PIANO-SSM: DIAGONAL STATE SPACE MODELS FOR EFFICIENT MIDI-TO-RAW AUDIO SYNTHESIS

Dominik Dallinger, Matthias Bittner, Daniel Schnöll, Matthias Wess and Axel Jantsch

Christian Doppler Laboratory for Embedded Machine Learning
TU Wien
Vienna, Austria
`firstname.lastname@tuwien.ac.at`

ABSTRACT

Deep State Space Models (SSMs) have shown remarkable performance in long-sequence reasoning tasks, such as raw audio classification, and audio generation. This paper introduces Piano-SSM, an end-to-end deep SSM neural network architecture designed to synthesize raw piano audio directly from MIDI input. The network requires no intermediate representations or domain-specific expert knowledge, simplifying training and improving accessibility. Quantitative evaluations on the MAESTRO dataset show that Piano-SSM achieves a Multi-Scale Spectral Loss (MSSL) of 7.02 at 16kHz, outperforming DDSP-Piano v1 with a MSSL of 7.09. At 24kHz, Piano-SSM maintains competitive performance with an MSSL of 6.75, closely matching DDSP-Piano v2's result of 6.58. Evaluations on the MAPS dataset achieve an MSSL score of 8.23, which demonstrates the generalization capability even when training with very limited data. Further analysis highlights Piano-SSM's ability to train on high sampling-rate audio while synthesizing audio at lower sampling rates, explicitly linking performance loss to aliasing effects. Additionally, the proposed model facilitates real-time causal inference through a custom C++17 header-only implementation. Using an Intel Core i7-12700 processor at 4.5GHz, with single core inference, allows synthesizing one second of audio at 44.1kHz in 0.44s with a workload of 23.1GFLOPS/s and an 10.1μs input/output delay with the largest network. While the smallest network at 16kHz only needs 0.04s with 2.3GFLOP/s and 2.6μs input/output delay. These results underscore Piano-SSM's practical utility and efficiency in real-time audio synthesis applications.

1. INTRODUCTION

Synthesizing the sound of physical instruments plays a crucial role in modern music creation. It is an important component in the evolution of music composition and production. However, compared to other domains, synthesizing audio remains a challenging task due to the complex dynamics of audio sound waves. Additionally, high sampling rates force synthesizers to be computationally efficient in order to comply with real-time constraints, such as action-sound latencies above 10 ms, might already be disturbing [1].

Over the last years, we have seen a trend of generative neural-network-based music generation. Especially Differentiable Digital Signal Processing (DDSP) [2] based methods seem promising by implementing traditional synthesizers and digital signal processing operations as differentiable layers controlled by deep neural networks. To fit within music creation frameworks a couple of works [3, 4] adopted the DDSP approach to work with Musical Instrument Digital Interface (MIDI) input. For the specific task of MIDI to raw piano performance synthesis Cooper et al. proposed Piano-TTS [5] by using text-to-speech (TTS) models. Renault et al. proposed DDSP-Piano v1 [6] and v2 [7] extending the DDSP framework to handle polyphonic input. They motivate their model architecture based on high-level modeling knowledge, including several sub-models dealing with specificities of the piano sound, such as partials, inharmonicity, and beating. So far, there is no existing work that proposes an approach without explicitly using pre-domain knowledge included during architecture design.

Learning the filter parameters of causal Linear Time Invariant (LTI) filters with an Infinite Impulse Response (IIR) in the discrete-time domain is not new. Moreover, comparing the simplest form of a Recurrent Neural Network (RNN), the Elman Network, with a discrete LTI State Space Model (SSM) representation, the only difference lies in the non-linear hidden state recurrence and the bias units [1]. Recently there has been an advent of deep neural networks modeled as continuous time SSMs, along with architectures like the S4 model [8], and its variants (DSS, S4D, S5 etc.) [9, 10, 11]. They show remarkable results on raw audio classification, raw audio generation [12], as well as long-range reasoning tasks, where vanilla RNNs struggle. RNNs are known for their computational efficiency. They have constant memory and computational costs that scale linearly $O(T)$ with the sequence length T. The efficiency of deep continuous time SSMs at inference time is equivalent to vanilla RNNs by materializing the discrete variant of the LTI systems' differential equations [8]. Modeling parameters in continuous time and applying discretization rules, motivated by modern control systems, allows for switching the discrete dynamics. This allows for training the network with a high sampling rate, then switching to a lower sampling rate for inference if wished or necessary due to the computational limitations of the target computing platform.

Motivated by the recent advances of deep SSM based sequence models, and their good performance on raw audio sequence modeling tasks we propose Piano-SSM, a simple end-to-end trainable MIDI to raw piano audio synthesis neural network architecture based on diagonal deep SSMs, along with the following contributions:

- **End-to-end MIDI-to-Raw Audio Synthesis with SSMs**. Piano-SSM only consists of four SSM layers and a single linear layer. The first SSM layer directly processes MIDI inputs, and the linear layer generates raw piano audio. For training Piano-SSM models we propose STFT-

Mel-Mean Loss (SMML). This audio-focused loss is combined of a Short-Time Fourier Transform (STFT) Loss, a Mel-STFT Loss, and a raw audio Mean Loss. The performance of Piano-SSM is evaluated on two existing benchmark datasets MAESTRO [13] and MAPS [14].

- **Variable Synthesis Sampling Rate.**
  We show Piano-SSM's ability to be trained on audio data with a high sampling rate while generating audio at lower sampling rates and connect the performance loss to aliasing effects. We evaluate the performance of models trained on different sampling rates in combination with down-sampled models on the MAESTRO dataset.
- **Efficient Autoregessive Causal Inference**.
  The discrete version of Piano-SSM can be seen as multiple IIR filters stacked with nonlinearities and allows for fast causal autoregressive audio synthesis. Based on a custom C++17 header-only implementation we show the real-time capabilities on an Intel Core i7-12700 Processor.

The rest of the paper is organized as follows: Section 2 presents related work in the context of deep SSMs, approaches for pianosynthesis, and MIDI-audio paired piano datasets. Section 3 outlines the Piano-SSM architecture and the loss function used for training. Sections 4 – 6 show the experimental setup and results for two benchmark datasets MAESTRO and MAPS. Audio samples[1] and the source-code[2] are provided online.

