations as the total number of floating-point operations per second (FLOPS). For light-weight vocoders such as Parallel WaveGAN [17] with 1.44 M parameters, generating one second of speech at 16 KHz would require 46 GFLOPS.

Latent-based models alleviate this computational burden by building the network as a stack of upsampling layers; where each layer operates at different resolution starting from the acoustic feature rate until reaching the target sampling rate of the signal. This can achieve moderate computational cost in ranges of 10 GFLOPS [18] and 8 GFLOPS [19], but with sophisticated architecture setups to maintain high quality.
Framewise WaveGAN is a first step towards running GAN vocoders in the time domain at the acoustic feature rate, without having to use upsampling layers which are the main source of high complexity. This is achieved by making the model generate one frame at a time. Normally, in WaveNet-based and latent-based models, the feature representations starting from the first layer until the last one are organized as tensors of $[\text{Batch} \_\text{dim}, \text{Channel} \_\text{dim}, \text{Temporal} \_\text{dim}]$; with Temporal_dim equal to the target signal resolution at the output layer. In Framewise WaveGAN, all feature representations are organized as $[\text{Batch} \_\text{dim}, \text{Sequence} \_\text{dim}, \text{Frame} \_\text{dim}]$, where Sequence_dim is equal everywhere to the acoustic feature resolution that is commonly much smaller than the signal one; and Frame_dim holds the representation of the target frame that is being generated. The final waveform is obtained by simply flattening the generated frames at the model output. This leads to significant computational saving even with models of large memory footprint.

![Framewise WaveGAN generator architecture. The numbers show $\{\text{Sequence} \_\text{dim}, \text{Frame} \_\text{dim}\}$ of the output representation from each layer to generate 1 sec of speech waveform at sampling rate of 16 kHz, using conditioning acoustic features at 100 Hz.](image)

**Fig. 1**: Framewise WaveGAN generator architecture. The numbers show $\{\text{Sequence} \_\text{dim}, \text{Frame} \_\text{dim}\}$ of the output representation from each layer to generate 1 sec of speech waveform at sampling rate of 16 kHz, using conditioning acoustic features at 100 Hz.

### 2.1. Architecture Overview

Fig. 1 illustrates the architecture of the proposed model. It mainly consists of two stacks of recurrent and fully-connected layers. The recurrent stack has 5 GRUs [20] to model long-term dependencies of the signal. All GRU outputs are concatenated with the conditioning (i.e., acoustic feature) representation and converted into lower-dimensional latent representation through a simple fully-connected layer. This representation is then utilized by the fully-connected stack that operate in framewise convolutional manner to model the short-term dependencies of the signal.

### 2.2. Framewise Convolution

The term framewise convolution refers to a kernel whose elements are frames instead of samples. We implement this by making the fully-connected layer receive at frame index $i$ a concatenation of $k$ frames at indices $\{i - k + 1, \ldots, i\}$ from the input tensor, where $k$ is the kernel size. The rest of the operation is same as normal convolution. There is also conditional framewise convolution that only differs from framewise convolution in concatenating an external feature frame (i.e., conditioning vector) to the layer input. In our model, we use 1 framewise convolution layer that receives the latent representation from the previous stack, with a kernel size of 3 frames, stride = dilation = 1 frame; and padding in non-causal manner (i.e., 1 look-ahead frame). Hence, if the input tensor to this layer has Frame_dim of e.g., 512, then the fully-connected network should have $3 \times 512 = 1536$ input dimensions. In addition, there are 4 conditional framewise convolution layers coming afterwards with a kernel size of 2 frames which are concatenated with 1 conditioning frame provided by the same latent representation obtained from the previous stack; with same stride, dilation and padding applied in causal manner. That’s why the fully-connected network for this conditional layer has the same dimensionality as the non-conditional one. All of these framewise convolution operations are running in a single-channel sense; i.e., there is only one fully-connected network per layer. We used this way of implementation instead of traditional multi-channel convolution layers to ease the efficient implementation of the model, especially when applying sparsification methods to these layers.

### 2.3. Activation Layers

For all layers in the recurrent and framewise convolution stacks, we use Gated Linear Unit (GLU) [21] to activate their feature representations:

$$GLU(X) = X \otimes \sigma(FC(X)),$$  

where $FC$ is a simple fully-connected network to learn the sigmoid gate and it has the same output dimension as $X$, $\otimes$ denotes element-wise multiplication. We also disabled the bias for all layers in the model, which was helpful for faster convergence with lower reconstruction artifacts.

