# AUDIO DIFFUSION FOR FOLEY SOUND SYNTHESIS

Technical Report

*Timo Wendner*
timo.wendner@gmail.com

*Tara Jadidi*
tara.jadidi@jku.at

*Patricia Hu*
patricia.hu@jku.at

*Alexander Neuhauser*
alexandermaxneuhauser@gmail.com

Johannes Kepler University
Linz, Austria

**ABSTRACT**

This technical report describes our approach to Task 7 (Foley Sound Synthesis), Track B (using no external resources other than the ones provided) of the DCASE2023 Challenge. This work was carried out by a student group, as part of an elective course in the Artificial Intelligence curriculum at Johannes Kepler University Linz.

We use an ensemble of U-Net based diffusion models for waveform generation in seven predefined sound categories. We apply gain reduction to normalize, and time shifting to augment the provided training data and test different noise schedulers and U-Net architectures. Applying different training strategies, we achieve competitive results for the majority of the sound classes while being more parameter efficient and allowing end-to-end generation on audio waveforms. Evaluated on the task's evaluation metric, i.e., the mean FAD score over all classes, we achieve a final score of 12.42 as compared to the score of the challenge baseline model of 9.68.

***Index Terms—*** Foley sound synthesis, diffusion, U-Net

## 1. INTRODUCTION

Foley sound effects are reproductions of everyday sound effects used in film and other forms of media. By providing realistic sounds, synchronized with the visual cues on screen, Foley sounds aim to enhance the perceived acoustic quality. The production of Foley sounds traditionally involves a human Foley artist, who manually records and manipulates sounds to create the desired acoustic properties. This process can be a time-consuming and expensive task in post-production, which leads to increased interest in using machine learning techniques to produce Foley sound. To reflect this increasing interest, the task of Foley sound generation has been introduced as a challenge in the DCASE23 [1].

For a considerable time, autoregressive models have been the dominant generative model architecture for audio generation [2, 3, 4]. Despite producing audio of adequate quality and long-term coherence, the generation process is often slow due to the sequential nature of sample generation. Non-autoregressive generative models have primarily focused on architectures involving Generative Adversarial Networks (GANs) [5, 6, 7], which are prone to unstable training. More recently, in the field of computer vision, diffusion models (DMs) have proven to be capable of generating high quality images overshadowing the performance of adversarial networks [8, 9]. Given the success in the image domain, some researchers have likewise applied DMs for generative tasks in the audio domain, focusing on either text-prompted audio [10, 11], speech [12] or music generation [13, 14].

Inspired by these results, we have chosen a U-Net based diffusion approach to tackle the Foley sound synthesis task in the B track variant (i.e., without the use of external models and datasets except the data provided). Moreover, we chose to apply our architecture on the waveform (as opposed to the spectrogram) representation given the recent results obtained by Huang et al. [14].

The remainder of this technical report is structured as follows: First, in Section 2 we shortly illustrate the task including a description of the baseline model, provided dataset and evaluation metric used. In Section 3 we explain our general approach using a waveform-based diffusion model with a U-Net-like architecture. Next, in Section 4 we report our experiments, including applied data augmentation and model configurations, and our final results. We conclude this report in Section 5.

## 2. TASK DESCRIPTION

### 2.1. Problem definition and evaluation metric

The Foley sound synthesis task is conceptualized as a category-to-sound generative task. For seven predefined sound categories, a generative system should generate 100 sound samples each. To further specify each sound class, a development set is provided which consists of 4,850 labeled audio files composed as a mixture from the UrbanSound8K [15], FSK50K [16] and BBC Sound Effects [17] datasets. All audio samples are provided (and should be generated) in mono, with a bit depth of 16-bit and sampling rate of 22,050 Hz, and a length of four seconds. The seven sound categories along with the number of samples provided for each class is shown in Table 1.

For the evaluation of the generated samples, the Fréchet Audio Distance (FAD) [18] is used. To this end, a pretrained classification model (in the context of the challenge, VGGish [19]) is used to compute embeddings for two given datasets (i.e., an evaluation and a generated set of samples). After estimating multivariate Gaussians

| ID | Category | Number of samples |
|---|---|---|
| 0 | DogBark | 617 |
| 1 | Footstep | 703 |
| 2 | GunShot | 777 |
| 3 | Keyboard | 800 |
| 4 | MovingMotorVehicle | 581 |
| 5 | Rain | 741 |
| 6 | Sneeze/Cough | 631 |

Table 1: Overview of sound categories and number of training samples in the development set.