2. BACKGROUND AND RELATED WORK

2.1. Deep State Space Models

Compared to standard RNNs, SSMs use a linear hidden state recurrence, which enables efficient, parallel computation via associative scan [11]. To form a universal function approximators, individual SSM layers are paired with nonlinearities [15]. Setting the foundation for good performance on long-range reasoning tasks, where RNNs struggle, Gu et al. proposed the HiPPO framework [16] for structured initialization of SSM layers. The resulting S4 layer is modeled using multiple Single Input Single Output (SISO) systems. The SISO approach allows for training in the frequency domain [8]. To improve training and inference speed, several authors proposed diagonal approximations of the state transition matrix initializations, resulting in S4D [9], DSS [10], the first diagonal Multi Input Multi Output (MIMO) version S5 [11].

The results for raw audio generation with architectures like SaShiMi [12], and the good performance on raw audio classification with models such as S4D and S5 show that SSM based architectures are good at processing raw audio samples. We therefore consider applying a SSM based neural network architecture for solving the challenging task of raw piano audio synthesis.

2.2. Methods for Piano Synthesis

Raw audio synthesis is a challenging task due to the complex dynamics of audio sound waves. As stated by Hayes et al. [1], piano synthesis can be categorized into four categories:
Physical model-based piano synthesis aims to accurately simulate the behavior of musical instruments by formulating and solving equations that model physical phenomena such as motion, energy propagation, and sound radiation. Due to the complexity of piano acoustics, physical modeling remains an active research field requiring deep expertise.
Signal model-based piano synthesis refers to techniques that generate sound by explicitly modeling the physical or mathematical properties of the underlying signal. Developing and fine-tuning such models requires expert knowledge involving complex mathematical formulations and signal-processing techniques.
Concatenative model-based piano synthesis is widely used in digital pianos. It involves mapping MIDI inputs to recorded note samples across different pitches, velocities, and playing styles, with interpolation to fill in missing variations. At the same time, this method provides high sound quality due to using real recordings, but it struggles to replicate complex polyphonic interactions like sympathetic resonance in pianos.
Neural Network model-based piano synthesis utilizes deep learning models to generate realistic piano sounds by learning complex mappings between input data and audio waveforms. Unlike traditional methods, these models do not rely on explicit signal processing or physics-based equations but instead learn patterns from large-scale datasets. However, they require extensive training data and high computational power and often struggle to capture complex dynamics. Recent research has tackled these challenges by exploring various neural architectures and training techniques. Dong et al. [17] introduced a novel system that adapts text-to-speech techniques to generate music performances from unaligned polyphonic scores utilizing a transformer-based encoder-decoder model. Cooper et al. [5] compared TTS configurations utilizing Tacotron 2 [18] as an acoustic model and neural source filter (NSF) [19] as waveform models. Shi et al. [20] expanded on this work by evaluating several configurations, including Tacotron and TransformerTTS [21] as waveform models, NSFs, and HiFi-GAN [22] as an acoustic model, introducing joint training to minimize feature mismatch. Additionally, Renault et al. presented a more complex approach requiring expert piano knowledge with DDSP-Piano v1 [6], which combines a signal-based model and a neural network-based model tailored explicitly for piano synthesis. Subsequently, Renault et al. developed DDSP-Piano v2 [7], integrating a new Differential Feedback Delay Network [23] to improve audio quality while optimizing for fewer parameters.

However, there is still a lack of neural network approaches that rely on a minimum of domain-specific expert knowledge during architecture design and training. As well as networks enabling real-time synthesis with the possibility of varying the synthesis sampling rates.

2.3. Piano Datasets

MAESTRO – MIDI and Audio Edited for Synchronous Tracks and Organization (MAESTRO) v3.0.0, introduced by Hawthorne et al. [13], is a dataset that contains paired MIDI data and high-quality audio recordings. The recordings are made with a Yamaha Disklavier during the International Piano-e-Competition and include 198.7 hours of sub-millisecond aligned MIDI and audio data. It is split into 159.2 hours of training data and 20.4 hours of test data. The audio is provided in 44.1 kHz, 16-bit PCM stereo format. Additionally, the dataset includes ten different recording years, containing different room acoustics and recording environments.
MAPS – MIDI Aligned Piano Sounds (MAPS), introduced by Emiya et al. [14], is a widely used piano database for multi-pitch estimation and automatic piano transcription. It consists of ∼65

[1]Audio Samples https://domdal.github.io/piano-ssm-samples/
[2]Github Repository https://github.com/domdal/piano-ssm

hours of MIDI-annotated piano recordings (44.1-kHz, 16-bit stereo) from both acoustic and software-based pianos, which are split into four sections each: ISOL set (isolated notes and monophonic excerpts), RAND set (chords with random pitch notes), UCHO set (usual chords from Western music), MUS set (pieces of piano music). This work uses the recordings using a Yamaha Disklavier Mark III in two distinct recording conditions, *Ambient* and *Close*. Since MAPS does not provide a predefined train-test split, the sections ISOL, RAND, and UCHO sets are used for training, and the MUS set is used for testing.

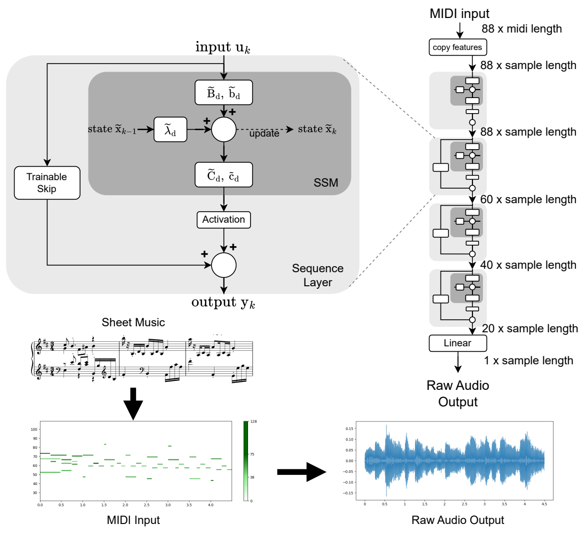

Figure 1: The Piano-SSM architecture. The model processes 88 MIDI input channels, which are progressively reduced through SSM layers to 20 latent dimensions. A final linear layer maps this representation to a single raw audio output channel.

3. DIAGONAL STATE SPACE MODELS FOR RAW PIANO AUDIO SYNTHESIS

This work proposes Piano-SSM, a multilayer SSM for the task of MIDI to raw audio waveform synthesis for real piano performances. The neural network architecture design enables end-to-end training without specific expert knowledge and utilizes a loss function combined of STFT, Mel-STFT, and Mean Loss. The approach allows for fast autoregressive synthesis and the SSM design also allows for synthesizing audio at different sampling rates.

