TECHNISCHE UNIVERSITÄT WIEN

Institut für Computertechnik
Institute of Computer Technology

A MASTER THESIS ON

# Raw Audio Piano Synthesis with Structured State Space Models

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF

Diplom-Ingenieur
(Equivalent to Master of Science)

in

Embedded Systems (066 504)

by

**Dominik Dallinger**

01529357

**Supervisor(s):**

Univ.Prof. Dipl.-Ing. Dr.techn. Axel Jantsch
Projektass. Dipl.-Ing. Matthias Bittner, MSc BSc
Projektass. Dipl.-Ing. Daniel Schnöll, BSc

Vienna, Austria
April 2025

# Abstract

This thesis introduces Piano-SSM, a novel Structured State Space Model (SSM) architecture for real-time raw piano audio synthesis. Unlike conventional neural audio synthesis models, Piano-SSM focuses on computational efficiency by utilizing the advantages of SSMs, such as linear computational complexity with the sequence length and constant memory consumption. The proposed model synthesizes audio directly from Musical Instrument Digital Interface (MIDI) input. The network requires no intermediate representations in the form of spectral representations or domain-specific expert knowledge, simplifying training and improving accessibility. Evaluations on the MIDI and Audio Edited for Synchronous TRacks and Organization (MAESTRO) dataset show that Piano-SSM achieves a Multi-Scale Spectral Loss (MSSL) comparable to state-of-the-art models. Moreover, evaluations on the MIDI Aligned Piano Sounds (MAPS) dataset demonstrate the model's generalization capabilities when trained on a dataset with very limited data. Further experiments on the MAESTRO dataset highlight the model's ability to be trained on a high sampling rate while synthesizing on lower sampling rates. Finally, utilizing a custom C++ implementation, the thesis demonstrates Piano-SSM's ability to synthesize high-quality piano audio in real-time.

# Kurzfassung

Diese Arbeit stellt Piano-SSM vor, eine neuartige Structured State Space Model (SSM)-Architektur, welches in Echtzeit, Roh-Klavieraudiodaten synthetisiert. Im Gegensatz zu konventionellen neuronalen Audiosynthesemodellen konzentriert sich Piano-SSM auf die Recheneffizienz, indem es die Vorteile von SSMs, wie z.B. lineare Rechenkomplexität (welche mit der Eingangssequenzlänge skaliert) und konstanten Speicherverbrauch, nutzt und so Echtzeit-Inferenz ohne Einschränkung der Audioqualität ermöglicht. Das vorgestellte Modell synthetisiert Audiosignale direkt aus der Musical Instrument Digital Interface (MIDI)-Eingabe. Das Netzwerk benötigt keine Zwischenrepräsentationen wie Spektraldarstellungen oder domänenspezifisches Expertenwissen, was das Training vereinfacht und die Zugänglichkeit verbessert. Auswertungen des MIDI and Audio Edited for Synchronous TRacks and Organization (MAESTRO)-Datensatzes zeigen, dass Piano-SSM einen Multi-Scale Spectral Loss (MSSL) erreicht, der mit den Modellen auf dem neuesten Stand der Forschung vergleichbar ist. Darüber hinaus zeigen die Auswertungen auf dem MIDI Aligned Piano Sounds (MAPS)-Datensatz die Generalisierungsfähigkeiten des Modells, wenn es auf einem Datensatz mit sehr begrenzten Daten trainiert wurde. Weitere Experimente mit dem MAESTRO-Datensatz unterstreichen die Fähigkeit des Modells, mit einer hohen Abtastrate trainiert zu werden und gleichzeitig mit niedrigeren Abtastraten zu synthetisieren. Durch die Verwendung einer benutzerdefinierten C++Implementierung wird die Echtzeitfähigkeit des Modells durch den Vergleich der Echtzeitfaktoren der verschiedenen Modellkonfigurationen gezeigt. Die experimentellen Ergebnisse zeigen, dass Piano-SSM in der Lage ist, hochwertige Klavierklänge in Echtzeit zu synthetisieren.

# Preface

The research presented in this thesis is presented in a more concise form as a paper **"Piano-SSM: Diagonal State Space Models for Efficient MIDI-to-Raw Audio Synthesis"** by, myself, Dallinger et al. [1] (currently under submission). The thesis provides a more detailed and comprehensive treatment of the subject, including expanded discussions, background, and additional results that could not be included in the short format of the paper. Additionally, the paper builds upon the thesis by introducing an interpretability section focused on analyzing aliasing errors.

The inference runntime measurements presented in Chapter 5.4 are based on a custom, header-only C++17 implementation presented in the paper **"Towards Optimal Implementations of Neural Networks on Micro-Controller"** by Schnöll et al. [2]. The specific implementation extending the support of State Space Model inference used in this thesis is based on the version presented in the paper **"Efficient and Interpretable Raw Audio Classification with Diagonal State Space Models"** by Bittner and Schnöll et al. [3].

Following tools have been used in addition to write this thesis:

**Grammarly** [1]

Grammarly has been used to spellcheck, grammar-check, and enhance the clarity of the written text by their generative AI based application.

[1] https://www.grammarly.com/

# *Erklärung*

*Hiermit erkläre ich, dass die vorliegende Arbeit ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt wurde. Die aus anderen Quellen oder indirekt übernommenen Daten und Konzepte sind unter Angabe der Quelle gekennzeichnet.*

*Die Arbeit wurde bisher weder im In- noch im Ausland in gleicher oder in ähnlicher Form in anderen Prüfungsverfahren vorgelegt.*

# Copyright Statement

I, Dominik Dallinger, hereby declare that this thesis is my own original work and, to the best of my knowledge and belief, it does not:

- Breach copyright or other intellectual property rights of a third party.
- Contain material previously published or written by a third party, except where this is appropriately cited through full and accurate referencing.
- Contain material which to a substantial extent has been accepted for the qualification of any other degree or diploma of a university or other institution of higher learning.
- Contain substantial portions of third party copyright material, including but not limited to charts, diagrams, graphs, photographs or maps, or in instances where it does, I have obtained permission to use such material and allow it to be made accessible worldwide via the Internet.

| | | |
|---|---|---|
| SIGNATURINFORMATIONEN | **Unterzeichner** | Dominik Michael Dallinger |
| | **Datum/Zeit-UTC** | 2025-04-08T10:32:51+02:00 |
| | **Prüfinformation** | Informationen zur Prüfung der elektronischen Signatur finden Sie unter: https://www.signaturpruefung.gv.at |
| **Hinweis** | Dieses mit einer qualifizierten elektronischen Signatur versehene Dokument hat gemäß Art. 25 Abs. 2 der Verordnung (EU) Nr. 910/2014 vom 23. Juli 2014 ("eIDAS-VO") die gleiche Rechtswirkung wie ein handschriftlich unterschriebenes Dokument. | |

Signature: _____

Vienna, Austria, April 2025 Dominik Dallinger

# Acknowledgment

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.

# Dedication

Four years ago (in 2021), before I started working in the Christian Doppler Laboratory for Embedded Machine Learning as a Research Student Assistant, I never thought of working in such an interesting and promising field like Machine Learning. I want to thank my professor, Axel Jantsch, for this great opportunity, which has not only shaped my academic journey but also opened the door to future innovations and meaningful contributions in this field. Moreover, this employment allowed me to get to know my colleagues Matthias Wess, Matthias Bittner, David Breuss, and Daniel Schnöll, who are not only the best colleagues, always willing to tackle my countless questions, but have also become real friends whom I don't want to miss. So thank you all for your support and the great time we spent together. Last but not least, I want to thank my parents, Birgit and Thomas, for always supporting me (not only financially but also in every other possible way). Moreover, I want to thank my girlfriend, Elise, for her patience, encouragement, and her understanding of my frequent late-night work sessions. Ending this with a quote from Einstein that, to my knowledge, perfectly describes the entire field of Machine Learning:

*"The more I learn, the more I realize how much I don't know."*

# Contents

# List of Tables

# List of Figures

# Acronyms

**AAC** Advanced Audio Coding. 13

**AIFF** Audio Interchange File Format. 14

**CNN** Convolutional Neural Network. xiii, 1, 2, 3, 16

**DDSP** Differentiable Digital Signal Processing. xiii, xvi, 17, 34, 35, 42

**FDN** Feedback Delay Network. xvii, 17, 35, 39, 40

**FFT** Fast Fourier Transformation. xiii, xv, xvi, 9, 19, 20, 28, 29, 30, 31, 32

**IIR** Infinite Impulse Response. 9, 42

**MAE** Mean Absolute Error. 8, 19, 28

**MAESTRO** MIDI and Audio Edited for Synchronous TRacks and Organization. iii, iv, xiii, xvi, xvii, 8, 15, 16, 17, 27, 33, 34, 37, 38, 39, 40, 41, 44, 47, 48

**MAPS** MIDI Aligned Piano Sounds. iii, iv, xiii, xvii, 15, 17, 18, 19, 33, 40, 41, 47

**MIDI** Musical Instrument Digital Interface. iii, iv, xi, xiii, xv, xvi, xvii, 1, 3, 4, 8, 11, 12, 13, 15, 16, 17, 18, 23, 25, 26, 30, 31, 36, 43, 44, 45, 47

**MIMO** Multiple Input Multiple Output. 16, 23

**MOS** Mean Opinion Score. 17

**MSE** Mean Squared Error. 8, 16, 19, 28

**MSSL** Multi-Scale Spectral Loss. iii, iv, xii, xvi, xvii, 15, 17, 19, 23, 28, 32, 33, 34, 35, 37, 38, 39, 40, 41

**MSTM** Mean-STFT-Mel. xii, 9, 23, 28, 32, 34

**NN** Neural Network. xiii, xv, 1, 2, 3, 4, 5, 8, 12, 15, 17, 23, 26, 32

**RNN** Recurrent Neural Network. 16

**S2S** Sequence to Sequence. 1, 2, 8

**SISO** Sinlge Input Single Output. 16

**SSM** Structured State Space Model. iii, iv, xi, xiii, xv, xvi, xvii, 1, 2, 3, 5, 6, 7, 8, 9, 15, 16, 23, 24, 25, 26, 32, 34, 35, 36, 42, 43, 45, 47

**STFT** Short-Time Fourier Transform. xii, xv, xvi, 9, 19, 20, 23, 28, 30, 31, 32

**TTS** Text-To-Speech. xii, xiii, xv, xvi, 2, 5, 17, 34, 35, 36

**WAV** Waveform Audio File Format. 12, 14

**ZOH** Zero-Order Hold. 6, 24, 25

# Chapter 1

# Introduction

Synthesizing raw audio in real-time is challenging due to the complex temporal dependencies in sound generation. Deep learning models, such as Transformers [9] or Convolutional Neural Network (CNN) [10], have succeeded in neural audio synthesis but often come with the disadvantage of high computational costs, making real-time processing difficult. Structured State Space Models (SSMs) [3, 11–13] offers a promising alternative by efficiently handling sequential data with lower memory and computational requirements [11]. This work explores the area of Sequence to Sequence (S2S) modeling for raw audio piano synthesis using SSM-based models for real-time piano synthesis. Unlike conventional approaches that rely on spectrogram-based intermediate representations, the proposed method enables direct Musical Instrument Digital Interface (MIDI)-to-raw-audio synthesis. The proposed model reduces complexity without compromising high-quality audio output by leveraging the advantages of a SSMs-based architecture. The thesis also addresses key challenges, such as input representation, model optimization, and loss function design, to provide a lightweight and interpretable solution for real-time neural piano synthesis.

The following chapters give an overview of sequence-to-sequence modeling and categorize different state-of-the-art methods used in piano synthesis. Additionally, it discusses the fundamentals of SSMs used as Neural Network (NN) and explains their transformation into diagonal form to improve efficiency. The chapter is organized into the following sections:

- **Sequence-to-Sequence Modeling**: This section discusses how sequence-to-sequence models, originally developed for natural language processing, can be adapted for audio synthesis. It compares different NN architectures and highlights the use of SSMs to handle complex audio sequences effectively.
- **Piano Synthesis**: This part reviews the latest techniques in piano synthesis, comparing data-

driven approaches with parametric methods.

- **Structured State Space Models**: This section explains the fundamentals of SSMs, including the diagonal form of SSMs to enhance computational efficiency
- **Research Question and Contribution**: This final section presents the research question addressed by the thesis and shows the contributions of the proposed model.

## 1.1 Sequence-to-Sequence Modeling

Sequence to Sequence (S2S) modeling is commonly known for its application in the field of neural language processing [14]. These models aim to model a sequence from one domain to another, typically using a two-part architecture consisting of encoder and decoder. S2S models, used in a wide range of applications, were initially developed for language-to-language translation as seen in Figure 1.1. However, they are most known for their use in chatbots and text summarization.