$\mathcal{N}_e$ and $\mathcal{N}_g$ for the evaluation and generation set embeddings, the Fréchet distance between the two is computed as follows:

$$\mathbf{F}\left(\mathcal{N}_e, \mathcal{N}_g\right) = \|\mu_e - \mu_g\|^2 + tr\left(\Sigma_e + \Sigma_g - 2\sqrt{\Sigma_e\Sigma_g}\right) \quad (1)$$

### 2.2. Baseline system

The baseline system [20] for this task uses a Vector Quantized-Variational Autoencoder (VQ-VAE) [21] to transform the mel spectrogram of an audio file into a low-dimensional latent representation. Using this latent representation, a PixelSnail model [22] is trained to generate vectors in the latent space given the class label. In the generation phase, these generated vectors are decoded into mel spectrograms and a Hifi-GAN [23] trained on the same dataset is used to convert the spectrogram from the frequency domain back into waveforms in the time domain.

## 3. MODEL ARCHITECTURE

### 3.1. Diffusion

Diffusion models [24, 8, 9] build on the idea of iteratively adding Gaussian noise to a data point sampled from the real distribution $x_0 \sim q(x)$ over $T$ time steps until $x_T$ resembles an isotropic Gaussian distribution, and subsequently learning to reverse the diffusion process to recreate the true sample from that distribution.

The forward diffusion process is defined as a Markov chain of diffusion steps producing a sequence of increasingly noisy samples $x_1, ..., x_T$, where each transition is parameterized as a diagonal Gaussian:

$$q(x_t|x_{t-1}) = \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t I) \quad (2)$$

The amount of noise added at time step $t$ is defined by $\beta_t$, and is controlled by a scheduler $\beta_t \in (0,1)_{t=1}^{T}$. Typically, $\beta_t$ increases with increasing time steps. Using a reparameterization trick involving helper variable $\alpha_t$ where $\alpha_t = 1-\beta_t$ and $\bar{\alpha}_t = \prod_{i=1}^{t}\alpha_i$ allows us to sample $x_t$ at any arbitrary time step $t$. We can interpret $\bar{\alpha}_t$ as a measure of how much of the original properties of the data point remains at time step $t$.

Given that the forward distributions $q(x_t|x_{t-1})$ are modeled as Gaussians with scheduled mean and variance parameters per time step, we are interested only in learning the reverse conditionals $p_\theta(x_{t-1}|x_t)$ to allow us to generate a new sample $x_0$ from $p(x_T)$ through iterative denoising transitions over $T$ steps. Each denoising transition is parameterized as a Gaussian, which takes both the sample $x_t$ and the time step $t$ as inputs to account for different noise levels associated with different time steps:

$$p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \sigma_\theta(x_t, t)) \quad (3)$$

Interpreting the combination of $q$ and $p$ as a VAE [25], and consequently, $x_0$ as observed and $x_1, ..., x_T$ as latent variables, minimizing the negative log-likelihood (NLL) is approached by minimizing the variational lower bound (VLB) (see [24, 8, 9] for the full formulation).

Further, the reverse process can be reparameterized to allow the model to predict the noise $\epsilon$, which leads to the following loss function (see [8] for the derivation):

$$L_{simple} = E_{t,x_0,\epsilon}[||\epsilon = \epsilon(x_t, t)||^2] \quad (4)$$

Training the reverse model involves sampling a data sample from the real distribution, $x_0 \sim q(x_0)$, a time step from a uniform distribution $t \sim U(1, T)$ and noise from a normal distribution $\epsilon \sim \mathcal{N}(0, I)$ and minimizing the reparameterized objective function in Equation 4.

For the synthesis of new samples, we start by sampling the fully noised sample from a normal distribution, $x_T \sim \mathcal{N}(0, I)$. For each time step $t > 1$, we sample a noise vector from a normal distribution $z \sim \mathcal{N}(0, I)$, and iteratively compute $x_{t-1}$:

$$x_{t-1} = \frac{1}{\sqrt{\alpha_t}} \cdot \left(x_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \cdot \epsilon_\theta(x_t; \theta)\right) + z \cdot \sqrt{\beta_t} \quad (5)$$

We repeat this process until we get to $t = 1$, where we repeat the same process but take $z = 0$. The result $x_0$ is then the new sample.