3.1. Piano-SSM

Model Design – Piano-SSM builds up on the S-Edge [24] layer, which is modeled as a complex-valued continuous time MIMO LTI system,

$$\begin{aligned}\dot{\widetilde{\mathbf{x}}}(t) &= \widetilde{\mathbf{A}}\widetilde{\mathbf{x}}(t) + \widetilde{\mathbf{B}}\mathbf{u}(t) + \widetilde{\mathbf{b}} \\ \mathbf{y}(t) &= \Re(\widetilde{\mathbf{C}}\widetilde{\mathbf{x}}(t) + \widetilde{\mathbf{c}}),\end{aligned} \tag{1}$$

with the complex-valued hidden state $\widetilde{\mathbf{x}}(t) \in \mathbb{C}^{\mathrm{H}}$, a diagonal state matrix $\widetilde{\mathbf{A}} = \mathrm{diag}(\widetilde{\boldsymbol{\lambda}}) \in \mathbb{C}^{\mathrm{H}\times\mathrm{H}}$ with the diagonal elements $\widetilde{\boldsymbol{\lambda}} \in \mathbb{C}^{\mathrm{H}}$ directly representing the complex eigenvalues. The input matrix $\widetilde{\mathbf{B}} \in \mathbb{C}^{\mathrm{H}\times\mathrm{Y}}$, the input bias $\widetilde{\mathbf{b}} \in \mathbb{C}^{\mathrm{H}}$, the output matrix $\widetilde{\mathbf{C}} \in \mathbb{C}^{\mathrm{O}\times\mathrm{H}}$, and the output bias $\widetilde{\mathbf{c}} \in \mathbb{C}^{\mathrm{O}}$ are also complex valued. The inputs $\mathbf{u}$ and outputs $\mathbf{y}$ of an individual SSM layer are considered to be real-valued. Compared to S5-like representations we add bias units and allow for variable input and output dimensions. Additionally instead of modeling the eigenvalues in cartesian coordinates $\widetilde{\boldsymbol{\lambda}} = \boldsymbol{\alpha} + j\boldsymbol{\beta}$, we represent and optimize them in the polar coordinate form with log-norm $\mathbf{r}$ and angle $\boldsymbol{\phi}$. Specifically, we model the eigenvalues with $\widetilde{\boldsymbol{\lambda}} = e^{\mathbf{r}+j\boldsymbol{\phi}}$, which optimizes the norm in log-space. In order to ensure exponential stability we constrain the real part of the eigenvalues to be negative with $\Re(\widetilde{\boldsymbol{\lambda}}) =$ -abs$(\Re(\widetilde{\boldsymbol{\lambda}}'))$. MIMO SSM structures like S5, by default, apply a trainable multiplicative time scale parameter Δ to each eigenvalue. Since in cartesian coordinates, this has the effect of scaling the norm, we neglect it, as we already optimize on the norm.

During training and inference, we need to map the continuous time representation to the discrete version. The Piano-SSM layer does this with Zero Order Hold (ZOH). Due to the diagonal implementation of the state transition matrix discretization can be performed efficiently on an elementwise basis,

$$\begin{aligned}\begin{bmatrix}\widetilde{\mathbf{B}}_{\mathrm{d}} & \widetilde{\mathbf{b}}_{\mathrm{d}}\end{bmatrix} &= \mathrm{diag}\left(\widetilde{\boldsymbol{\lambda}}^{-1} \circ (\widetilde{\boldsymbol{\lambda}}_{\mathrm{d}} - 1)\right)\begin{bmatrix}\widetilde{\mathbf{B}} & \widetilde{\mathbf{b}}\end{bmatrix}, \\ \widetilde{\boldsymbol{\lambda}}_{\mathrm{d}} = e^{\widetilde{\boldsymbol{\lambda}}\mathrm{T_s}}, \quad \widetilde{\mathbf{C}}_{\mathrm{d}} &= \widetilde{\mathbf{C}}, \quad \widetilde{\mathbf{c}}_{\mathrm{d}} = \widetilde{\mathbf{c}}.\end{aligned} \tag{2}$$

By adjusting the sampling time ($\mathrm{T_s}$), the discrete SSM layers can operate at different effective sampling rates, which means we can train or network with, an original sampling rate $\mathrm{f_{train}}$ and then switch the system dynamics to generate audio with a lower sampling rate $\mathrm{f_{synth.}}$, by setting $\mathrm{T_s} = \mathrm{f_{train}}/\mathrm{f_{synth.}}$

The forward path of the discrete SSM is then defined by,

$$\begin{aligned}\widetilde{\mathbf{x}}_k &= \widetilde{\boldsymbol{\lambda}}_d \circ \widetilde{\mathbf{x}}_{k-1} + \widetilde{\mathbf{B}}_{\mathrm{d}}\mathbf{u}_k + \widetilde{\mathbf{b}}_{\mathrm{d}}, \\ \mathbf{y}_k &= \Re(\widetilde{\mathbf{C}}_{\mathrm{d}}\widetilde{\mathbf{x}}_k + \widetilde{\mathbf{c}}_{\mathrm{d}}) \\ \mathrm{Output} &= \mathrm{Skip}(\mathbf{u}_k) + \mathrm{Activation}(\mathbf{y}_k),\end{aligned} \tag{3}$$

where $\mathrm{Skip} \in \mathbb{R}^{\mathrm{O}\times\mathrm{Y}}$ is a trainable real-valued matrix enabling our MIMO SSM layer to have different input and output sizes. The number of parameters for the discrete representation follows with,

$$\text{Total Parameters} = \underbrace{2\mathrm{YH}}_{\widetilde{\mathbf{B}}} + \underbrace{2\mathrm{OH}}_{\widetilde{\mathbf{C}}} + \underbrace{\mathrm{YO}}_{\text{Skip}} + \underbrace{4\mathrm{H} + \mathrm{O}}_{\widetilde{\boldsymbol{\lambda}}_d,\text{ biases}}. \tag{4}$$

Figure 1 outlines the general structure of the proposed Piano-SSM architecture. It consists of six layers,

- Layer 1: Upsamples the MIDI features from midi rate to the audio sampling rate by ZOH.
- Layer 2: A single SSM Layer without a skip connection with 88 input and 88 output channels.
- Layer 3-5: Three stacked SSM layers (with Skip), reducing intermediate output channels from 88 to 20.
- Layer 6: A final fully connected linear layer reducing the last 20 channels to a single audio output channel.

For our experiments, we define three different Piano-SSM configurations (XL, L, S), which only differ in their internal state sizes H. The biggest configuration Piano-SSM XL with $\mathrm{H} = 256$, and 268.4k parameters. A large configuration Piano-SSM L with

$\mathrm{H} = 128$, and 142.4k parameters. The smallest configuration Piano-SSM S with $\mathrm{H} = 64$, and only 79.4k parameters. As an activation function, we use $\mathrm{Tanh}()$.