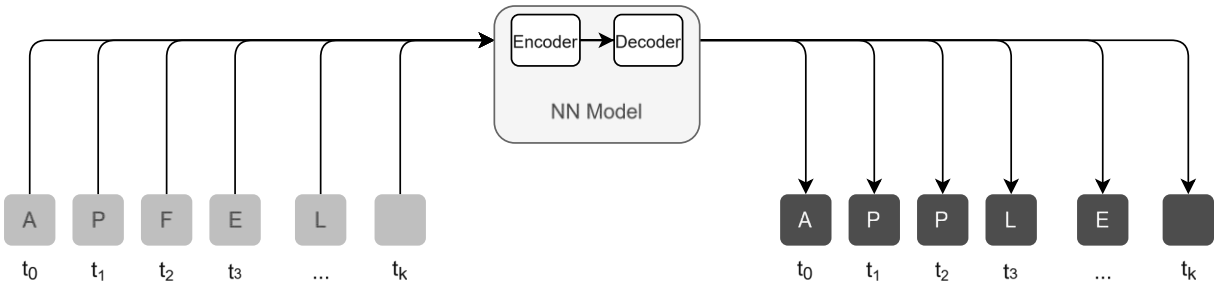



Figure 1.1: Illustration of a classical Encoder-Decoder design for Sequence-to-Sequence Modeling, translating the input 'APFEL' to 'APPLE'

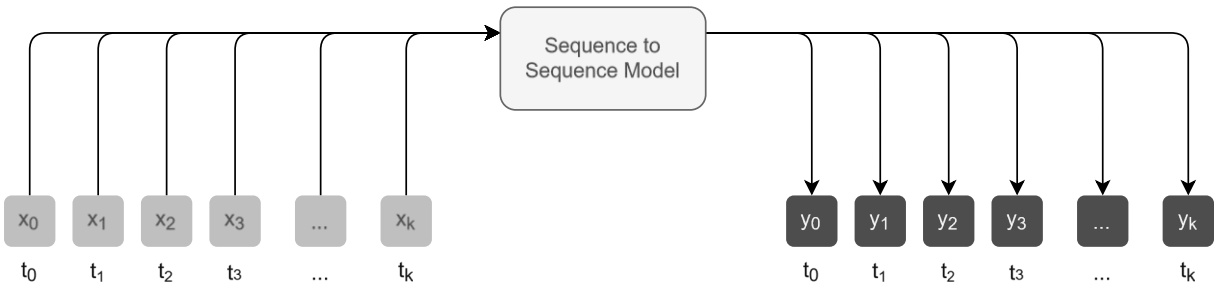



Figure 1.2: Illustration of a Sequence-to-Sequence Model design

There are also more unknown tasks for S2S models in the time series domain, including Text-To-Speech (TTS), audio generation, or audio synthesis. Figure 1.2 shows a S2S model, which models a time series sequence to another. Current NN-based audio synthesis approaches include CNNs [10] and Transformer [9] based models. This thesis proposed to use state space model-based audio synthesis. As shown in Table 1.1, SSMs have many advantages compared to other NN architectures like Transform-

ers or CNNs. Transformers are very powerful in handling data dependencies due to their self-attention mechanism. However, they usually have millions of parameters contributing to their computational intensity. Due to their high computational costs, which scale quadratically $O(T^2)$ with the sequence length $T$, they are less suitable for real-time applications. Moreover, Transformer model-based audio synthesis is often a two-stage process: an acoustic model predicts a time-frequency representation from MIDI input, and then a vocoder model reconstructs the final waveform. This two-part training setup increases complexity and computational cost. CNNs are often known for their complex network structures, and high computational demand. However, they also struggle with temporal dynamics, making them unsuitable for real-time applications and audio synthesis. SSMs are known for their lower computational demand, which scales linear $O(T)$ with the sequence length $T$ and constant memory consumption independent of the sequence length, making them suitable for real-time applications. Moreover, SSMs enable end-to-end training, synthesizing raw audio directly from MIDI input without requiring intermediate representations such as spectrograms. This approach reduces complexity and enables real-time performance, making SSMs a promising candidate for neural audio synthesis.

| Feature | Transformers | CNNs | Structured State Space Models |
|---|---|---|---|
| Computational Demand | Very high | High | Moderate to low |
| Memory Efficiency | Low | Moderate | High |
| Real-time Capability | Low | Moderate | High |
| Temporal Dynamics | Moderate | Poor | Excellent |
| Model Interpretability | Low (black box) | Low (black box) | High |

Table 1.1: Comparison of different NN architectures focusing on Transformers, CNNs, and Structured State Space Model

## 1.2 Piano Synthesis

Synthesis of raw audio is a challenging task due to the complex dynamics of sound waves. As stated in Hayes et al. [15], piano synthesis can be categorized into two main categories: Data-Driven synthesis and Parametric Piano synthesis. This section will briefly present the methods of the two categories.

### 1.2.1 Parametric Piano Synthesis

Parametric models in piano sound synthesis include prior knowledge in their mathematical formulation, reducing the number of parameters that need to be modified while simultaneously specializing the model for specific instruments. As a result, they are less adaptable for instruments (or different pianos) that do not align with the embedded prior information. Both physical and signal-based models

can be categorized as parametric models, but they differ in the amount of knowledge utilized and their general applicability.

- **Signal model-based piano synthesis**: Signal model-based audio synthesis is based on techniques that generate sound by explicitly modeling the physical or mathematical properties of the underlying signal. Signal models describe sound as a time-domain or frequency-domain function described by frequency, amplitude, phase, and spectral content parameters. Designing and parameterizing such models requires expert knowledge involving complex mathematical formulations and signal-processing techniques.
- **Physical model-based piano synthesis**: Physical model-based audio synthesis simulates the behavior of musical instruments by formulating and solving equations that model the underlying 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.

### 1.2.2 Data-Driven Piano Synthesis

Data-driven models focus more on reproducing musical instruments' sound output by utilizing a large amount of audio recordings. Both concatenative model-based piano synthesis and NN model-based piano synthesis require a large data quantity for their operation/training, with the advantage that they need little a priori knowledge, hardly any piano domain-specific knowledge and are therefore highly adaptable to different instruments.

- **Concatenative model-based piano synthesis**: Concatenative model-based piano synthesis is widely used in digital pianos and other instruments. It involves mapping MIDI inputs to recorded note samples across different pitches, velocities, and playing styles, with interpolation to fill in missing variations. In contrast, this method provides high sound quality due to using real recordings, but it struggles to replicate complex polyphonic interactions such as the natural resonance of pianos.
- **Neural Network model-based piano synthesis**: NN-based piano synthesis utilizes deep learning models to synthesize 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. The synthesis process often follows a two-stage pipeline as shown in Figure 1.3: an *acoustic model* predicts a time-frequency representation from MIDI input, while a *vocoder model* reconstructs the final waveform. Neural approaches offer high-quality audio, different instrument adaptability, and

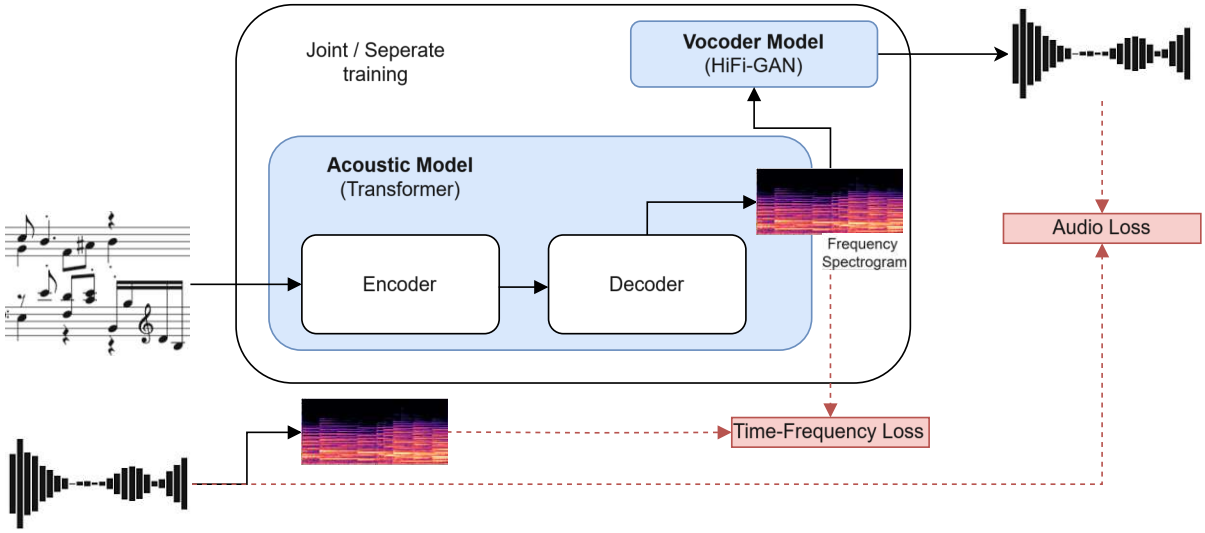



Figure 1.3: Architecture of a state-of-the-art NN model-based piano synthesis [5] [6]. The Architecture shows a joint training of a Transformer-TTS [7] acoustic model and a HiFiGan [8] vocoder model. Frequency spectrograms are used as an intermediate audio representation between the acoustic and vocoder models.

reduced domain-specific knowledge. However, they require extensive training data and high computational power and, therefore, are not usable for real-time applications

## 1.3 Structured State Space Model (SSM)

Structured State Space Models (SSMs) can be used to model sequential processes. They are traditionally used in the control theory to model linear continuous dynamical systems as first-order differential equations via state variables. Their discrete representation can be used as an autoregressive model in machine learning. A SSM consists of two main components.

- State Equations: These describe the evolution of the system's hidden, internal state over time. It accounts for the influence of the previous state and external inputs.
- Observation Equations: These detail the connection between the system's internal state and the input data.

$$\begin{aligned} \widetilde{\mathbf{x}}'(t) &= \widetilde{\mathbf{A}}\widetilde{\mathbf{x}}(t) + \widetilde{\mathbf{B}}\mathbf{u}(t) \\ \mathbf{y}(t) &= \Re(\widetilde{\mathbf{C}}\widetilde{\mathbf{x}}(t) + \mathbf{D}\mathbf{u}(t)) \end{aligned} \tag{1.1}$$

Equation 1.1 shows the state and output equation of a complex-valued continuous-time SSM representation (as commonly used within the machine learning community). The vector $\widetilde{\mathbf{x}}(t) \in \mathbb{C}^n$ represents the $n$ state variables with $\widetilde{\mathbf{x}}'(t) \in \mathbb{C}^n$ as its derivation. The vector $\mathbf{u}(t) \in \mathbb{R}^m$ represents the input with $m$ as input dimension and $\mathbf{y}(t) \in \mathbb{R}^p$ represents the output with $p$ as the output dimension. The

matrices are defined by $\widetilde{\mathbf{A}} \in \widetilde{\mathbb{C}}^{n\times n}$ as the state matrix, $\widetilde{\mathrm{B}} \in \mathbb{C}^{n\times m}$ as the input matrix, $\widetilde{\mathbf{C}} \in \mathbb{C}^{p\times n}$ as the output matrix and $\mathbf{D} \in \mathbb{R}^{p\times m}$ as the feed-through matrix. Figure 1.4 shows a graphical representation of the state and output equations of the continuous-time SSM.

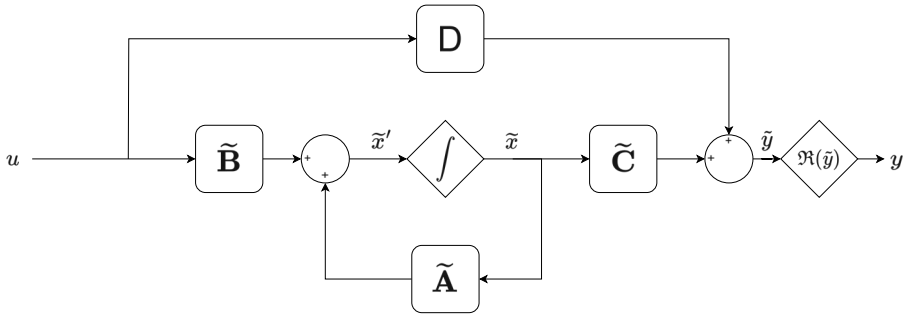



Figure 1.4: Block diagram representation of the linear state-space equations commonly used in machine learning. It maps a real input to a real output with a complex valued SSM.