### 2.4. Conditioning Acoustic Features

We use LPCNet acoustic features [8] to condition our vocoder model. They consist of 18 Bark-Frequency Cepstral Coefficients (BFCCs), a pitch period and a pitch correlation; which are extracted by 20 ms overlapping windows at 100 Hz frame rate from a 16 kHz speech waveform. The model generates one 10 ms frame per conditioning vector. The pitch period is fed to an embedding layer of 256 levels and 128 dimensions, while the BFCCs with the pitch correlation are fed to a simple causal convolution layer of 128 output channels and kernel size of 3. The outputs from these two layers are then concatenated and fed to another causal convolution layer of 256 output channels, kernel size of 3 and Leaky_ReLU activation to obtain the acoustic feature representation that is used for framewise generation, as shown in figure 1. LPCNet features are calculated on 10 ms frames with 5 ms look-ahead; and Framewise WaveGAN requires one feature frame look-ahead. Hence, the total delay is 10 ms for framing plus 15 ms look-ahead, which sums to 25 ms.

### 3. TRAINING PROCEDURE

**3.1. Training in the Perceptual Domain**

Speech signals are characterized by their high dynamic range as they go wider in bandwidth. When we apply a simple pre-emphasis filter before training, we find the vocoder able to learn high frequency
components faster than training in the normal signal domain. We reinforce this benefit by additionally using perceptual filtering, as detailed in AMR-WB [22], so that the vocoder can learn high frequency content even faster. The perceptual weighting filter is defined by the following transfer function:

\[ W(z) = \frac{A(z/\gamma_1)}{(1 - \gamma_2 z^{-1})}, \]

where \( A(z) \) is the linear prediction (LPC) filter whose coefficients are computed from BFCCs. \( \gamma_1 = 0.92 \) and \( \gamma_2 = 0.85 \). This filtering increases the spectral flatness of signals during the training, which enables faster convergence. Moreover, when applying inverse filtering to obtain the final signal, the noise of reconstruction artifacts is shaped by \( W^{-1}(z)P^{-1}(z) \), where \( P^{-1}(z) \) is the de-emphasis applied at end of the synthesis. The computational cost of this perceptual filtering is also quite cheap and still keeps the overall complexity low.

### 3.2. Spectral Pre-training

We first pre-train the model using the spectral reconstruction loss \( \mathcal{L}_{\text{mag}} \) defined by Equation (6) in [17]. It is a combination of spectral magnitude and convergence losses obtained by different STFT resolutions. We use all power-of-two FFT sizes between 64 and 2048 (6 sizes), with same values for window sizes and 75% window overlap. For the spectral magnitude loss \( \mathcal{L}_{\text{mag}} \) defined by Equation (5) in [17], we apply \( \sqrt{\text{log}} \) instead of \( \log \) as a non-linearity, which was found better for early convergence. The spectral pre-training gives a metallic-sounding signal with over-smoothed high frequency content, which is a good prior signal to start adversarial training for achieving realistic signal reconstruction.

### 3.3. Spectral Adversarial Training

Using time-domain discriminators is a major challenge in adversarial training of the proposed model. We tried different time-domain discriminator architectures [23, 12, 11] that all failed to achieve stable training. Instead, we use multi-resolution spectrogram discriminators, which achieve much better training behavior and reliably increase the fidelity of generated signals. We follow the same spectrogram discriminator architecture defined in [24] and use 6 models running on spectrograms of the same STFT resolutions used for spectral pre-training; with \( \sqrt{\text{log}} \) as a non-linearity. The adversarial training uses least-square loss as a metric for evaluating discriminators, which achieve much better training behavior and reliably increase the fidelity of generated signals. We follow the same spectrogram discriminator architecture defined in [24] and use 6 models running on spectrograms of the same STFT resolutions used for spectral pre-training; with \( \sqrt{\text{log}} \) as a non-linearity. The adversarial training uses least-square loss as a metric for evaluating discriminators, which achieve much better training behavior and reliably increase the fidelity of generated signals.

\[ \min_{g} \left\{ \mathbb{E}_{g} \left[ \sum_{k=1}^{6} (D_k(G(s)) - 1)^2 \right] + \mathcal{L}_{\text{aux}}(G) \right\}, \]

where \( s \) represents the conditioning features (e.g., LPCNet features). Weight normalization [25] is applied to all convolution layers of the discriminators \( D_k \) and all fully-connected layers of the generator \( G \).