Model Interpretability – Theory stemming from linear control theory allows for analyzing dynamics of LTI systems. Piano-SSM can be seen as multiple linear IIR filters stacked with nonlinearities. For interpretability purposes, we only analyze the linear aspects within each layer. Due to the diagonal representation of our continuous time SSM layers the solution of the autonomous system (neglecting bias units and inputs) results in

$$\widetilde{\mathbf{x}}(t) = e^{\widetilde{\boldsymbol{\lambda}}t} \circ \widetilde{\mathbf{x}}_0 = e^{\boldsymbol{\alpha}t}\Big(\cos(\boldsymbol{\beta}t) + j\sin(\boldsymbol{\beta}t)\Big) \circ \widetilde{\mathbf{x}}_0, \qquad (5)$$

where $\widetilde{\mathbf{x}}(t)$ describes the temporal evolution of the states/memory based on an initial condition $\widetilde{\mathbf{x}}_0$, with the learned eigenvalues in cartesian coordinates $\widetilde{\boldsymbol{\lambda}} = \boldsymbol{\alpha} + j\boldsymbol{\beta}$. This representation shows how to interpret the learned state dynamics within each individual layer. The real parts of the learned complex eigenvalues give information about the exponential decay time constants $\boldsymbol{\tau} = 1/(\boldsymbol{\alpha}_{\mathbf{r}} \cdot \mathrm{f}_{\mathrm{train}})$. Smaller time constants result in faster state/memory decay. The frequencies $\mathbf{f} = (\boldsymbol{\beta} \cdot \mathrm{f}_{\mathrm{train}})/2\pi$ can be used to analyze the oscillation contained in the state evolution. Analyzing the eigenvalues in continuous time representation can give valuable insights into the learned dynamics for each individual layer. However, in the end, we discretize the continuous time SSM layers to get the autoregressive discrete inference model. Mapping from continuous to discrete can introduce aliasing effects, which also might alter the discrete dynamic behavior. Of special concern are those frequencies f (imaginary parts of eigenvalues) violating the Nyquist [25] criteria $f > f_{\mathrm{synth}}/2$. In Section 5.2 we show experiments for training networks with a high sampling rate and synthesizing audio with a lower sampling rate $f_{\mathrm{synth}} < f_{\mathrm{train}}$. We see that the number of aliased eigenvalues (imaginary parts violating Nyquist) correlates with the error induced due to discretization (see Table 3).

3.2. Audio Loss Functions

Classical point-wise loss functions such as L1-Loss (Mean Absolute Error) or L2-Loss (Mean Squared Error), often used to train sequential regression tasks, are not very good at capturing spectral details of raw audio signals due to their limitation in capturing perceptual differences. Moreover, two perceptually identical audio signals can still produce a high loss due to small phase shifts. This limitation motivates the use of spectral-based loss functions. Within our work, we use the Multi Scale Spectral Loss (MSSL), introduced by Engel et al. [2], as a metric for comparing synthesized audio with the real ground truth. Besides using this evaluation metric, we propose SMML for training our Piano-SSM. SMML is combined of a point-wise raw audio Mean Loss, the STFT Loss, and a Mel-STFT Loss. MSSL and SMML are briefly outlined in the following two subsections.

Multi Scale Spectral Loss

MSSL, introduced by Engel et al. [2], aims to better align with the human auditory perception. The loss is computed by the STFT with multiple sampling rate f_{s} dependent Fast Fourier Transform (FFT) window sizes $w_{f_{\mathrm{s}},i}$ as follows,

$$\mathrm{MSSL} = \sum_i \Big\| |X_i| - |Y_i| \Big\|_1 + \Big\| \log|X_i| - \log|Y_i| \Big\|_1. \qquad (6)$$

The loss is calculated as the L1 difference between the magnitude spectrograms of predictions $|X_i|$ and targets $|Y_i|$, along with the L1 difference between their logarithmic representations. Both $|X_i|$ and $|Y_i|$ are obtained by applying the STFT with a 75% overlap and a Hann window function. $\|\cdot\|_1$ states the L1 norm. The window sizes $w_{f_{\mathrm{s}},i}$ have to be adapted to the sampling rates. As an example, for 16 kHz we define the window sizes as follows:

$$w_{16\mathrm{k},i} \in \{4096, 2048, 1024, 512, 256, 128, 64\}$$

This multi-scale approach effectively captures spectral differences across various frequency resolutions and is commonly used to evaluate the quality of synthesized raw piano audio [1, 6].

STFT-Mel-Mean Loss

SMML combines three equally weighted loss terms to effectively capture different spectral properties of an audio signal. With SMML we aim to provide a balanced trade-off between frequency and time resolution while enhancing the stability of the neural network training. Initial experiments using MSSL as a loss function for our Piano-SSM revealed convergence issues. Consequently, we propose SMML as a more robust alternative to MSSL. The loss function is composed of the following three terms:

1. STFT Loss $= \Big\| |X| - |Y| \Big\|_1$, with $|X|$ and $|Y|$ being the magnitude spectrograms of the predicted and ground truth audio signals, obtained using STFT.

- Covers the full frequency spectrum up to the Nyquist [25] limit, with an FFT size equal to the sampling rate.
- A large window size improves frequency resolution but reduces time resolution (1s window, 100ms hop size).

2. Mel-STFT Loss $= \dfrac{\Big\| |Y_{\mathrm{mel}}| - |X_{\mathrm{mel}}| \Big\|_F}{\Big\| |Y_{\mathrm{mel}}| \Big\|_F} + \Big\| \log|X| - \log|Y| \Big\|_1$.

$|X_{\mathrm{mel}}|$ and $|Y_{\mathrm{mel}}|$ are the corresponding Mel-scaled STFT magnitudes, $\|\cdot\|_F$ is the Frobenius norm. Mel-STFT Loss was introduced by Steinmetz et al. [26].

- Mel-Scaling compresses high and expands low frequencies to approximate human hearing.
- A smaller window (46ms window, 5.8ms hop size) improves temporal resolution, making the loss sensitive to transient structures and fine timing differences.

3. Mean Loss $= \mathbb{E}\Big[\mathbb{E}[x]^2 - \mathbb{E}[y]^2\Big]$, where x and y are the predicted and ground truth audio signals.

- Ensures no drift towards a negative or positive mean.