Since the digital world is not in a continuous-time domain (as the name already states), the state-space representation has to be transformed into a discrete representation using Zero-Order Hold (ZOH). The system transforms accordingly to

$$\begin{aligned} \widetilde{\mathbf{x}}[k+1] &= \widetilde{\mathbf{A}}_{\mathrm{d}}\widetilde{\mathbf{x}}[k] + \widetilde{\mathbf{B}}_{\mathrm{d}}\mathbf{u}[k] \\ \mathbf{y}[k] &= \Re(\widetilde{\mathrm{C}}_{\mathrm{d}}\widetilde{\mathbf{x}}[k] + \mathrm{D}_{\mathrm{d}}\mathbf{u}[k]), \end{aligned} \tag{1.2}$$

where $\widetilde{\mathbf{A}}_{\mathrm{d}}, \widetilde{\mathbf{B}}_{\mathrm{d}}, \widetilde{\mathbf{C}}_{\mathrm{d}}$ and $\mathbf{D_d}$ are the discrete-time system matrices.
The transformation from continuous-time to discrete-time is given by,

$$\widetilde{\mathbf{A}}_{\mathrm{d}} = e^{\widetilde{\mathbf{A}}\mathrm{T_s}}, \quad \widetilde{\mathbf{B}}_{\mathrm{d}} = \left(\int_0^{\mathrm{T_s}} e^{\widetilde{\mathbf{A}}\tau} d\tau\right) \widetilde{\mathbf{B}}, \quad \widetilde{\mathbf{C}}_{\mathrm{d}} = \widetilde{\mathbf{C}}, \quad \mathbf{D_d} = \mathbf{D}. \tag{1.3}$$

$\widetilde{\mathbf{A}}_{\mathrm{d}}$ is obtained from the matrix exponential of the continuous-time matrix with sampling rate $\mathrm{T_s}$. The matrix exponential is computed using its Taylor series expansion $e^{\widetilde{\mathbf{A}}\mathrm{T_s}} = \sum_{k=0}^{\infty} \frac{(\widetilde{\mathbf{A}}\mathrm{T_s})^k}{k!}$. Matrix $\widetilde{\mathbf{B}}_{\mathrm{d}}$ is obtained by integrating over the sample interval $\mathrm{T_s}$ using the matrix exponential of $\widetilde{\mathbf{A}}$ multiplied by the continuous input matrix $\widetilde{\mathbf{B}}$. $\widetilde{\mathbf{C}_{\mathrm{d}}}$ and $\mathbf{D_d}$ remain unchanged from their continuous representations. As these transformations require a high computational cost. The matrices' properties can be changed to achieve a simpler transformation. Assuming that the continuous-time matrix $\widetilde{\mathbf{A}}$ is diagonal, the matrix exponential simplifies to a diagonal matrix where the exponential of each eigenvalue is applied element-wise as seen in Equation 1.4

$$\widetilde{\mathbf{A}} = \begin{bmatrix} \widetilde{\lambda}_1 & 0 & \dots & 0 \\ 0 & \widetilde{\lambda}_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \widetilde{\lambda}_n \end{bmatrix}, \quad \Rightarrow \quad e^{\widetilde{\mathbf{A}}T_s} = \begin{bmatrix} e^{\widetilde{\lambda}_1 T_s} & 0 & \dots & 0 \\ 0 & e^{\widetilde{\lambda}_2 T_s} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & e^{\widetilde{\lambda}_n T_s} \end{bmatrix}. \tag{1.4}$$

The discrete-time matrix $\widetilde{\mathbf{B}}_\mathrm{d}$ is shown in Equation 1.5 where the integral of the matrix exponential is computed element-wise.

$$\widetilde{\mathbf{B}}_\mathrm{d} = \left(\int_0^{T_s} e^{\widetilde{\mathbf{A}}\tau} d\tau\right) \widetilde{\mathbf{B}}, \quad with \quad \int_0^{T_s} e^{\widetilde{\mathbf{A}}\tau} d\tau = \begin{bmatrix} \int_0^{T_s} e^{\widetilde{\lambda}_1 \tau} d\tau & 0 & \dots & 0 \\ 0 & \int_0^{T_s} e^{\widetilde{\lambda}_2 \tau} d\tau & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \int_0^{T_s} e^{\widetilde{\lambda}_n \tau} d\tau \end{bmatrix}. \tag{1.5}$$

Since the integration is element-wise for a diagonal matrix, $\int_0^{\mathrm{T_s}} e^{\widetilde{\lambda}_i \tau} d\tau = \frac{e^{\widetilde{\lambda}_i \mathrm{T_s}} - 1}{\widetilde{\lambda}_i}$ is computed and insert into Equation 1.5 which leads to Equation 1.6, where also the final representation for $\widetilde{\mathbf{B}}_\mathrm{d}$ can be seen.

$$\widetilde{\mathbf{B}}_\mathrm{d} = \begin{bmatrix} \frac{e^{\widetilde{\lambda}_1 T} - 1}{\widetilde{\lambda}_1} & 0 & \dots & 0 \\ 0 & \frac{e^{\widetilde{\lambda}_2 T} - 1}{\widetilde{\lambda}_2} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \frac{e^{\widetilde{\lambda}_n T} - 1}{\widetilde{\lambda}_n} \end{bmatrix} \mathbf{B} \tag{1.6}$$

## 1.4 Research Question and Contribution

SSMs offer a promising approach for neural audio synthesis due to their ability to capture long-range dependencies within sequential data. Due to their linear recurrence in the hidden state, they are processed efficiently and are parallelizable in computation. Moreover, SSMs are well suited for capturing temporal dynamics in audio due to time-dependent hidden states that evolve over time. This architecture allows them to learn complex audio patterns while maintaining efficiency in both training and inference compared to state-of-the-art neural audio synthesis networks.

This work presents a novel approach for synthesizing raw audio of real piano performances using SSMs, providing an efficient alternative to traditional models.

### 1.4.1 Research Question

**How can we design a lightweight and efficient, SSM-based Sequence to Sequence model to synthesize raw audio of piano performances in real-time?**

To further clarify the research question, following points are beeing researched to fulfill the research question:

- **SSM Architecture**: A SSM architecture must be found to learn the complex spectral characteristics of real piano audio while maintaining computational efficiency.
- **Model Complexity**: The complexity of the model, including the number of states and SSM layers, must be determined to provide real-time processing capabilities without compromising audio quality.
- **Hyperparameter Search**: Given the inherent sensitivity of SSMs, an appropriate hyperparameter tuning strategy must be found to ensure stable model behavior and prevent divergence due to infinite output loss.
- **Audio-Aware Loss Function**: Traditional loss functions such as Mean Squared Error (MSE) and Mean Absolute Error (MAE) fail to capture the spectral properties of audio. A specialized audio-aware loss function must be designed to learn spectral details while ensuring stable training dynamics.

### 1.4.2 Contribution

To address this research question, this thesis introduces a simple end-to-end trainable MIDI to raw piano audio synthesis NN architecture based on diagonal deep SSMs called Piano-SSM. The proposed Piano-SSM model offers the following contributions:

- **End-to-end MIDI-to-Raw Audio Synthesis with SSMs**: The proposed Piano-SSM only consists of six layers. The first layer, an input layer, upsamples the midi input to the sampling rate. The second to fifth layers are four S-Edge SSM layers that directly process MIDI inputs, and the last layer, the linear layer, generates raw piano audio. This lightweight architecture ensures real-time capabilities while synthesizing high-quality audio.
- **Evaluation on Subsampling from 44.1 to 16 kHz**: Due to the fact that SSMs can be inferred at different sampling rates, the Piano-SSM model's ability to be trained on audio data with a high sampling rate while synthesizing audio at lower sampling rates is shown. The model is evaluated on the MIDI and Audio Edited for Synchronous TRacks and Organization (MAESTRO) benchmark dataset.

- **Audio Focused Loss for SSM Training**: Mean-STFT-Mel (MSTM) loss for training the Piano-SSM models is proposed. MSTM loss is combined with a Short-Time Fourier Transform (STFT) Loss, a Mel-STFT Loss, and a pointwise raw audio Mean Loss. STFT loss covers the full frequency spectrum up to the Nyquist limit, with an Fast Fourier Transformation (FFT) size equal to the sampling rate. Mel-STFT uses a smaller window size, enabling higher temporal resolution accompanying the STFT loss with high-frequency resolution but lower time resolution, and the Mean Loss ensures no drift toward a negative or positive mean. This loss provides stable training while capturing spectral details important for human perception.
- **Efficient Autoregessive Causal Inference**: The discrete version of Piano-SSM can be seen as multiple Infinite Impulse Response (IIR) filters stacked with nonlinearities, allowing for fast causal autoregressive audio synthesis. Based on a custom C++17 head-only implementation, the real-time capabilities of an Intel Core i7-12700 Processor are shown.

As a complement to this thesis, audio samples[1] and the source-code[2] are provided online.

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

# Chapter 2

# Technical Background

This chapter shows the technical details of MIDI and the waveform representations used in the field of instrumentalists and audio processing. Specifically we discuss:

- **Musical Instrument Digital Interface (MIDI)** and **Piano Roll Conversion**: Introduces the MIDI protocol, explaining the technical specifications, and its role in modern musical data processing. Additionally it explains the transformation of MIDI data into piano roll format.
- **Waveform Representation**: Discusses digital audio formats, comparing the trade-offs between uncompressed and compressed formats (in terms of sampling rate, bit depth and format) and their impact on audio processing and quality.

## 2.1 Musical Instrument Digital Interface (MIDI)

MIDI, first introduced by Smith and Wood [16] in 1981, characterizes the communication between electronic instruments, audio devices, and computers. MIDI is an event-based protocol that consists of 7-bit messages containing two parameters: **Note On** event parameterized by the note number (from 0 to 127) and its velocity (from 0 to 127), and a **Note Off** event. The **Note Off** is parameterized by the same note number and velocity as the associated **Note On** and is sent when the event is finished. The note number is quantized by the note frequencies based on the equal temperament system, which spaces every semi-tone with a constant frequency ratio. Using note A3 with 440Hz as a base reference for note number 69, a frequency $f$ can be converted to its MIDI note $n$ by,

$$n = 69 + 12 \times \log_2 \left( \frac{f}{440} \right). \tag{2.1}$$

Vice versa, the equal-temperament frequency $f$ can be calculated from the MIDI note $n$ by using

$$f = 440 \times 2^{\frac{(n-69)}{12}}. \tag{2.2}$$

While the original MIDI standard from 1981 only encoded basic notes, more modern MIDI versions can contain various sets of information. However, it does not store the audio data like MP3 or Waveform Audio File Format (WAV) but the encoded instructions for an instrument that can synthesize audio. The audio can be synthesized by a human playing an instrument according to the MIDI instructions or by a MIDI-to-audio synthesizer. The key components used in this thesis of MIDI are:

- **Notes and pitches:** Notes are represented by their pitch using a number from 0 to 127. MIDI note numbers range from 0 (C-1) to 127 (G9). For a standard piano (88 keys), the range is from MIDI note 21 (A0) to 108 (C8).
- **Velocity:** Indicates the dynamic of the note. Ranges from 0 to 127.
- **Duration:** Specifies how long the note is held.
- **Channels and Instruments**: Specifies which instrument is played. For this thesis, only the piano is used
- **Tempo:** Indicates the beats per minute

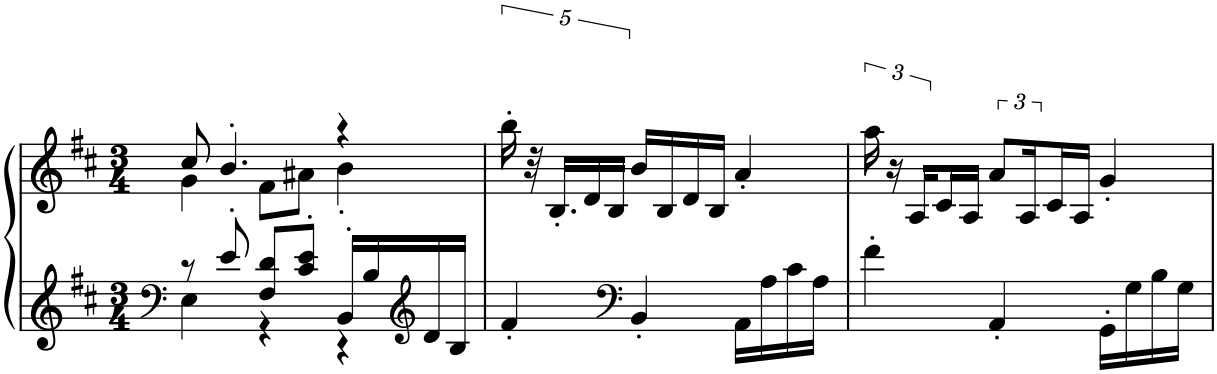
The tabular representation of MIDI in Table 2.1 provides a structured overview of MIDI notes regarding their pitch, timing, and dynamics. Each row represents a note event.