### 3.2. U-Net

For the denoising model we choose a U-Net architecture [26] (see Fig. 1) given its robust latent representation capabilities paired with the ability to generate fine details. The model functionality resembles that of the baseline VAE, which likewise encodes data into into a lower-dimensional space before decoding it back into its original shape. However, the U-Net differs in that it incorporates skip connections between the encoder and the decoder part, which stabilize training and facilitate convergence by reusing features of the same dimensionality from earlier layers.

To further improve model performance, we augment the U-Net using two types of positional encodings, enabling the model to determine the current time step and class label. First, we use a sinusoidal position embedding [27], which encodes time step and label information as a sequence of sine and cosine functions with geometrically increasing wavelengths. We concatenate sinusoidal embeddings with the feature map at each down- and upsampling level. Second, we add another set of label and time step encodings at the lowest bottleneck representation of the model. This is implemented using a one-hot encoding and a simple fully connected layer.

## 4. EXPERIMENTS

### 4.1. Inter-class and intra-class similarity

To better understand the sample variance between and within classes, we calculate the inter- and intra-class similarity in terms of their Fréchet distances. To compute the inter-class similarity of

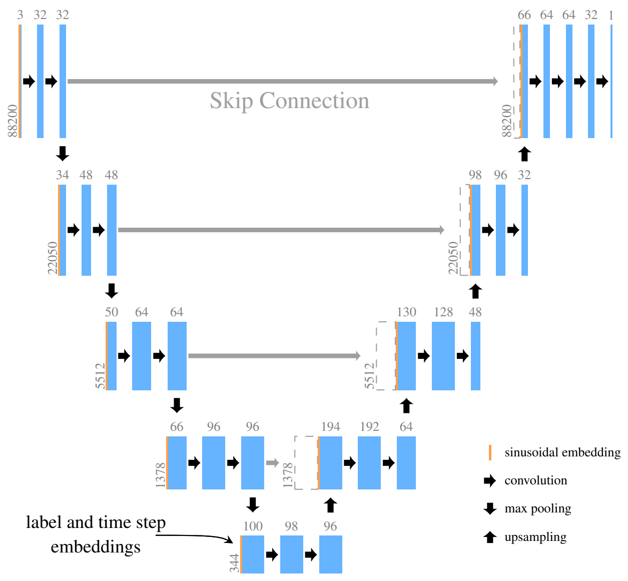

Figure 1: U-Net architecture used for classes DogBark, GunShot, Sneeze/Cough

a given class, we compare a random subset of 100 samples of one class with random subsets of the same size of each of the other classes. We follow a similar procedure for the intra-class similarity computation, where we compute the distance between two random subsets of 100 samples of the same class (note that samples in those subsets might overlap).

From the result of the inter-class similarity computation (see Table 2) we can see that class 4 (MovingMotorVehicle) is particularly dissimilar to the other classes, which partly explains why pre-training on the entire dataset did not yield improved results for this class (as opposed to other classes). Given these observations, we decided to split the classes into separate groups based on their similarity, and try pre-training on the group a class belonged to.

| class ID | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 15.67 | 18.6 | 18.62 | 39.99 | 28.35 | 15.45 |
| 1 | 15.67 | 0 | 13.25 | 7.38 | 38.32 | 21.66 | 13.10 |
| 2 | 18.6 | 13.25 | 0 | 15.75 | 35.9 | 24.68 | 18.00 |
| 3 | 18.62 | 7.38 | 15.75 | 0 | 36.11 | 21.38 | 14.88 |
| 4 | 39.99 | 38.32 | 35.9 | 36.11 | 0 | 20.36 | 52.15 |
| 5 | 28.35 | 21.66 | 24.68 | 21.38 | 20.36 | 0 | 14.88 |
| 6 | 15.45 | 13.10 | 18.00 | 14.88 | 52.15 | 14.88 | 0 |

Table 2: "Similarity matrix" computed as the FAD distances between each pair of classes in the training set.

## 4.2. Data preprocessing and augmentation

As part of the preprocessing stage, we apply normalization followed by a reduction of 2% to prevent digital clipping. To augment the dataset we perform gain reduction according to a normal distribution, as well as time shifting on arbitrary time points. Both gain reduction and time shifting are performed online during training, resulting in theoretically infinitely augmented data set sizes. This also allows us to keep the dataset in memory during training.