4. EXPERIMENTAL SETUP

We perform experiments on MAESTRO [13] and MAPS [14] to evaluate our proposed Piano-SSM models. Table 1 show the train and test split audio durations in hours for the individual dataset configurations used during training and evaluation. We train our models in PyTorch and use the Adam optimizer with a learning rate of 1e−4 and a weight decay of 1e−4. The learning rate is dynamically adjusted using a cosine annealing schedule with a warmup phase, following the approach of Katsura et al. [27]. This scheduler enhances training stability by gradually increasing the learning rate during the warmup phase before transitioning into a cosine

Dataset	Split	Total Duration
MAPS Close	Train	**2.3 hours**
	Test	3.5 hours
MAPS Ambient	Train	**2.1 hours**
	Test	3.4 hours
MAESTRO	Train	159.2 hours
	Test	20.4 hours

Table 1: Total duration of datasets used for training and evaluation. MAPS datasets train on isolated notes and chords and evaluate on pieces of piano performances, while MAESTRO datasets contain complete piano performances for both training and evaluation

Model	Parameters	Model	Parameters
Piano-TTS v1 [5]	31.4M	Piano-TTS v2 [20]	31.5M
- Tacotron-2 [18]	30.6M	- Transformer-TTS [21]	17.6M
- NSF [19]	736.3k	- HiFi-GAN [22]	13.9M
DDSP-Piano v1 [6]	512.5k	DDSP-Piano v2 [7]	344.5k
- Sub-models	281.5k	- Sub-models	341.5k
- Tuning Models	33	- Tuning Models	70
- Reverb	240k	- FDN Reverb [23]	2820
		Piano-SSM XL ($H = 256$)	268.4k
		Piano-SSM L ($H = 128$)	142.4k
		Piano-SSM S ($H = 64$)	79.4k

Table 2: Model parameters for Piano-TTS, DDSP-Piano, and Piano-SSM (this work). Numbers for DDSP-Piano and Piano-TTS models are taken from Renault et al. [7].

decay pattern with periodic restarts. Training on the MAESTRO dataset needs 50 epochs, with a sample size of 50,000 samples per epoch. After training the models are finetuned using MSSL for 20 epochs with a learning rate of $2e{-}5$. When training on the MAPS dataset, we use a single linear output layer. Instead, when training our Piano-SSM on the MAESTRO dataset we employ a distinct linear layer per year, ensuring model adaptability across different years. All Piano-SSM models are trained using SMML Loss with a 4 second sample length and evaluated using the MSSL Loss with a 10 second sample length to ensure comparability to the DDSP-Piano models, which are also evaluated using a 10 second audio samples. The window sizes for MSSL are scaled accordingly to the sampling rate f_s, $w_{f_s,i} = \frac{s_r}{16\text{kHz}} w_{16\text{k},i}$. As an example, training the Piano-SSM XL model on audio with a sampling rate of 24kHz requires 116 hours of training on a single NVIDIA A100 80GB GPU for the MAESTRO dataset. The training setup, in general, uses a batch size of 8 and a sample length of 4 seconds.

For initializing the individual SSM layers we consider the following strategy. The H independent eigenvalues $\widetilde{\boldsymbol{\lambda}}$ are initialized with a linear interpolation between 0 and $\mathrm{H}/2$ with H steps for the imaginary parts and -0.5 for the real part and then elementwise scaled which $\boldsymbol{\Delta}$, where $\ln(\boldsymbol{\Delta})$ is initialized uniformly in the domain of $[\ln(0.001), \ln(0.1)]$. $\widetilde{\mathbf{B}}$ and $\widetilde{\mathbf{C}}$ are initialized orthogonally with a gain of $\sqrt{1/3}$. The real and imaginary parts are initialized separately. The input $\widetilde{\mathbf{b}}$ and output $\widetilde{\mathbf{c}}$ biases are uniformly $[0, 1]$ initialized for both real and imaginary parts. Accordingly, we also scale $\widetilde{\mathbf{B}}$, $\widetilde{\mathbf{b}}$ with the time scale parameter $\boldsymbol{\Delta}$.

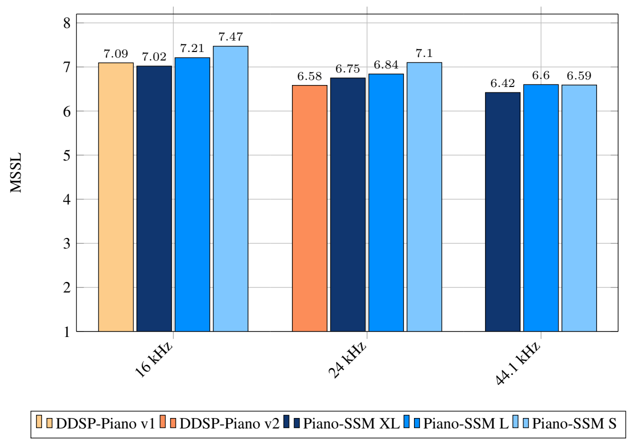

Figure 2: Comparison of MSSL values (lower is better) for Piano-SSM XL, L, S and DDSP-Piano models across different synthesis sampling rates. The Piano-SSM models were trained separately at each sampling rate, while DDSP-Piano v1 was trained at 16 kHz, and DDSP-Piano v2 at 24 kHz.

5. EXPERIMENTAL EVALUATIONS MAESTRO

We compare Piano-SSM XL with DDSP-Piano v1 [6] & v2 [7] on the MAESTRO v3.0.0 dataset [13] and evaluate the performance when training with audio data at different sampling rates such as 16kHz, 24kHz, and 44.1kHz (see Section 5.1).

Furthermore, we evaluate the Piano-SSM model's ability to train it on a high sampling rate and switch to a lower sampling rate for generation/synthesis. Without a need for retraining, this is achieved by changing the sampling rate $\mathrm{T_s} = \mathrm{f_{train}}/\mathrm{f_{synth.}}$ of all SSM layers when mapping from the continuous to discrete representation. Switching system dynamics introduces only a small loss in performance (see Section 5.2) but allows for faster audio generation if computational resources are limited (see Section 7).

5.1. Baseline Comparison

Table 2 shows parameter counts for state-of-the-art audio synthesis models and our three Piano-SSM configurations (XL, L and S, which only differ in the SSMs state sizes H). Piano-SSM is substantially smaller compared to the Piano-TTS models [20, 5], even in its largest configuration (Piano-SSM XL); it is slightly smaller than DDSP-Piano v1 and v2, while achieving similar synthesis quality (see Figure 2). Unfortunately we were not able to resynthesize or get access to raw audio files for Piano-TTS. Nevertheless, Piano-TTS parameter count is 117× Piano-SSM XL parameter count, and DDSP-Piano already outperformed TTS-Piano. Therefore, we only compare ourselfs with DDSP-Piano. For DDSP-Piano v1 & v2, we were able to resynthesize audios based on the code provided by the authors. We used this synthesized audio for evaluation.