| Pitch (MIDI Note) | Start (s) | End (s) | Velocity (0–127) |
|---|---|---|---|
| 73 | 0.00 | 0.24 | 75 |
| 67 | 0.00 | 0.47 | 47 |
| 52 | 0.00 | 0.47 | 51 |
| 71 | 0.25 | 0.62 | 72 |
| 64 | 0.25 | 0.37 | 59 |
| 66 | 0.50 | 0.74 | 65 |
| 54 | 0.50 | 0.74 | 52 |
| 62 | 0.50 | 0.74 | 50 |
| 70 | 0.75 | 0.99 | 65 |
| 61 | 0.75 | 0.87 | 69 |
| ... | ... | ... | ... |

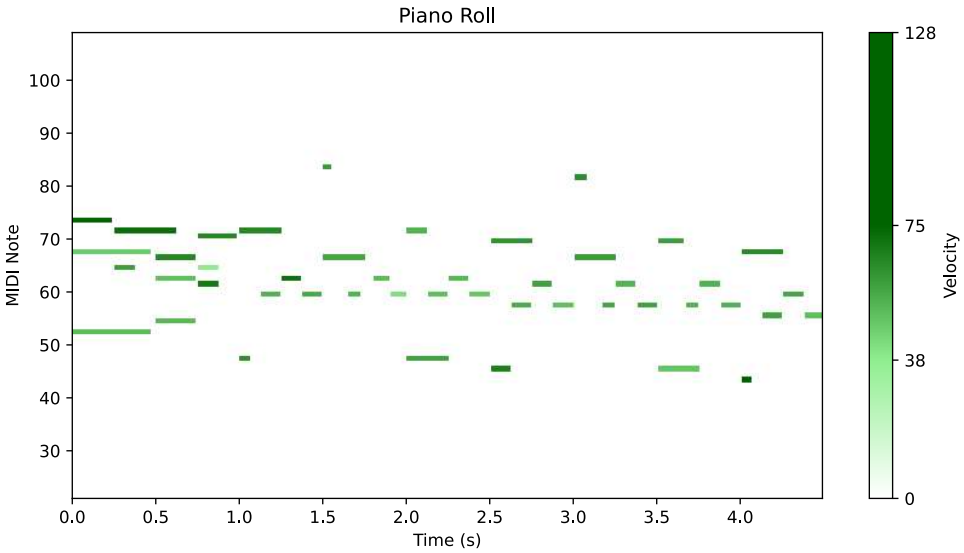
Table 2.1: Tabular representation of MIDI notes with pitch, start time, end time, and velocity of Domenico Scarlatti's *Sonata in B Minor, K. 27*

To make this information processable for a NN, a suitable representation must be found. MIDI can be converted into a Piano Roll format, which is a 2D matrix where the rows represent 88 possible (piano)

notes, and the columns correspond to the sample length of the audio snippet to be processed, with each value indicating the note's velocity, further described as MIDI-Input in this thesis. This process involves transforming a sheet music representation (see Figure 2.1a) into a piano roll format (see Figure 2.1b).




(a) Sheet Music Representation




(b) MIDI Piano Roll Representation

Figure 2.1: Comparison of Sheet Music Representation and MIDI Piano Roll Representation: (a) shows the sheet music for Domenico Scarlatti's *Sonata in B Minor, K. 27*, while (b) visualizes the corresponding MIDI piano roll. The velocities are represented in shades of green. The horizontal axis indicates the time in seconds, and the vertical axis corresponds to MIDI note numbers (21–108, spanning the full piano range).

## 2.2 Waveform representation

Digital audio formats can be categorized into compressed and uncompressed formats, each having advantages and trade-offs. Most digital sound signals are compressed to reduce file size and optimize bandwidth usage. Standard compressed formats include MP3, Advanced Audio Coding (AAC), and

OGG, which utilize perceptual coding to remove inaudible frequencies and redundant data, making them suitable for streaming and storage-efficient applications but unsuitable for audio synthesis tasks. Uncompressed formats, such as WAV and Audio Interchange File Format (AIFF), retain all original audio data and are typically characterized by their sampling rate and bit depth. Bit depth determines the resolution of an audio signal and typically ranges from 8 to 64 bits. Higher bit depths offer better dynamic range, improving sound quality. The sampling rate, which defines the number of times per second an analog signal is sampled to be converted into digital form, varies from 8 kHz to 192 kHz. Typical sampling rates include:

- **8 kHz**: Used in early telephone systems.
- **16 kHz**: Common for VoIP and voice recordings.
- **24 kHz**: Used in some low-bitrate audio codecs and streaming applications.
- **44.1 kHz**: Standard sampling rate for CDs and most digital audio.
- **48 kHz**: Standard for professional audio recording.
- **96 kHz to 192 kHz**: Used in high-resolution audio applications.

The Nyquist-Shannon theorem states that a signal can only be reconstructed from its samples if the sampling rate is at least twice as high as the highest frequency present in the signal, $f_s > 2f_{max}$, where $f_s$ is the sample frequency and $f_{max}$ is the highest frequency component of the signal [17]. Since the human auditory perception ranges from approximately 20 Hz to 20 kHz, a sampling rate of 44.1 kHz is sufficient for most audio tasks, ensuring accurate sound reproduction without unnecessary data redundancy. Figure 2.2, shows an example for a WAV audio representation.

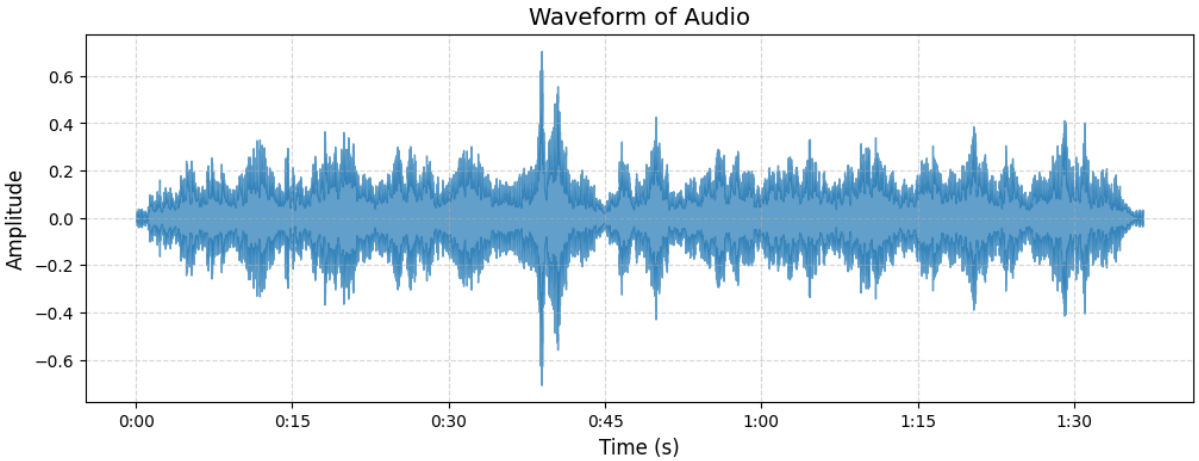



Figure 2.2: Waveform representation of Domenico Scarlatti's *Sonata in B Minor, K. 27*

# Chapter 3

# State Of The Art

The field of neural audio synthesis has seen remarkable developments in recent years, driven by advancements in machine learning, signal modeling, and efficient sequence modeling techniques. This chapter overviews the current state of the art in SSMs, raw audio generation with SSMs, and raw piano audio synthesis. Specifically, the next sections focus on the following areas:

- **Structured State Space Models (SSMs)**: SSM offer a promising architecture for modeling long-range dependencies in sequences and have recently been adapted to audio generation tasks. Several iterations, including S4, S4D, S5, and S-Edge, are discussed.
- **Neural Network Audio Synthesis**: This section presents NN-based approaches for synthesizing raw audio waveforms, focusing on Transformer based systems and differentiable signal models like DDSP.
- **Piano Music Datasets**: High-quality datasets are essential for training NN models. This section shows two benchmark datasets: MIDI and Audio Edited for Synchronous TRacks and Organization (MAESTRO) and MIDI Aligned Piano Sounds (MAPS), both containing paired MIDI and audio recordings.
- **Audio Focused Loss Functions**: This section explores state-of-the-art loss functions such as Multi-Scale Spectral Loss (MSSL) and Chroma Loss.

## 3.1 Structured State Space Models (SSMs)

Structured State Space Models (SSMs) are sequence models that use parameterized mappings to process sequential input data. These models have been increasingly explored for NN training, offering efficient memory mechanisms and computational advantages. The first significant application of State Space Models in NNs has been introduced by Gu et al. [18] with the HiPPO framework. This approach ini-

tializes State Space Models using specially constructed state matrices, enabling scalable and adaptive memory updates through polynomial projections. HiPPO efficiently maintains information across long sequences. Building on this foundation, Gu et al. [11] developed the S4 layer. S4 is modeled of multiple Sinlge Input Single Output (SISO) SSM layers and can be trained in the frequency domain. Further optimization are proposed with S4D, introduced by Gu et al. [12]. S4D is based on the S4 structure, enhancing its computing by a diagonal matrix instead of the previous diagonal plus low-rank matrices, leading to simplified and more efficient computations. Another advancement in Multiple Input Multiple Output (MIMO) SSMs was introduced by Smith et al. [13] with the S5 layer. Unlike the SISO SSMs used in S4, S5 uses a diagonal MIMO structure. Their models allow for parallelizing the training with the help of the associative scan operation, making S5 more computationally scalable. The latest generation of SSMs, S-Edge, is introduced by Bittner and Schnöll et al. [3]. S-Edge enables dynamic input and output shapes of MIMO SSM in its diagonal form and adds input and output bias units. For further refining the efficiency and scalability of SSMs they provide a custom C++17 head-only implementation.

SSMs have also been successfully applied in the audio waveform domain. Goel et al. [19] introduced SaShiMi, a framework that utilizes the S4 [11] model to generate raw audio waveforms. One of the key challenges in using S4 for autoregressive audio generation is its inherent instability. SaShiMi addresses this issue by introducing a modified parameterization, which enhances stability during generation. This adaptation enables SaShiMi to generate state-of-the-art performance in unconditional waveform generation tasks and output high-quality audio samples.

## 3.2 Neural Network Audio Synthesis

Neural Network Audio Synthesis uses neural network-based models to generate raw audio directly from abstract representations, such as MIDI data. By using neural network architectures, such as CNNs, Recurrent Neural Networks (RNNs), Transformers, or SSMs, they can synthesize high-quality audio signals.

Hawthorne et al. [20] introduced the Wave2Midi2Wave framework to model and generate piano music trained on the MAESTRO dataset. The framework addresses three challenges in music representation: transcription, composition, and synthesis. For the synthesis, the framework uses conditional WaveNet [10] to synthesize audio from MIDI representation. The system marks a significant step forward in generating expressive and realistic piano performances. Dong et al. [21] introduced a novel system that adapts the text to speech techniques to generate music performances from unaligned polyphonic scores utilizing a transformer-based encoder-decoder model (as visualized in Figure 1.3) and evaluated the results on MSE score and a subjective metric, which is a listening test consisting of a

Mean Opinion Score (MOS) where the participants rate the overall quality of an audio sample on a scale from 1 (very annoying) to 5 (actual recording). Cooper et al. [6] compared different TTS configurations utilizing Tacotron [22] as an acoustic model and neural source-filter [23] as waveform models also as encoder-decoder style architecture. For evaluation, they used Pitch mismatch scores and a mean opinion score. Shi et al. [5] improved this work and evaluated several configurations, including Tacotron and TransformerTTS [7] as waveform models and neural source-filter and HiFi GAN [8] as acoustic model. They also introduced a joint training of acoustic and waveform models to minimize the feature mismatch when trained separately. They evaluated their work on different objective evaluation metrics, including a pitch score, a chroma score, and a spectrogram score. For subjective evaluation, they also used a mean opinion score. Renault et al. [24,25], introduces Differentiable Digital Signal Processing (DDSP)-Piano v1 a mixture of NN model and a Signal model that consists of an extension of the DDSP [26] framework. The proposed approach designs a differentiable piano synthesizer with high-level modeling knowledge. The proposed model is evaluated using a MSSL score and a mean opinion score. DDSP-Piano v2 is also introduced by Renault et al. [4], where they optimize the signal model and introduce a new differentiable Feedback Delay Network (FDN) reverb layer [27] whose architecture is motivated by room reverberation modeling.

## 3.3 Piano Music Datasets

The dataset is one of the most important parts of NN training. If the dataset is inconsistent or contains many errors, the NN model cannot learn the data correctly. In the special case of piano music synthesis, the dataset must consist of real piano recordings aligned with the paired MIDI data. Additionally, it is crucial to ensure that the MIDI information is correctly annotated. Therefore, the dataset must fulfill the following key requirements:

- High MIDI to audio alignment
- High-quality audio recordings of a real piano

Two publicly available datasets that meet these criteria to a large extent are MAESTRO and MIDI Aligned Piano Sounds (MAPS). Both datasets contain paired MIDI and audio recordings and are widely used in the research community. They will be introduced in the following subsections.