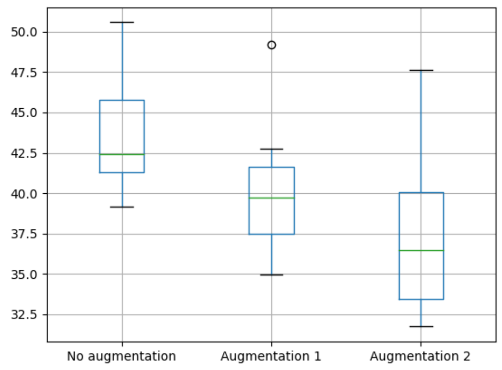

Figure 2: Training for the class MovingMotorVehicle (class ID 4) using no offline data augmentation (left), one (middle) and two (right) rounds augmentation using generated samples.

Given the poor performance of the sound category MovingMotorVehicle (class 4), we apply an additional offline data augmentation step by adding generated samples to the dataset, where we choose the generated samples based on the following procedure: we train a model on the original (preprocessed and online augmented) dataset for 400 epochs, and generate 100 samples every 25th epoch. We then calculate the FAD score between of each of these sets of 100 samples and a random subset of the original training set, select the model yielding the lowest score, and generate another 1000 samples using this model. Among these samples we select a random subset of 100 samples 100 times, again calculate the FAD score to the training set, and subsequently choose the 100 samples closest to the training set in terms of FAD to add to the training data and retrain the model. Overall we perform this process, effectively increasing the dataset size for this class by 200 samples and find that it is effective in lowering the mean FAD scores over the same amount of epochs (see Figure 2).

## 4.3. Empirical observations

**Diffusion variance scheduler.** Determining the number of time steps $T$ and the optimal amount of noise added at each time step is a critical consideration. For all our models we chose the number of time steps to be $T = 250$, resulting in a significant reduction in training time. Given the reduced number of time steps, we set a higher value at the final time step $T$ while applying the same start value at $t_0$ as [8], i.e., $\beta_T = .4$ and $\beta_0 = 10^{-4}$.

Likewise, we experimented with different types of variance schedulers, specifically linear, sigmoidal and quadratic functions, and found the quadratic schedule to perform the best for most models and classes (all except for Moving Motor Vehicle (4), although the difference was not significant). Looking at Fig. 3, in which we compare different variance schedulers, we can see that the quadratic one provides a near-linear decrease in the middle, and more subtle changes around the start and end of the training process. This curve pattern has been found beneficial for the denoising process in previous work [9].

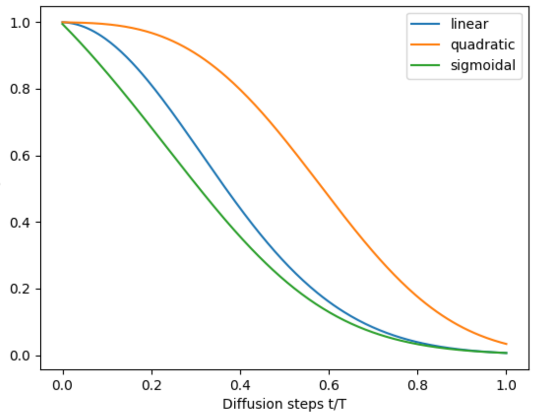

Figure 3: Comparison of different variance schedulers for the forward diffusion process.

**Pretraining.** We employed various pretraining strategies, depending on the sound class to be generated and its similarity to other classes, as described in Section 4.1. For classes DogBark (ID 0), GunShot (2), and Keyboard (3), we found that pretraining on the whole or parts of the dataset did not yield significant improvement. For classes Footstep (1) and Rain (5), we adopted a two-stage pretraining approach, where the first stage involved pretraining on the entire dataset, followed by pretraining on a group of classes most similar to the target class and eventual finetuning on each class. For MovingMotorVehicle (4), we obtained the best results by pretraining exclusively on samples from the same class and class Rain (5), followed by finetuning. Finally, pretraining on the whole dataset followed by finetuning yielded the best results for class Sneeze/Cough (6).

The duration of training varied for each class, ranging from approximately 600 epochs for the class Keyboard (3) to approximately 4400 epochs for the class Gunshot (2). We trained all our models using the Adam optimizer [28] and a learning rate of $10^{-4}$.

**Model Sizes.** We experimented with models of different sizes which all followed a U-Net architecture, differing either in kernel sizes, channel sizes and/or number of convolutional layers, and achieved the best results with two small models with 170k and 620k parameters each, and a bigger model with 1.4M parameters.