The models are evaluated using the MSSL on different synthesis sampling rates. Figure 2 shows the MSSL scores for Piano-SSM and the DDSP-Piano v1 and v2 models. DDSP-Piano v1 was trained at 16kHz, while DDSP-Piano v2 was trained at 24kHz. In contrast, Piano-SSM was trained separately at each sampling rate to achieve a fair comparison, as the MSSL has a decreasing trend at higher sampling rates. The results indicate that Piano-

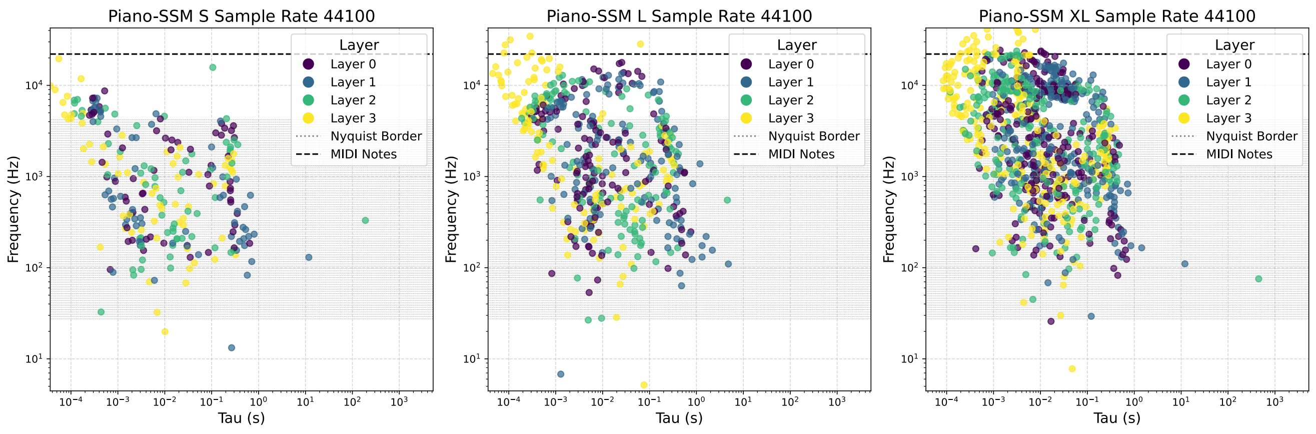

Figure 3: Frequencies $\mathbf{f}=(\boldsymbol{\beta}\ \cdot \mathrm{f_{train}})\ /\ 2\pi$ over exponential decay time constants $\boldsymbol{\tau}=1/(\boldsymbol{\alpha}_\mathbf{r}\cdot \mathrm{f_{train}})$ for all eigenvalues $\widetilde{\boldsymbol{\lambda}}=\boldsymbol{\alpha}+j\boldsymbol{\beta}$ across the layers of the Piano-SSM XL models trained on MAESTRO. Each point represents an eigenvalue in a specific layer, highlighting the learned dynamics. Smaller time constants $\boldsymbol{\tau}$ indicate faster decay of internal states/memory.

SSM XL maintains comparable synthesis quality to DDSP-Piano v1 and v2 while being more parameter-efficient and has a considerably simpler network architecture, which requires no expert knowledge about pianos.

5.2. Variable Synthesis Sampling Rates

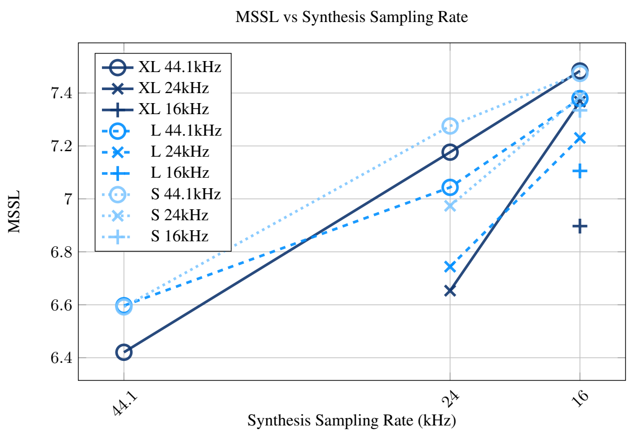

Figure 4: Comparison of MSSL values for different synthesis sampling rates (16 kHz, 24 kHz, and 44.1 kHz) of models trained and evaluated on the MAESTRO dataset. The plot shows results for three Piano-SSM model variants: XL (solid), L (dashed), and S (dotted). Each model variant was trained separately at 16kHz (+), 24kHz (×), and 44.1kHz (○). Lower MSSL values indicate better performance, with a general trend of increasing MSSL as the synthesis sampling rate decreases.

Here we show the effects of the Piano-SSM model's ability to synthesize audio at variable sampling rates. We evaluate the models trained at a high sampling rate and compare their MSSL when synthesizing audio at lower sampling rates. Specifically, we examine the effect of training at 44.1 kHz and synthesizing with 24 kHz and 16 kHz, as well as training at 24 kHz and synthesizing with 16 kHz. Figure 4 presents the MSSL values for the three Piano-SSM variants (XL, L, and S) on the MAESTRO dataset. The models were synthesized using the specified sampling rate. For MSSL evaluation we upsampled and evaluated audios at 44.1kHz to ensure a consistent basis for comparison. Additionally, to assess the impact of resampling, audio downsampled to 16kHz and then upsampled back to 44.1kHz was compared to the original 44.1kHz signal, resulting in an MSSL of 17.54. In comparison, audio resampled from 24kHz back to 44.1kHz yielded an MSSL of 9.6. To see the loss in performance when switching system dynamics for synthesizing with a lower sampling rate we compare with the baselines, where training and synthesis sampling rates are the same. The baselines have always lower MSSL, but the error incorporated due to system downsampling is acceptable.

Config.	Train SR [kHz]	Test SR [kHz]	Aliased Eigenvalues	MSSL Error
XL	44.1	24.0	13.09% (134/1024)	11.77%
XL	44.1	16.0	31.64% (324/1024)	16.55%
XL	24.0	16.0	11.23% (115/1024)	10.73%
L	44.1	24.0	6.64% (34/512)	6.77%
L	44.1	16.0	15.23% (78/512)	11.86%
L	24.0	16.0	6.64% (34/512)	7.21%
S	44.1	24.0	0.78% (2/256)	10.37%
S	44.1	16.0	2.34% (6/256)	13.38%
S	24.0	16.0	1.56% (4/256)	5.93%

Table 3: The increase in reconstruction error (MSSL Error) due to downsampling and connected aliasing effects across different Piano-SSM configurations and sampling rates (SR). The *Aliased Eigenvalues* column indicates the proportion and absolute number of eigenvalues violating the Nyquist frequency, leading to aliasing.