### 3.3.1 MAESTRO

MIDI and Audio Edited for Synchronous TRacks and Organization (MAESTRO) dataset, developed by Google AI (Hawthorne et al.) [20], is a dataset that consists of MIDI data with high-quality audio recordings derived from performances captured during the International Piano-e-Competition. During each

installment of the competition, virtuoso pianists perform on Yamaha Disklaviers, which, in addition to being concert-quality acoustic grand pianos, utilize an integrated high-precision MIDI capture and playback system. The key features of the dataset are:

- **Paired MIDI and audio data:** The dataset consists of 198.7 hours of paired MIDI and audio recordings.
- **Train, Validation, and Test split:** The dataset is split into 159.2 hours of training data, 19.4 hours of validation data, and 20.4 hours of test data.
- **Sub-millisecond alignment:** The MIDI and audio tracks are synchronized with sub-millisecond accuracy.
- **Audio quality:** All recordings are provided in 44.1 kHz audio with 16-bit PCM stereo, except for 2017 and 2018 competitions, which are sampled at 48 kHz with 16-bit PCM stereo.
- **Metadata:** The dataset is also annotated with metadata, including the composer, year, and title of the piece
- **Content:** The dataset contains ten different years with different room acoustics and recording environments.

### 3.3.2 MAPS

MIDI Aligned Piano Sounds (MAPS) [28] is a piano database for multi-pitch estimation and automatic piano music transcription. It consists of MIDI-annotated piano recordings from real and software-based pianos. Table 3.1 shows all nine configurations in terms of instrument and recording condition. For this thesis only the *Instrument model* "Yamaha Disklavier Mark III (upright)" is used. The key features of the dataset are:

- **Paired MIDI and audio data:** The dataset consists out of ∼65 hours paired MIDI and audio recordings.
- **Train, Validation and Test Split:** There is no proposed Train, Validation, and Test split.
- **Sub-millisecond alignment:** The MIDI and audio tracks are synchronized with sub-millisecond accuracy.
- **Audio quality:** All recordings are provided in 44.1 kHz stereo audio with 16-bit PCM stereo.
- **Content:** The dataset is split into four sets:
  - 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

| Instrument model | Recording conditions | Real instrument or software |
|---|---|---|
| Hybrid | Software default | The Grand 2 (Steinberg) |
| Boesendorfer 290 Imperial | church | Akoustik Piano (Native Instruments) |
| Bechstein D 280 | concert hall | Akoustik Piano (Native Instruments) |
| Concert Grand D | studio | Akoustik Piano (Native Instruments) |
| Steingraeber 130 (upright) | jazz club | Akoustik Piano (Native Instruments) |
| Steinway D | "Ambient" | The Black Grand (Sampletekk) |
| Steinway D | "Close" | The Black Grand (Sampletekk) |
| Yamaha Disklavier Mark III (upright) | "Ambient" | Real piano (Disklavier) |
| Yamaha Disklavier Mark III (upright) | "Close" | Real piano (Disklavier) |

Table 3.1: MAPS instrument and recording conditions. The rows in **red** indicate software-based instruments, while the rows in **green** represent real pianos.

## 3.4 Audio Focused Loss Functions

In conventional machine learning tasks, loss functions are often based on direct signal differences, such as MSE or MAE. However, these standard point-wise losses do not adequately reflect perceptual differences for neural audio synthesis tasks. Human perception of audio is highly sensitive to spectral and harmonic features rather than raw waveform accuracy. Therefore, audio-focused loss functions aim to guide model training towards perceptually relevant representations. These losses are designed to capture the timbral, tonal, and structural properties of the audio signal in the time-frequency domain or perceptual space. Such audio focused losses improve the fidelity and naturalness of synthesized audio, even when exact waveform alignment is not achieved. The following subsections describe two commonly used audio-focused losses MSSL and Chroma Loss.

### 3.4.1 Multi Scale Spectral Loss (MSSL)

MSSL, introduced by Engel et al. [26], describes a loss function that compares the spectral properties of two audio signals using the L1 and log L1 difference between $|X_i|$ and $|Y_i|$ as seen in Equation 3.1.

$$\text{SpectralLoss} = \sum_i \Big\| |X_i| - |Y_i| \Big\|_1 + \Big\| \log|X_i| - \log|Y_i| \Big\|_1 \tag{3.1}$$

$|X_i|$ and $|Y_i|$ are calculated by using the STFT with an overlap of 75% and $hann$ as windowing function. $\|\cdot\|_1$ states the L1 norm. For the FFT-size, Engel et al. [26], and Renault et al. [25] state that they are using $i \in \{2048, 1024, 512, 256, 128, 64\}$ with a sampling rate of 16 kHz for their experiments, in [4] they also used this FFT-sizes but with a sampling rate of 24 kHz. Jonason et al. [29] are using

$i \in \{12288, 6144, 3072, 1526, 384, 192\}$ at a sampling rate of 48 kHz. Figure 3.1 visually represents different FFT-sizes at a 16 kHz sampling rate. Each plot shows the spectrogram calculated by a STFT of an input signal with different FFT-sizes. Bigger FFT-sizes display a broader frequency spectrum but less latent information. Vice-versa, a small FFT-sizes display a small frequency spectrum but retain a good latent representation.

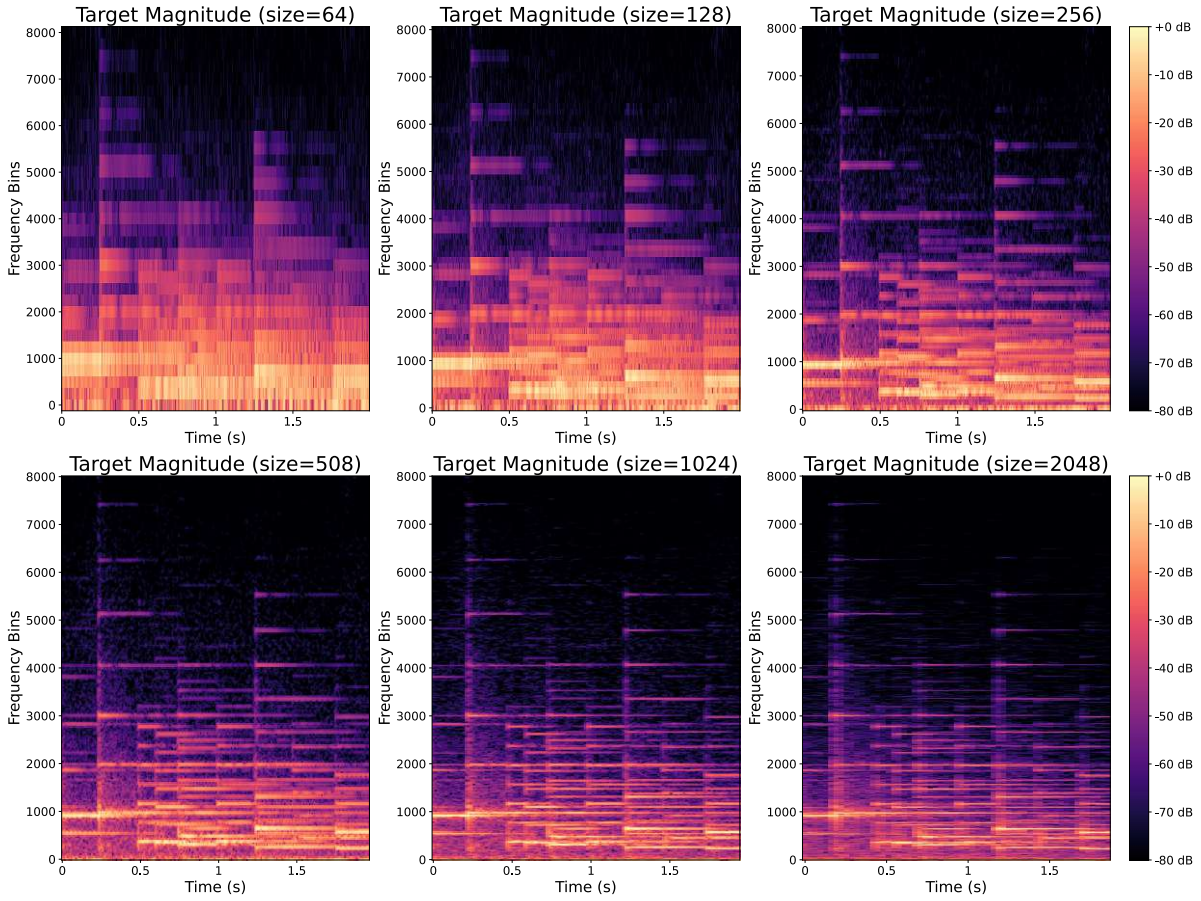



Figure 3.1: Comparative analysis of STFT spectrograms of Domenico Scarlatti's *Sonata in B Minor, K. 27* for different window sizes. Each plot has a sampling rate of 16 kHz and different FFT-sizes $i \in \{2048, 1024, 512, 256, 128, 64\}$.

### 3.4.2 Chroma Loss

Chroma Loss, used by Shi et al. [5], is a perceptual loss function designed to evaluate the harmonic similarity between two audio signals using their chroma features. Chroma features are extracted using the *librosa.feature.chroma_cqt* function from the *librosa* [30] library. These features represent the tonal content of an audio signal by mapping its frequency components into 12 pitch classes. The ChromaLoss function, as defined in Equation 3.2, measures the distance between the chroma features of the synthesized and the natural audio. ChromaLoss is calculated as the sum of absolute differences between the

target and the generated audio, normalized by the sum of a masking function (mask), which filters out low-intensity chroma values.

$$\text{ChromaLoss} = \frac{\sum |\text{target} - \text{generated}|}{\sum \text{mask}} \tag{3.2}$$

The target values are obtained by clipping the reference chroma values to the range [0,1]: target $=$ $\min(1, \max(0, \text{ref\_chroma}))$. Similar, the generated chroma values are constrained to the range [$10^{-5}$, $1 - 10^{-5}$]: generated $= \min(1 - 10^{-5}, \max(10^{-5}, \text{syn\_chroma}))$. The masking function (mask) filters out the ref_chroma low-intensity chroma values which are smaller than $0.3$, mask $= \{x \in \text{ref\_chroma} \mid x > 0.3\}$. Chroma features ref_chroma and syn_chroma are obtained using the function chroma_cqt. The reference chroma is calculated from the natural audio ref_chroma $=$ chroma_cqt(natural_audio) and the synthesized chroma is extracted from the generated audio:
syn_chroma $=$ chroma_cqt(synthesized_audio). Figure 3.2 visually represents the chroma features of natural, synthesized audio and compares their difference.

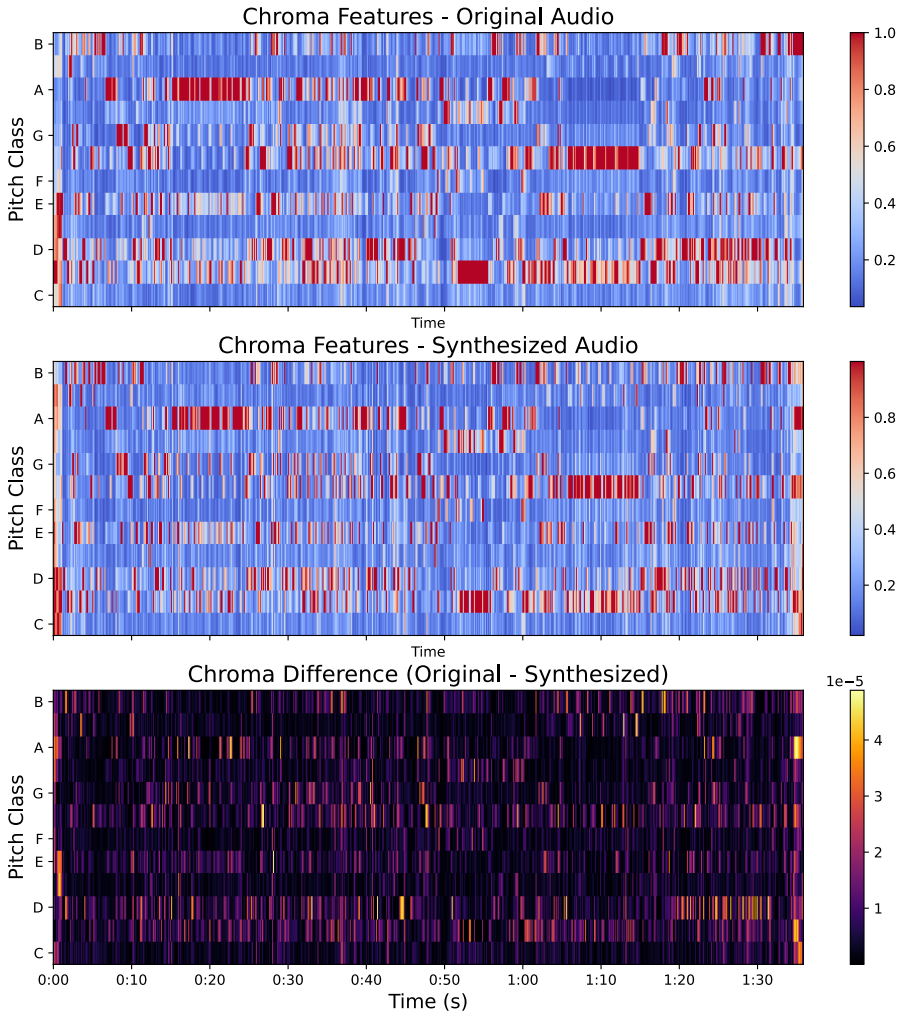



Figure 3.2: Comparison of Chroma Features of Domenico Scarlatti's *Sonata in B Minor, K. 27* between original and synthesized audio: The first two plots display the chroma feature representations of the original and synthesized audio. The third plot highlights the chromatic differences, with brighter regions indicating greater differences