For classes DogBark (0), Gunshot (2), MovingMotorVehicle (4) and Sneeze/Cough (6) we obtained the best results with our biggest model, which corresponds to Figure 1. At each level in the downsampling path, two one-dimensional convolutions are computed with kernel sizes of 9 and padding of 4 to restore the original sample size. This is followed by a pooling layer with a kernel size of 4 and a stride of 4, resulting in the down-sampled signal having a size of $\frac{1}{4}$ of the sample size at the preceding level. At each level in the upsampling path, two convolutions using the same kernel size and amount of padding are computed, after which the signal is expanded using transposed convolution with a kernel size of 4 and stride of 4.

For classes Footstep (1) and Rain (5), we used a similar model structure but applied increasing kernel sizes in the downward path, along with decreasing kernel sizes in the upward path. For the class Keyboard (3) we obtained the best result using our smallest model,

| **Class** | **Train** | **Baseline** | **Ours** |
|---|---|---|---|
| DogBark | 1.47 | 13.41 | 8.60 |
| Footstep | 2.48 | 8.11 | 9.33 |
| GunShot | 3.27 | 7.95 | 11.77 |
| Keyboard | 3.86 | 4.84 | 6.41 |
| MovingMotorVehicle | 7.02 | 16.11 | 30.66 |
| Rain | 4.52 | 13.34 | 9.74 |
| Sneeze/Cough | 0.64 | 4.01 | 10.50 |
| Mean | 3.32 | 9.68 | 12.42 |

Table 3: FAD scores for the training set, baseline model and generated samples from our models.

which differs in structure to that of our biggest one in the number of channels used at each level.

Overall, our ensemble of models of different sizes has a parameter count of 7.1M parameters compared to the 105M parameters in the cascaded baseline model.

**Results.** Using our ensemble of models and class-specific training schedule, we were able to surpass the baseline score for the DogBark (0) and Rain (5) classes, and obtain comparable results for the classes Footstep (1) and Keyboard (3). An overview of the final FAD scores our ensemble achieved can be seen in Table 3, along with the FAD scores computed for (a random subset) of the training set, and generated samples of the baseline model.

## 5. CONCLUSION

We presented our approach for Task 7, Foley sound synthesis, track B (i.e., without the use of external models and datasets except those provided) in this technical report. To generate samples in seven predefined sound categories, we used an ensemble of U-Net based diffusion models. We outlined data preprocessing as well as online and offline augmentation techniques used, and reported our findings and observations with respect to experiments with different pretraining strategies and model configurations.

Overall, our ensemble of models is significantly more parameter efficient (totaling a parameter count of 7.1M compared to 105M in the challenge baseline model) and allows for end-to-end generation of audio waveforms. We report a final mean FAD score over all sound classes of 12.42 compared to the baseline score of 9.68.

## 6. ACKNOWLEDGEMENTS

We are all students in the AI curriculum at the Johannes Kepler University (JKU), Linz, Austria. We want to thank the Institute of Computational Perception and its head Prof. Gerhard Widmer for giving us the opportunity to work on this task. Furthermore, we want to give special thanks to Paul Primus and Florian Schmid for providing us with guidance and support over the course of the challenge.

## 7. REFERENCES

[1] K. Choi, J. Im, L. Heller, B. McFee, K. Imoto, Y. Okamoto, M. Lagrange, and S. Takamichi, “Foley Sound Synthesis at the

DCASE 2023 Challenge," 2023, https://arxiv.org/abs/2304.12521.

[2] A. v. d. Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, "Wavenet: A generative model for raw audio," *arXiv preprint arXiv:1609.03499*, 2016.

[3] P. Dhariwal, H. Jun, C. Payne, J. W. Kim, A. Radford, and I. Sutskever, "Jukebox: A generative model for music," *arXiv preprint arXiv:2005.00341*, 2020.

[4] A. Caillon and P. Esling, "RAVE: A variational autoencoder for fast and high-quality neural audio synthesis," 2021.

[5] C. Donahue, J. McAuley, and M. Puckette, "Adversarial audio synthesis," *7th International Conference on Learning Representations (ICLR)*, May 2019.

[6] J. Engel, K. K. Agrawal, S. Chen, I. Gulrajani, C. Donahue, and A. Roberts, "GANSynth: Adversarial neural audio synthesis," *7th International Confer- ence on Learning Representations (ICLR)*, May 2019.

[7] K. Kumar, R. Kumar, T. de Boissiere, L. Gestin, W. Z. Teoh, J. Sotelo, A. de Brebisson, Y. Bengio, and A. Courville, "MelGAN: Generative Adversarial Networks for Conditional Waveform Synthesis," 2019.