As explained in Section 3.1, we can investigate the learned state dynamics by analyzing the eigenvalues. Figure 3 shows frequencies $\mathbf{f}=(\boldsymbol{\beta}\ \cdot \mathrm{f_{train}})\ /\ 2\pi$ over exponential decay time constants $\boldsymbol{\tau}\ =\ 1/(\boldsymbol{\alpha}_\mathbf{r}\cdot \mathrm{f_{train}})$ of all the eigenvalues across the layers for the there Piano-SSM model configurations trained at $\mathrm{f_{train}} = 44.1$kHz. We observe that a decent amount of eigenvalues always lie within the MIDI frequency range, and the last layer always has the fastest dynamics with the highest frequency components and smallest decay rates. Further interpretability remains still a challenge due to the fact that this only interprets the linear aspects, and state dynamics do not incorporate input/output behavior. However, we are still able to investigate the amount of eigenvalues violating the Nyquist borders, having a strong influence on the discretization/aliasing effects. After training there are already some eigenvalues violating the Nyquist border (see Fig.3). In Tab. 3 we compare the increase in MSSL and also show the amount of eigenvalues violating the related Nyquist border. Results indicate a strong correlation between the increase in MSSL and the increase in aliased eigenvalues.

6. EXPERIMENTAL EVALUATIONS ON MAPS

Within this experiment, we consider training the Piano-SSM XL model solely on isolated notes and chords using the ISOL, RAND, and UCHO sets for the piano Yamaha Disklavier Mark III of the MAPS [14] dataset.

The evaluation is done on the MUS set of MAPS, which consists of pieces of piano performances. Training only on isolated notes and chords makes this an even more challenging task than the MAESTRO dataset. Table 1 shows the dataset durations in hours used for training and evaluation for the different datasets. We train three models for three recording conditions. *MAPS close*, *MAPS ambient* and their combination *MAPS Close & Ambient*. MAESTRO consists of complete piano performances and offers a larger training set compared to the MAPS dataset.

Table 4 shows MSSL scores evaluated for 10s synthesizing windows for models trained at 24kHz. Despite being trained only on single isolated notes and chords (ISOL, RAND, and UCHO sets of MAPS), the model was tested on full piano performances (MUS set of MAPS). This demonstrates its ability to generalize beyond the training data. We also show the results of a Piano-SSM XL model trained on the MAESTRO dataset but evaluated on the MUS set of MAPS. While the results of training on MAPS do not match those models trained on the MAESTRO dataset, they remain remarkable, given the limited training set.

Config.	**Train.**	**Eval.**	**MSSL** $\downarrow$
XL	Close	Close	8.71
XL	Ambient	Ambient	8.30
XL	Close & Ambient	Close & Ambient	8.23
XL	MAESTRO	Close & Ambient	7.95

Table 4: Piano-SSM XL evaluated for MAPS-MUS at 24kHz. Comparing three models trained on MAPS (Close, Ambient, Ambient & Close) with isolated notes and chords. Additionally, a model trained on MAESTRO. Lower ($\downarrow$) MSSL indicates better temporal and spectral alignment with ground truth audio.

7. AUTO-REGRESSIVE REAL TIME SYNTHESIS

A key advantage of SSMs is their ability to efficiently infer in an auto-regressive mode. For comparison, DDSP-Piano v2 reports a real-time factor (RTF) of 1.9±0.1 on a 2.6GHz Intel Xeon E5-2623 v4 CPU, evaluated on a Intel Core i7-12700 at $\sim$4.5GHz DDSP-Piano v2 achieves a RTF of 1.4±0.1 in multi threading mode and 2.8±0.1 in single core mode. We evaluate the RTF performance of Piano-SSM on a single core of an Intel Core i7-12700 at $\sim$4.5GHz, across multiple sampling rates and model sizes. Both CPUs support AVX2; they mostly differ in cache size and clock rate. The RTF is calculated as the ratio of synthesis time to actual audio duration, with RTF < 1.0 indicating real-time processing. All measurements are based on a custom C++17 header-only implementation optimized for low-latency audio synthesis, internally inferring one sample at a time. In addition to RTF, we report an average per-sample inference delay from input to output: $10\mu s$ for the XL model, $5\mu s$ for the L model, and $3\mu s$ for the S model. The results are shown in Table 5.

Config.	**SR** [kHz]	**RTF** $\downarrow$	**Workload** [GFLOP/s]	**Delay** [μs]
XL	44.1	0.44	23.105	10.1
L	44.1	0.22	11.996	4.9
S	44.1	0.12	6.442	2.6
XL	24.0	0.24	12.574	10.1
L	24.0	0.12	6.529	4.9
S	24.0	0.06	3.506	2.6
XL	16.0	0.16	8.383	10.1
L	16.0	0.08	4.352	4.9
S	16.0	0.04	2.337	2.6

Table 5: Real-time factor (**RTF**) of different Piano-SSM **Configurations** at different sampling rates **SR**, measured on a Intel Core i7-12700 Processor. Lower ($\downarrow$) RTF indicate faster inference time. The **Workload** required to synthesize one second of raw audio is reported, along with the average per-sample inference **Delay** from input to output.

8. CONCLUSION

This work introduces Piano-SSM, a novel neural network architecture enabling causal real-time piano audio synthesis at different sampling rates. Our models cover a parameter range of 268.4k–79.4k. The largest model allows for synthesizing one second of audio at 44.1kHz in 0.44s with a workload of 23.1GFLOPS/s and an 10.1μs input/output delay. The fastest model at 16kHz only needs 0.04s with 2.3GFLOP/s and 2.6μs input/output delay, using an Intel Core i7-12700 processor at 4.5GHz with single-core inference. Benchmarks on the Maestro dataset show comparable results to state-of-the-art DDSP-Piano v1 and DDSP-Piano v2. Evaluations with the MAPS dataset also show that reasonable results can be achieved with a very limited amount of training data. A key feature of the proposed SSM based architecture is the ability to train at high sampling rates while synthesizing at lower rates without a need for retraining. We utilize the interpretability capabilities of SSMs to show the correlation between performance loss and aliasing effects when synthesizing at lower sampling rates. Future work will explore how aliasing effects can be more effectively addressed during training to reduce downsampling errors.