# Chapter 4

# Methodology

This chapter describes the methods and design decisions used to develop and train the proposed Piano-SSM model for real-time raw audio synthesis. The aim is to construct a system that can generate high-quality piano audio directly from MIDI input while remaining interpretable and computationally efficient. The individual sections provide a detailed explanation of the model architecture, the training strategy, and the design of loss functions as follows:

- **Model Architecture:** Introduces Piano-SSM, a Structured State Space Model (SSM) based on the S-Edge layer. It describes the mathematical formulation, discretization, and architectural details. A learning rate scheduler is shown, which adapts the learning rate by a dynamic schedule based on cosine annealing with warmup and restarts to improve training stability. Additionally, a differentiable reverberation module based on feedback delay networks used to model room acoustics across different datasets and sampling rates is shown.
- **Audio Focused Loss:** Presents audio loss functions that better match human auditory perception, including spectral losses such as MSSL, STFT, Mel-STFT, as well as the proposed MSTM composite loss.

## 4.1 Model Architecture

### 4.1.1 Piano-SSM - A Real-Time Piano Raw Audio Synthesis Model

This work introduces a SSM based NN model architecture Piano-SSM that builds upon the hardware-friendly S-Edge Layer proposed by Bittner and Schnöll et al. [3]. The key features of the S-Edge layer include MIMO, efficient GPU-based computation via parallelization, due to the diagonal state transition matrix $\widetilde{\mathbf{A}}$. The S-Edge layer is modeled as a complex-valued continuous-time MIMO system,

$$\begin{aligned}\dot{\widetilde{\mathbf{x}}}(t) &= \widetilde{\mathbf{A}}\widetilde{\mathbf{x}}(t) + \widetilde{\mathbf{B}}\mathbf{u}(t) + \widetilde{\mathbf{b}} \\ \mathbf{y}(t) &= \Re(\widetilde{\mathbf{C}}\widetilde{\mathbf{x}}(t) + \widetilde{\mathbf{c}}),\end{aligned} \tag{4.1}$$

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. The schematics of the S-Edge layer can be seen in Figure 4.1. Compared to S-Edge, Piano-SSM uses a slightly modified representation of the eigenvalues. Instead of modeling the eigenvalues in cartesian coordinates $\widetilde{\boldsymbol{\lambda}} = \boldsymbol{\alpha} + i\boldsymbol{\beta}$, they are represented and optimized in the polar coordinate form with norm $\mathbf{r}$ and angle $\boldsymbol{\phi}$. Specifically, it is modeled with $\widetilde{\boldsymbol{\lambda}} = e^{\mathbf{r}+i\boldsymbol{\phi}}$, which optimizes the norm in log-space. In order to ensure exponential stability the real part of the eigenvalues is constraint to be negative with $\Re(\widetilde{\boldsymbol{\lambda}}) = \text{-abs}(\Re(\widetilde{\boldsymbol{\lambda}}'))$. $\mathbf{r}$ and $\boldsymbol{\phi}$ are implemented as two PyTorch parameter tensors of shape state_size × 1, capturing both modulus and phase components. S-Edge or S5 applies a trainable multiplicative time scale parameter $\Delta$ to each eigenvalue. Since this has the effect of scaling the norm in cartesian coordinates, it is neglected, as the norm is already optimized. During training and inference, the continuous-time representation has to be mapped to the discrete version. The Piano-SSM layer does this with ZOH. Due to the diagonal implementation of the state transition matrix, discretization can be performed efficiently on an elementwise basis,

$$\begin{aligned}\begin{bmatrix}\widetilde{\mathbf{B}}_{\mathrm{d}} & \widetilde{\mathbf{b}}_{\mathrm{d}}\end{bmatrix} &= \mathrm{diag}\left(\widetilde{\boldsymbol{\lambda}}^{-1} \circ (\widetilde{\boldsymbol{\lambda}}_{\mathrm{d}} - 1)\right)\begin{bmatrix}\widetilde{\mathbf{B}} & \widetilde{\mathbf{b}}\end{bmatrix}, \\ \widetilde{\boldsymbol{\lambda}}_{\mathrm{d}} = e^{\widetilde{\boldsymbol{\lambda}}\mathrm{T_s}}, &\quad \widetilde{\mathbf{C}}_{\mathrm{d}} = \widetilde{\mathrm{C}}, \quad \widetilde{\mathbf{c}}_{\mathrm{d}} = \widetilde{\mathbf{c}}.\end{aligned} \tag{4.2}$$

By adjusting $\mathrm{T_s}$, the discrete SSM layers can operate at different effective sampling rates, which means the network can be trained at the original sampling rate $\mathrm{f_{s\_train}}$ and then switch the system dynamics to generate audio with a lower sampling rate $\mathrm{f_{s\_inf.}}$, by setting $\mathrm{T_s} = \mathrm{f_{s\_train}}/\mathrm{f_{s\_inf.}}$

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

$$\begin{aligned}\widetilde{\mathbf{x}}_k &= \widetilde{\boldsymbol{\lambda}}_d \circ \widetilde{\mathbf{x}}_{k-1} + \widetilde{\mathbf{B}}_{\mathrm{d}}\mathrm{u}_k + \widetilde{\mathbf{b}}_{\mathrm{d}}, \\ \mathbf{y}_k &= \Re(\widetilde{\mathbf{C}}_{\mathrm{d}}\widetilde{\mathbf{x}}_k + \widetilde{\mathbf{c}}_{\mathrm{d}}) \\ \text{Output} &= \text{Skip}(\mathbf{u}_k) + \text{Activation}(\mathbf{y}_k),\end{aligned} \tag{4.3}$$

where Skip $\in \mathbb{R}^{\mathrm{O}\times\mathrm{Y}}$ is a trainable real-valued matrix. The number of parameters for the discrete

representation follows with,

$$\text{Total Parameters} = \underbrace{2\text{YH}}_{\widetilde{B}} + \underbrace{2\text{OH}}_{\widetilde{C}} + \underbrace{\text{YO}}_{\text{Skip}} + \underbrace{4\text{H} + \text{O}}_{\widetilde{\lambda}_d,\ \text{biases}}. \tag{4.4}$$

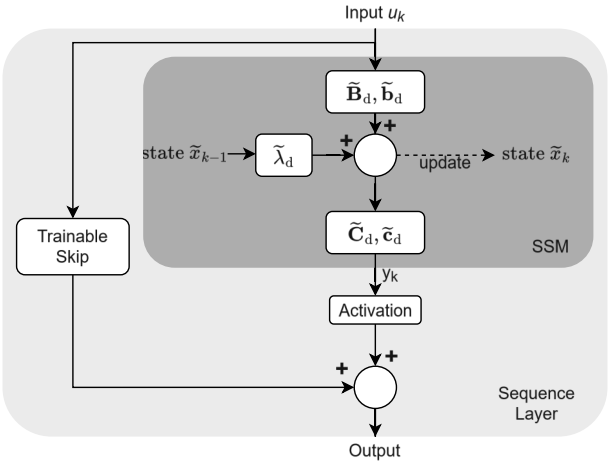



Figure 4.1: Schematics of the S-Edge Layer introduced by Bittner and Schnöll et al. [3]

Figure 4.2 outlines the general structure of the Piano-SSM architecture. As an activation function, the hyperbolic tangent is used. The whole architecture consists of six layers:

- Layer 1 – Copy Features Layer to up sample the MIDI input from MIDI rate to sampling rate by ZOH and is not train-able.
- Layer 2 – One SSM layer without a skip connection to uncouple the input better from the output. Not reducing the channels, staying at 88 channels.
- Layer 3-5 – Three stacked SSM-Piano layers, which reduce the dimensionality from 88 to 20 channels.
- Layer 6 – A final fully connected layer Linear Layer that reduces the 20-channel representation to a single output channel.

Different versions of Piano-SSM vary in state size H, impacting the total number of parameters. Table 4.1 summarizes these variations.

| Model | State Size | Parameters |
|---|---|---|
| XL | 256 | 268.4k |
| L | 128 | 142.4k |
| S | 64 | 79.4k |

Table 4.1: Model Parameters for different configurations of Piano-SSM

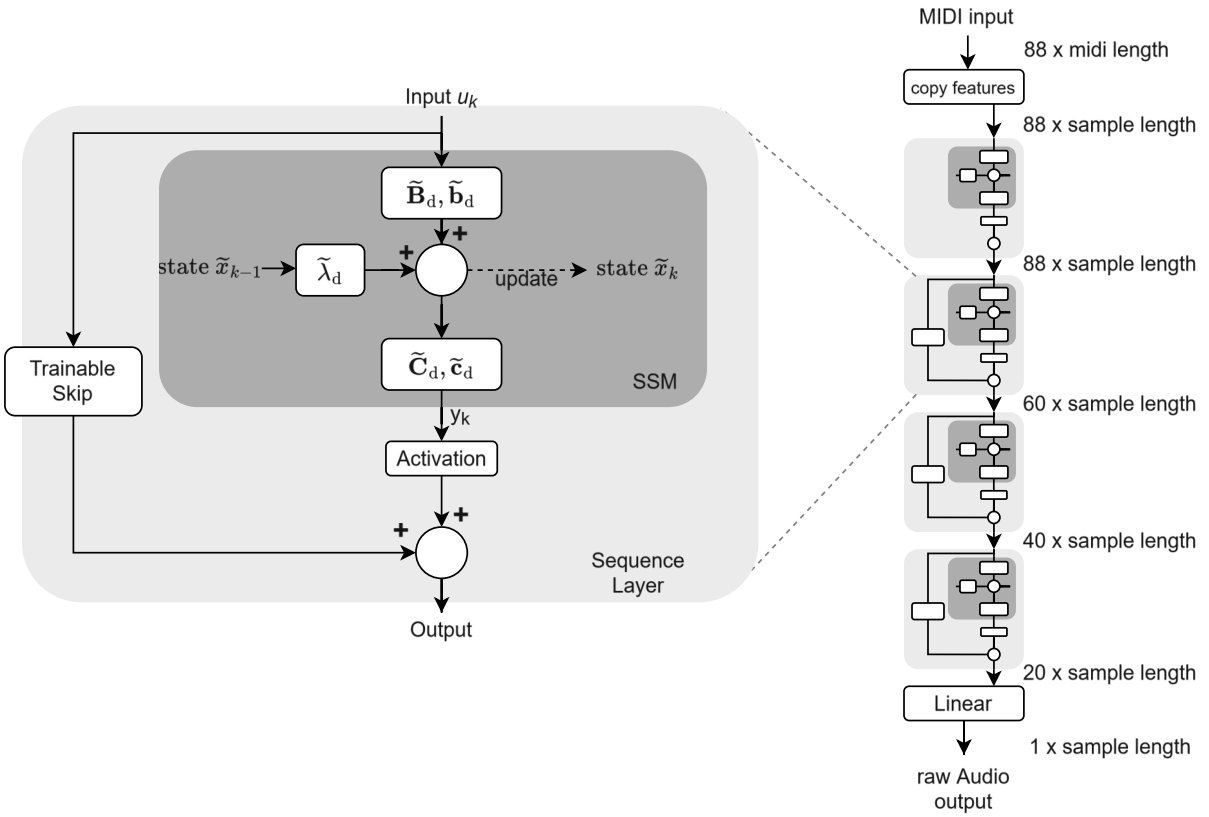



Figure 4.2: Architecture of Piano-SSM. The model processes 88 MIDI input channels, which are progressively reduced through SSM layers to 20 latent dimensions. A final linear layer maps this representation to a single-channel raw audio output.