[8] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," *Advances in Neural Information Processing Systems*, vol. 33, pp. 6840–6851, 2020.

[9] A. Q. Nichol and P. Dhariwal, "Improved denoising diffusion probabilistic models," in *International Conference on Machine Learning*. PMLR, 2021, pp. 8162–8171.

[10] D. Ghosal, N. Majumder, A. Mehrish, and S. Poria, "Text-to-Audio Generation using Instruction-Tuned LLM and Latent Diffusion Model," 2023.

[11] R. Huang, J. Huang, D. Yang, Y. Ren, L. Liu, M. Li, Z. Ye, J. Liu, X. Yin, and Z. Zhao, "Make-An-Audio: Text-To-Audio Generation with Prompt-Enhanced Diffusion Models," 2023.

[12] K. Shen, Z. Ju, X. Tan, Y. Liu, Y. Leng, L. He, T. Qin, S. Zhao, and J. Bian, "NaturalSpeech 2: Latent Diffusion Models are Natural and Zero-Shot Speech and Singing Synthesizers," 2023.

[13] F. Schneider, Z. Jin, and B. Schölkopf, "Mo\^ usai: Text-to-Music Generation with Long-Context Latent Diffusion," *arXiv preprint arXiv:2301.11757*, 2023.

[14] Q. Huang, D. S. Park, T. Wang, T. I. Denk, A. Ly, N. Chen, Z. Zhang, Z. Zhang, J. Yu, C. Frank, J. Engel, Q. V. Le, W. Chan, Z. Chen, and W. Han, "Noise2Music: Text-conditioned Music Generation with Diffusion Models," 2023.

[15] J. Salamon, C. Jacoby, and J. P. Bello, "A Dataset and Taxonomy for Urban Sound Research," in *22nd ACM International Conference on Multimedia (ACM-MM'14)*, Orlando, FL, USA, Nov. 2014, pp. 1041–1044.

[16] E. Fonseca, X. Favory, J. Pons, F. Font, and X. Serra, "FSD50K: an open dataset of human-labeled sound events," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 30, pp. 829–852, 2022.

[17] BBC. (1991) BBC Sound Effects Library. [Online]. Available: https://sound-effects.bbcrewind.co.uk/

[18] K. Kilgour, M. Zuluaga, D. Roblek, and M. Sharifi, "Fréchet Audio Distance: A Metric for Evaluating Music Enhancement Algorithms," 2019.

[19] S. Hershey, S. Chaudhuri, D. P. Ellis, J. F. Gemmeke, A. Jansen, R. C. Moore, M. Plakal, D. Platt, R. A. Saurous, B. Seybold, *et al.*, "CNN Architectures for Large-Scale Audio Classification," in *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2017, pp. 131–135.

[20] X. Liu, T. Iqbal, J. Zhao, Q. Huang, M. D. Plumbley, and W. Wang, "Conditional Sound Generation Using Neural Discrete Time-Frequency Representation Learning," *2021 IEEE 31st International Workshop on Machine Learning for Signal Processing (MLSP)*, pp. 1–6, 2021.

[21] A. van den Oord, O. Vinyals, and K. Kavukcuoglu, "Neural Discrete Representation Learning," 2018.

[22] X. Chen, N. Mishra, M. Rohaninejad, and P. Abbeel, "PixelSNAIL: An Improved Autoregressive Generative Model," 2017.

[23] J. Kong, J. Kim, and J. Bae, "HiFi-GAN: Generative Adversarial Networks for Efficient and High Fidelity Speech Synthesis," 2020.

[24] J. Sohl-Dickstein, E. Weiss, N. Maheswaranathan, and S. Ganguli, "Deep unsupervised learning using nonequilibrium thermodynamics," in *International Conference on Machine Learning*. PMLR, 2015, pp. 2256–2265.

[25] D. P. Kingma and M. Welling, "Auto-Encoding Variational Bayes," 2014, https://arxiv.org/abs/1312.6114.

[26] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional Networks for Biomedical Image Segmentation," 2015, https://arxiv.org/abs/1505.04597.

[27] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention Is All You Need," 2017, https://arxiv.org/abs/1706.03762.

[28] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization." in *ICLR (Poster)*, Y. Bengio and Y. LeCun, Eds., 2015. [Online]. Available: http://dblp.uni-trier.de/db/conf/iclr/iclr2015.html#KingmaB14