9. ACKNOWLEDGMENTS

This work is supported by the Austrian Federal Ministry for Digital and Economic Affairs, the National Foundation for Research, Technology, and Development, the Christian Doppler Research Association.

10. REFERENCES

[1] Ben Hayes, Jordie Shier, György Fazekas, Andrew McPherson, and Charalampos Saitis, "A review of differentiable digital signal processing for music and speech synthesis," *Frontiers in Signal Processing*, vol. 3, pp. 1284100, 2024.

[2] Jesse H. Engel, Lamtharn Hantrakul, Chenjie Gu, and Adam Roberts, "DDSP: differentiable digital signal processing," in *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*, 2020.

[3] Rodrigo Castellon, Chris Donahue, and Percy Liang, "Towards realistic midi instrument synthesizers," in *NeurIPS Workshop on Machine Learning for Creativity and Design*, 2020.

[4] Yusong Wu, Ethan Manilow, Yi Deng, Rigel Jacob Swavely, Kyle Kastner, Tim Cooijmans, Aaron Courville, Anna Huang, and Jesse Engel, "Midi-ddsp: Hierarchical modeling of music for detailed control," in *Proceedings of the Tenth International Conference on Learning Representations (ICLR)(Online)(2022 Apr.)*, 2022.

[5] Erica Cooper, Xin Wang, and Junichi Yamagishi, "Text-to-Speech Synthesis Techniques for MIDI-to-Audio Synthesis," Feb. 2022, arXiv:2104.12292 [cs].

[6] Lenny Renault, Rémi Mignot, and Axel Roebel, "Differentiable piano model for midi-to-audio performance synthesis," in *25th International Conference on Digital Audio Effects (DAFx20in22)*, 2022.

[7] Lenny Renault, *Neural audio synthesis of realistic piano performances*, Theses, Sorbonne Université, July 2024, Issue: 2024SORUS196.

[8] Albert Gu, Karan Goel, and Christopher Ré, "Efficiently modeling long sequences with structured state spaces," in *The International Conference on Learning Representations*, 2022.

[9] Albert Gu, Ankit Gupta, Karan Goel, and Christopher Ré, "On the parameterization and initialization of diagonal state space models," in *Proceedings of the 36th International Conference on Neural Information Processing Systems*, Red Hook, NY, USA, 2024, NIPS '22, Curran Associates Inc.

[10] Ankit Gupta, Albert Gu, and Jonathan Berant, "Diagonal state spaces are as effective as structured state spaces," in *Proceedings of the 36th International Conference on Neural Information Processing Systems*, Red Hook, NY, USA, 2024, NIPS '22, Curran Associates Inc.

[11] Jimmy T.H. Smith, Andrew Warrington, and Scott Linderman, "Simplified state space layers for sequence modeling," in *The Eleventh International Conference on Learning Representations*, 2023.

[12] Karan Goel, Albert Gu, Chris Donahue, and Christopher Ré, "It's raw! audio generation with state-space models," in *International conference on machine learning*. PMLR, 2022, pp. 7616–7633.

[13] Curtis Hawthorne, Andriy Stasyuk, Adam Roberts, Ian Simon, Cheng-Zhi Anna Huang, Sander Dieleman, Erich Elsen, Jesse Engel, and Douglas Eck, "Enabling Factorized Piano Music Modeling and Generation with the MAESTRO Dataset," Jan. 2019, arXiv:1810.12247 [cs].

[14] Valentin Emiya, Nancy Bertin, Bertrand David, and Roland Badeau, "MAPS - A piano database for multipitch estimation and automatic transcription of music," Research Report, July 2010.

[15] Shida Wang and Beichen Xue, "State-space models with layer-wise nonlinearity are universal approximators with exponential decaying memory," in *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

[16] Albert Gu, Isys Johnson, Aman Timalsina, Atri Rudra, and Christopher Re, "How to train your HIPPO: State space models with generalized orthogonal basis projections," in *International Conference on Learning Representations*, 2023.

[17] Hao-Wen Dong, Cong Zhou, Taylor Berg-Kirkpatrick, and Julian McAuley, "Deep Performer: Score-to-Audio Music Performance Synthesis," Feb. 2022, arXiv:2202.06034 [cs].

[18] Jonathan Shen, Ruoming Pang, Ron J. Weiss, Mike Schuster, Navdeep Jaitly, Zongheng Yang, Zhifeng Chen, Yu Zhang, Yuxuan Wang, R. J. Skerry-Ryan, Rif A. Saurous, Yannis Agiomyrgiannakis, and Yonghui Wu, "Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions," Feb. 2018, arXiv:1712.05884 [cs].

[19] Xin Wang, Shinji Takaki, and Junichi Yamagishi, "Neural source-filter waveform models for statistical parametric speech synthesis," Nov. 2019, arXiv:1904.12088 [eess].

[20] Xuan Shi, Erica Cooper, Xin Wang, Junichi Yamagishi, and Shrikanth Narayanan, "Can Knowledge of End-to-End Text-to-Speech Models Improve Neural MIDI-to-Audio Synthesis Systems?," Mar. 2023, arXiv:2211.13868 [cs].

[21] Naihan Li, Shujie Liu, Yanqing Liu, Sheng Zhao, Ming Liu, and Ming Zhou, "Neural Speech Synthesis with Transformer Network," Jan. 2019, arXiv:1809.08895 [cs].

[22] Jungil Kong, Jaehyeon Kim, and Jaekyoung Bae, "HiFi-GAN: Generative Adversarial Networks for Efficient and High Fidelity Speech Synthesis," Oct. 2020, arXiv:2010.05646 [cs].

[23] Sungho Lee, Hyeong-Seok Choi, and Kyogu Lee, "Differentiable Artificial Reverberation," July 2022, arXiv:2105.13940 [cs].

[24] Matthias Bittner, Daniel Schnöll, Matthias Wess, and Axel Jantsch, "Efficient and interpretable raw audio classification with diagonal state space models," *Machine Learning*, vol. 114, no. 8, pp. 175, Jun 2025.

[25] C. E. Shannon, "Communication Theory of Secrecy Systems*," *Bell System Technical Journal*, vol. 28, no. 4, pp. 656–715, Oct. 1949.

[26] C. Steinmetz and Joshua D. Reiss, "auraloss: Audio-focused loss functions in PyTorch," 2020.

[27] Naoki Katsura, "Cosine Annealing with Warmup for PyTorch," `https://github.com/katsura-jp/pytorch-cosine-annealing-with-warmup`, Mar. 2025.