### 4.1.2 Scheduler - Cosine Annealing with Warmup

The learning rate is an important factor in NN training, significantly influencing the model's convergence and performance. Dynamic learning rate schedulers have been proposed to improve training stability. Cosine annealing [31] adjusts the learning rate according to a cosine function dependent on the current epoch. Periodic restarts allow the model to escape local minima (or maxima). The learning rate for the current epoch $t$ is defined as $\eta_t = \eta_{\min} + \frac{1}{2}(\eta_{\max} - \eta_{\min})\left(1 + \cos\left(\frac{T_{\text{cur}}}{T_i}\pi\right)\right)$ where $\eta_{\min}$ and $\eta_{\max}$ are the minimum and maximum learning rates, $T_{\text{cur}}$ is the number of epochs since the last restart and $T_i$ is the number of epochs between two warm restarts. Since SSMs can become unstable when starting with a high learning rate, it is critical to begin with a sufficiently small learning rate to prevent divergence and numerical instability. To solve this issue, the implementation of Katsura et al. *CosineAnnealingWarmupRestarts* scheduler [32] is used, which implements a cosine annealing algorithm with warmup phase and restarts. The warmup phase enhances the training stability by gradually increasing the learning rate from a lower initial value to the target maximum value over a predefined number of epochs.

The key parameters of *CosineAnnealingWarmupRestarts* include:

- *first_cycle_steps*: First cycle step size.
- *cycle_mult*: Cycle steps magnification.
- *max_lr*: First cycle's maximum learning rate.
- *min_lr*: Minimum learning rate.
- *warmup_steps*: Number of linear warmup step size.
- *gamma*: Factor for decreasing the maximum learning rate by cycle.

Figure 4.3 illustrates the learning rate schedule with the following configuration:

- lr = $0.0001$
- epochs = $200$
- max_lr = $5 \times$ lr
- min_lr = $\frac{\text{lr}}{10}$
- first_cycle_steps = $0.4 \times$ epochs
- cycle_mult = $1.0$
- warmup_steps = $0.25 \times$ epochs
- gamma = $\left(\frac{\text{min\_lr}}{\text{max\_lr}}\right)^{\frac{1}{\text{epochs/first\_cycle\_steps}}}$

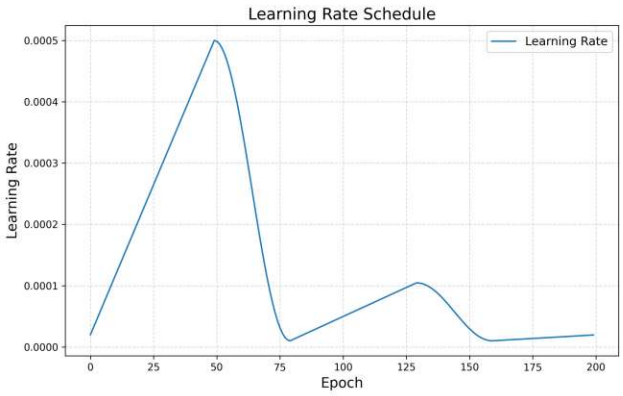



Figure 4.3: Learning rate schedule with cosine annealing, warmup and two restarts

### 4.1.3 Reverberation

The reverberation layer used in this work is based on the differentiable reverberation model proposed by Lee et al. [27]. This model captures the complex dynamics of room acoustics and is particularly useful for simulating the diverse recording environments found in the MAESTRO dataset. Renault et al. [4] implemented this reverberation model in TensorFlow [33] using eight delay lines with delay lengths: $d \in \{233, 311, 421, 461, 587, 613, 789, 891\}$ at a sampling rate of 24 kHz. This extends the original implementation from Lee et al. [27], which was designed with six delay lines at a sampling rate of 48 kHz, using the delay lengths: $d = \{233, 311, 421, 461, 587, 613\}$. For this work, the reverberation layer from Renault et al. is re-implemented in PyTorch [34], maintaining six delay lines. To adapt the model to different sampling rates, a scaling factor: $s = \frac{f_s}{48000}$ is applied. This adjusts the delay lengths to: $d = s \cdot \{233, 311, 421, 461, 587, 613\}$. Furthermore, the initialization parameters for the all-pass delay filters are also scaled according to the sampling rate: delay_allpass_std $= s \cdot 400$ and delay_allpass_mean $= s \cdot 60$. This implementation ensures that the reverberation characteristics remain consistent across different sampling rates while maintaining the perceptual qualities of the original model.

## 4.2 Audio Focused Loss

Classic loss functions like MSE or MAE do not catch any spectral details and are very sensitive to small time shifts between label and prediction. This disadvantage makes them unsuitable for audio synthesis tasks. For this reason, a loss function is used, which is more similar to the human hearing perception. MSSL is a widespread loss function to capture spectral details. Unfortunately, this work has shown that the Piano-SSM model has difficulties being trained with this loss function due to convergence and instability issues. For this reason, a new loss function MSTM is introduced to gain training stability while capturing audio spectral details.

### 4.2.1 Multi-Scale Spectral Loss (MSSL)

Section 3.4.1, highlights the mathematical foundation of MSSL. Renault et al. [24, 25] uses it for 16 kHz with adapted parameters, Engel et al. [26] uses the configuration for 48 Hz. Renault also uses it in [4] with 24 kHz but with the same parameters as 16 kHz. When observing Figure 4.4, it becomes clear that resolution and details vary significantly if the FFT-sizes are not adapted to changes in the sampling rate. For example, at a sampling rate of 48 kHz and FFT-size 256, the spectral image is inaccurate compared to 16 kHz with FFT-size 256.

Due to this limitation, a dynamic scaling of MSSL is applied. Different FFT-sizes can be seen in Table 4.2, where the frequency resolution scales with the used sampling rate: Frequency resolution $= \frac{f_{s\_eval}}{\text{FFT-Size}}$, with $f_{s\_eval}$ as the evaluation sampling rate. Using different FFT-sizes assures that the frequency resolution stays the same for every sampling rate. Therefore, it is ensured that the FFT models the same dynamics for every sampling rate. This dynamic scaling, which depends on the sampling rate, ensures that the exact spectral details are gathered at every sampling rate making it better comparable across different sampling rates.

### 4.2.2 Short-Time Fourier Transform (STFT) Loss

The STFT is widely used in audio signal processing because it provides a time-frequency representation of signals, enabling high-resolution spectral analysis. The key feature of the STFT loss function is that it can capture time, phase, and magnitude information. The STFT is computed using a sliding window approach, where an FFT is applied to overlapping signal segments. The windowed segments are defined by a chosen window function, commonly a Hann window, which smooths the edges and reduces spectral leakage. The degree of overlap between successive STFT windows is controlled by the hop length, determining the temporal resolution of the spectrogram. The Hann window function is defined as: $w[n] = \frac{1}{2}(1 - \cos(\frac{2\pi n}{N})), 0 \leq n \leq N$. for a sequence of $N + 1$ samples.

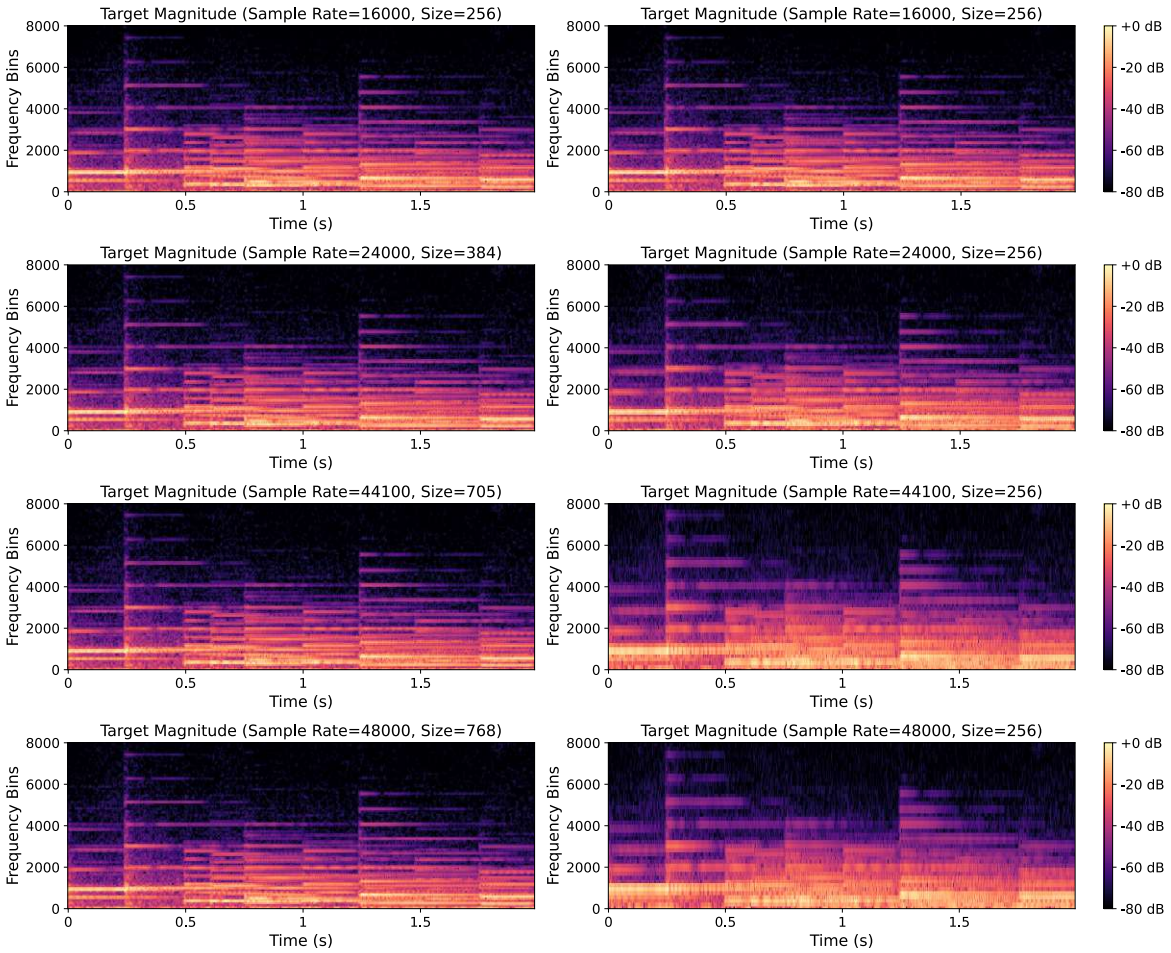



Figure 4.4: Comparative analysis of spectrograms for different window sizes of Domenico Scarlatti's *Sonata in B Minor, K. 27*. Each row corresponds to a specific sampling rate from 16 to 48 kHz. The left side has a scaled FFT-sizes, and the right side has always 256 as FFT-size. The color scale (dB) remains consistent across all plots for accurate comparison.

| $\frac{f_{s\_eval}}{\textbf{FFT-Size}}$ = **Frequency Resolution (Hz per bin)** | | | |
|---|---|---|---|
| $\frac{16000}{4096} = 3.91$ | $\frac{24000}{6144} = 3.91$ | $\frac{44100}{11289} = 3.91$ | $\frac{48000}{12288} = 3.91$ |
| $\frac{16000}{2048} = 7.81$ | $\frac{24000}{3072} = 7.81$ | $\frac{44100}{5644} = 7.81$ | $\frac{48000}{6144} = 7.81$ |
| $\frac{16000}{1024} = 15.63$ | $\frac{24000}{1536} = 15.63$ | $\frac{44100}{2822} = 15.63$ | $\frac{48000}{3072} = 15.63$ |
| $\frac{16000}{512} = 31.25$ | $\frac{24000}{763} = 31.25$ | $\frac{44100}{1402} = 31.25$ | $\frac{48000}{1526} = 31.25$ |
| $\frac{16000}{256} = 62.50$ | $\frac{24000}{384} = 62.50$ | $\frac{44100}{705} = 62.50$ | $\frac{48000}{768} = 62.50$ |
| $\frac{16000}{128} = 125.00$ | $\frac{24000}{192} = 125.00$ | $\frac{44100}{352} = 125.00$ | $\frac{48000}{384} = 125.00$ |
| $\frac{16000}{64} = 250.00$ | $\frac{24000}{96} = 250.00$ | $\frac{44100}{176} = 250.00$ | $\frac{48000}{192} = 250.00$ |

Table 4.2: Comparison of frequency resolution (Hz per bin) for different sampling rates $f_{s\_eval} \in \{16000, 24000, 44100, 48000\}$ and FFT-sizes. The table illustrates how different sampling rates yield equivalent spectral resolutions when using proportionally scaled FFT-sizes.

The STFT is computed using the following parameters:

- **FFT-size**: The size of the Fourier transform, defined as $n\_fft = f_s$
- **Window length**: The size of the windowed frame and STFT filter, given by $win\_length = f_s$
- **Hop length**: The step size between neighboring sliding windows, computed as $hop\_length = \frac{f_s}{10}$
- **Window function**: Hann window

This results in an frequency resolution of 1 Hz per bin and a time resolution of 100 ms per frame, determined by the shifting window.