### 4. EXPERIMENTS AND RESULTS

#### 4.1. Experimental Setup

We train the proposed Framewise WaveGAN model for a universal vocoding task, where the synthesis is speech and language independent. To achieve this, we use a training speech corpus of 205 hours sampled at 16 kHz and obtained from a combination of TTS datasets [26, 27, 28, 29, 30, 31, 32, 33, 34]. The training data contains more than 900 speakers in more than 34 languages and dialects. Pre-emphasis with a factor of 0.85 is applied to the speech signals when extracting the input LPCNet features. The complete synthesis is performed by running the vocoder model and then applying inverse perceptual filtering followed by de-emphasis. The training runs on an NVIDIA Tesla V100 GPU. We use batch size of 32 and each sample in the batch corresponds to 1 sec for both features and speech data tensors. The spectral pre-training is carried out for 1 M steps (i.e., ~ 50 epochs), with \( lr_{\beta} = 10^{-4} \); and then the adversarial training runs for another 1 M steps, with \( lr_{\alpha} = 2 \times 10^{-4} \) and \( lr_{\beta} \) is reduced to \( 5 \times 10^{-5} \). AdamW [35] is used for both generator and discriminator optimizers, with \( \beta = \{0.8, 0.99\} \).

#### 4.2. Complexity

In Framewise WaveGAN, all model parameters are used at the same rate of the conditioning features, which is 100 Hz. Thus, each parameter contributes to one multiply and one accumulation (i.e., 2 floating-point operations) per generation step. The total computational complexity to generate one second is then given as follows:

\[ C = N \times 2 \times S, \]

where \( C \) denotes the number of floating-point operations per second (FLOPS), \( N \) is the total count of model parameters and \( S \) is the number of generation steps to create one second. Framewise WaveGAN according to the architecture defined in sec 2 has 7.8 M parameters. This gives a total complexity of 1.5 GFLOPS, including 7.3 MFLOPS for calculating \( \text{tanh} \) and \( \text{sigmoid} \) functions. We further reduce this complexity by sparsifying the dense model, as done in [7, 8], with weight densities of 0.6 for all GRUs and 0.65 for all fully-connected layers except the last three, which are kept dense. This decreases the number of active parameters to 5.9 M and makes the total complexity 1.2 GFLOPS. We also checked the parallel generation speed for the Pytorch implementation of Framewise WaveGAN at 1.2 GFLOPS in real-time factor (RTF). The model runs 20x and 75x faster than real-time on CPU (Intel Xeon Platinum 8175M 2.50GHz) and GPU (Nvidia Tesla V100), respectively.

#### 4.3. Quality

We evaluate the model on the PTDB-TUG speech corpus [36] and the NTT Multi-Lingual Speech Database for Telephonometry. From PTDB-TUG, we use all English speakers (10 male, 10 female) and randomly pick 200 concatenated pairs of sentences. For the NTT database, we select the American English and British English speakers (8 male, 8 female), which account for a total of 192 samples (12 samples per speaker). The training material did not include any hours sampled at 16 kHz and obtained from a combination of TTS datasets [26, 27, 28, 29, 30, 31, 32, 33, 34]. The training data contains more than 900 speakers in more than 34 languages and dialects. Pre-emphasis with a factor of 0.85 is applied to the speech signals when extracting the input LPCNet features. The complete synthesis is performed by running the vocoder model and then applying inverse perceptual filtering followed by de-emphasis. The training runs on an NVIDIA Tesla V100 GPU. We use batch size of 32 and each sample in the batch corresponds to 1 sec for both features and speech data tensors. The spectral pre-training is carried out for 1 M steps (i.e., ~ 50 epochs), with \( lr_{\beta} = 10^{-4} \); and then the adversarial training runs for another 1 M steps, with \( lr_{\alpha} = 2 \times 10^{-4} \) and \( lr_{\beta} \) is reduced to \( 5 \times 10^{-5} \). AdamW [35] is used for both generator and discriminator optimizers, with \( \beta = \{0.8, 0.99\} \).
Fig. 2 demonstrates the MOS test results performed by 30 listeners using Amazon Mechanical Turk. Framewise WaveGAN (FWGAN) at 1.2 GFLOPS achieves significantly higher quality than LPCNet at the same complexity\(^1\), even slightly outperforming (statistically significant) LPCNet operating at 3 GFLOPS. On the other hand, the higher-complexity FWGANs cannot outperform the 1.2 GFLOPS model. This is mainly due to the weaker discriminator behavior in adversarial training against generators with higher number of parameters. This may motivate future work to come up with better discriminator models that enable quality scaling with higher generator complexities. We also found Framewise WaveGAN has clearly better pitch consistency than the LPCNet model, as shown in Table 1; where PMAE is the pitch mean absolute error between ground truth and vocoder samples, VDE is the voicing decision error (i.e., the percentage of frames with incorrect voicing decision). We use YAAFT pitch tracker [40] for extracting the voicing information. Although we do not target singing voice generation with high accuracy, lower is better.

We would like to thank Umut Isik, Timothy B. Terriberry, Karim Helwani and Erfan Soltanmohammadi for their kind support and fruitful tips.

### 7. REFERENCES


---

\(^1\)See our demo samples at the following url: https://ahmed-fau.github.io/fwgan_demo/