The STFT loss is computed as

$$\text{STFT Loss} = \Big\| |X| - |Y| \Big\|_1, \tag{4.5}$$

where $|X|$ and $|Y|$ magnitude spectrograms of the predicted and ground truth audio signals, obtained using the STFT. $\|\cdot\|_1$ states the L1 norm. A nonreduced difference graph and a comparison of prediction and ground truth can be seen in Figure 4.5.

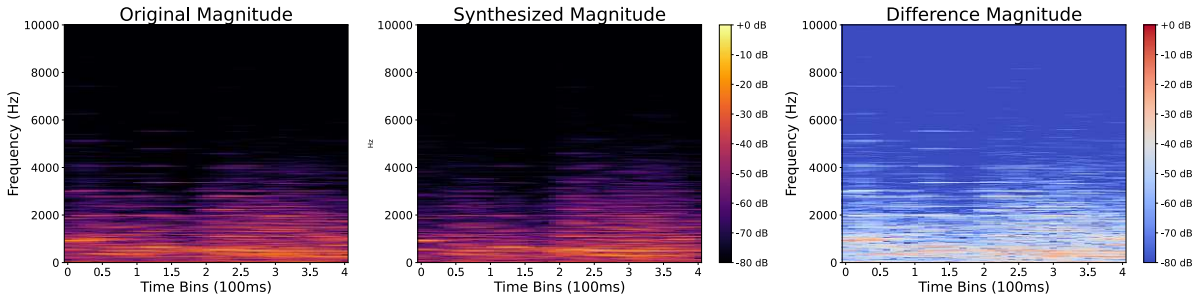



Figure 4.5: Spectrogram comparison of ground truth (label), predicted output, and their difference of Domenico Scarlatti's *Sonata in B Minor, K. 27.* The difference highlights regions where the model's prediction deviates from the actual signal.

The STFT parameters can be adjusted to enhance the visibility of MIDI note frequencies in the spectrogram. The MIDI notes are converted to corresponding frequencies using Equation 2.2. The adjusted STFT parameters are:

- **FFT-size**: $n\_fft = \frac{f_s}{10}$ (corresponding to 100 ms)
- **Hop length**: $\frac{f_s}{1000}$ (corresponding to 1 ms)
- **Window function**: Hann window

The visualization of the MIDI notes (in green) can be seen in Figure 4.6, which overlays the MIDI representation on the STFT spectrogram.

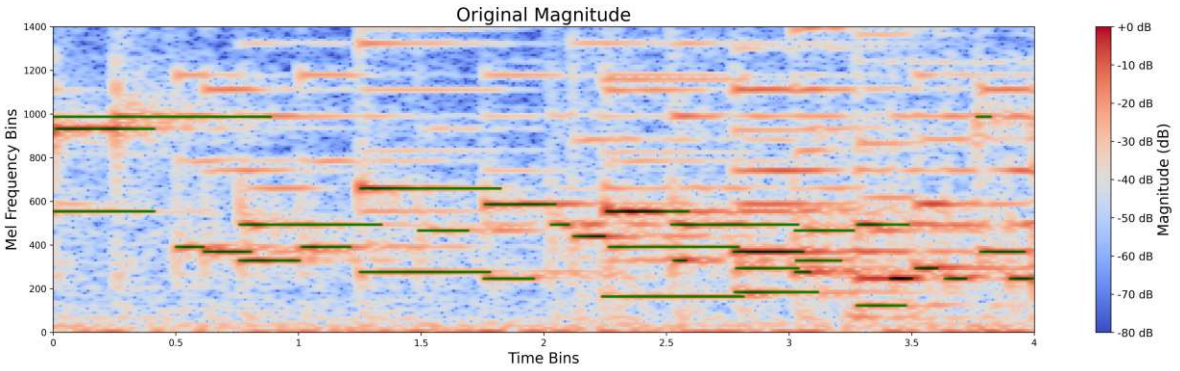



Figure 4.6: STFT spectrogram of the audio from Domenico Scarlatti's *Sonata in B Minor, K. 27*, with an overlay of the corresponding MIDI notes (in green). The visualization illustrates the alignment between the spectral content and musical pitch.

### 4.2.3 Mel-STFT Loss

The Mel-STFT loss, introduced by Steinmetz et al. [35] is part of the *auraloss* library, which provides audio-focused loss functions for audio processing. This loss function builds upon the standard STFT but incorporates Mel-scaling to emphasize perceptually relevant frequency components. Unlike a STFT, which operates on a linear frequency scale, resulting in a spectrogram where frequency bins are spaced evenly, the Mel-STFT applies a transformation that maps the frequency axis to the Mel scale. The Mel scale is designed to align closer with human auditory perception. Providing higher resolution at lower frequencies and lower resolution at higher frequencies. This transformation is achieved by applying a Mel filter bank to the STFT spectrogram, summing the spectral energy into Mel-frequency bins. As a result, the Mel-STFT loss better captures perceptual audio difference which makes it a good candidate for the audio synthesis task. The Mel-STFT loss is defined as,

$$\text{Mel-STFT Loss} = \frac{\big\| |Y_{\text{mel}}| - |X_{\text{mel}}| \big\|_F}{\big\| |Y_{\text{mel}}| \big\|_F} + \big\| \log |X| - \log |Y| \big\|_1, \tag{4.6}$$

where $X$ and $Y$ are the STFT outputs of the predicted and ground truth audio signals. $|X_{\text{mel}}|$ and $|Y_{\text{mel}}|$ are the corresponding Mel-scaled STFT magnitudes with $F_{\text{mel}} \in \mathbb{R}^{M \times K}$ as the Mel filterbank matrix, $M$ as the number of Mel bins and $K$ as the number of STFT bins. $\| \cdot \|_F$ is the Frobenius norm with $\|A\|_F = \sqrt{\sum_{i,j} |A_{i,j}|^2}$.

The Mel-scaled STFT is computed with the following parameters, adjusted according to the sampling rate using the scaling factor: $s = \frac{f_s}{44100}$

- **FFT-size**: $fft\_size = 2048 \times s$
- **Window length**: $win\_length = 1024 \times s$
- **Number of Mel bands**: 128

- **Hop size**: $256 \times scaling$
- **Window function**: Hann window

### 4.2.4 Mean-Mean Squared Loss

STFT and MSSL do not effectively capture the actual mean value of the audio signal. Due to the nature of SSM models, a drift toward a negative or positive mean is likely to occur. To address this issue, a small mean loss term is added to ensure that the audio's mean remains at zero, as follows,

$$\text{Mean Loss} = \mathbb{E}\Big[\mathbb{E}[x]^2 - \mathbb{E}[y]^2\Big], \tag{4.7}$$

where $x$ and $y$ are the predicted and ground truth audio signals.

### 4.2.5 Mean-STFT-Mel (MSTM) Loss

For training our Piano-SSM model, a combined loss function is proposed. The MSTM Loss combines multiple loss terms to effectively capture different spectral properties of an audio signal. This approach balances frequency and time resolution while ensuring the stability of the NN training. The Loss consists of the following components:

- STFT Loss: Covers the full frequency spectrum up to the Nyquist limit, with an FFT-size equal to the sampling rate. Additionally a large window size improves frequency resolution but reduces time resolution (1s window, 100ms hop size).
- Mel-STFT Loss: Approximates human hearing better through a Mel-Scaling which compresses high and expands low frequencies. The smaller window (46ms window, 5.8ms hop size) improves temporal resolution, making the loss sensitive to transient structures and fine timing differences.
- Mean Loss: Ensures that there is no drift toward a negative or positive mean.

The combined MSTM Loss is computed with

$$\text{MSTM Loss} = \text{STFT Loss} + \text{Mel-STFT Loss} + \text{Mean Loss}. \tag{4.8}$$

# Chapter 5

# Experiments

This chapter presents the experimental evaluations of the proposed Piano-SSM model with a detailed focus on:

- **Experimental Setup:** Details the training and evaluation parameters for the Piano-SSM model.
- **MAESTRO Evaluation:** Comparison of Piano-SSM against DDSP-Piano v1/v2 and Piano-TTS models on the MAESTRO dataset and a evaluation of synthesis across different sampling rates.
- **MAPS Evaluation:** Generalization capability of Piano-SSM when trained on isolated notes and chords from the MAPS dataset and evaluated on full musical pieces.
- **Real-Time Performance:** Analysis of auto-regressive inference capabilities and real-time factors (RTFs) for various model configurations.
- **Interpretability:** Investigation of the model's learned dynamics based on the eigenvalue of the state matrices, as well as visualization of the audio evolution across layers.

## 5.1 Experimental Setup

This thesis performs experiments on the MAESTRO and MAPS dataset to evaluate the proposed Piano-SSM models: XL with a state size of 256, L with a state size of 128, and S with a state size of 64. The models are trained using PyTorch, utilizing 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. [32]. Training on the MAESTRO dataset needs 50 epochs, with a sample size of 50,000 samples per epoch. After the 50 epochs, the models are finetuned using MSSL Loss for an additional 20 epochs with a learning rate of $2e{-}5$. In contrast, training on the MAESTRO 2009 or MAPS dataset runs for 200 epochs, with a smaller sample size of 5,000 per epoch. A single linear output layer is used when training on the MAPS dataset. Instead,

when training our Piano-SSM on the MAESTRO dataset, a distinct linear layer is employed per year, ensuring model adaptability across different years. All Piano-SSM models are trained using MSTM 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 v1/v2 models, which are also evaluated using 10 second audio samples. The window sizes for MSSL are scaled as described in section 4.2.1.
For example, training the Piano-SSM XL model on audio with a sampling rate of 24 kHz requires 116 hours of training on a single NVIDIA A100 80GB GPU for the MAESTRO dataset. The training setup generally uses a batch size of 8 and a sample length of 4 seconds.

## 5.2 Experimental Evaluations MAESTRO

In this thesis, Piano-SSM is compared with DDSP-Piano v1, DDSP-Piano v2 and Piano-TTS on the MAESTRO v3.0.0 dataset to evaluate the performance of different model configurations and sampling rates. The goal is to evaluate how SSM-based synthesis performs at different sampling rates and to compare it against existing state-of-the-art models. Furthermore, Piano-SSM is evaluated at different sampling rates, demonstrating its ability to sub-sample for faster inference while only losing a small amount of MSSL. This is achieved through adjusting the sampling time $\mathrm{T_s} = \mathrm{f_{s\_train}}/\mathrm{f_{s\_inf.}}$, when mapping from the continuous SSM representation to discrete. It ensures a consistent system of the discrete inference model across different sampling rates. Piano-SSM is substantially smaller compared to audio synthesis Piano-TTS models, even in its largest configuration (Piano-SSM XL), it is slightly smaller than DDSP-Piano v1 and v2, while achieving similar synthesis quality. Table 5.1 presents the sizes of state-of-the-art audio synthesis models and three different Piano-SSM configurations, which differ only in SSM state size (see Table 4.1)

### 5.2.1 DDSP-Piano Comparison

To evaluate the performance of the proposed Piano-SSM model, it is compared against DDSP-Piano v1 and DDSP-Piano v2 on the MAESTRO v3.0.0 dataset. The models are evaluated using the Multi-Scale Spectral Loss (MSSL) (see Section 4.2.1) on different synthesis sampling rates. Figure 5.1 shows the MSSL scores for Piano-SSM XL and the DDSP-Piano v1 and v2 models. DDSP-Piano v1 was trained at 16 kHz, while DDSP-Piano v2 was trained at 24 kHz. In contrast, Piano-SSM XL 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-SSM XL maintains comparable synthesis quality to DDSP-Piano v1 and v2 while being more parameter-efficient. Additionally Piano-SSM is aconsiderably simpler network architecture, which requires no expert knowledge about pianos.

| Model | Parameters | Model | Parameters |
|---|---|---|---|
| Piano-TTS v2 [5] | 31.5M | Piano-TTS v1 [6] | 31.4M |
| - Transformer-TTS [7] | 17.6M | - Tacotron-2 [36] | 30.6M |
| - HiFi-GAN [8] | 13.9M | - NSF [23] | 736.3k |
| DDSP-Piano v2 [4] | 344.5k | DDSP-Piano v1 [24, 25] | 512.5k |
| - Sub-models | 341.5k | - Sub-models | 281.5k |
| - Tuning Models | 70 | - Tuning Models | 33 |
| - FDN Reverb [27] | 2820 | - Reverb | 240k |
| Piano-SSM XL | 268.4k | Piano-SSM S | 79.4k |
| Piano-SSM L | 142.4k | | |
| Piano-SSM FDN XL | 268.7k | Piano-SSM FDN S | 79.7k |
| - SSM Model | 268.4k | - SSM Model | 79.4k |
| - FDN Reverb [27] | 2820 | - FDN Reverb [27] | 2820 |
| Piano-SSM FDN L | 142.7k | | |
| - SSM Model | 142.4k | | |
| - FDN Reverb [27] | 2820 | | |

Table 5.1: 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. [4].

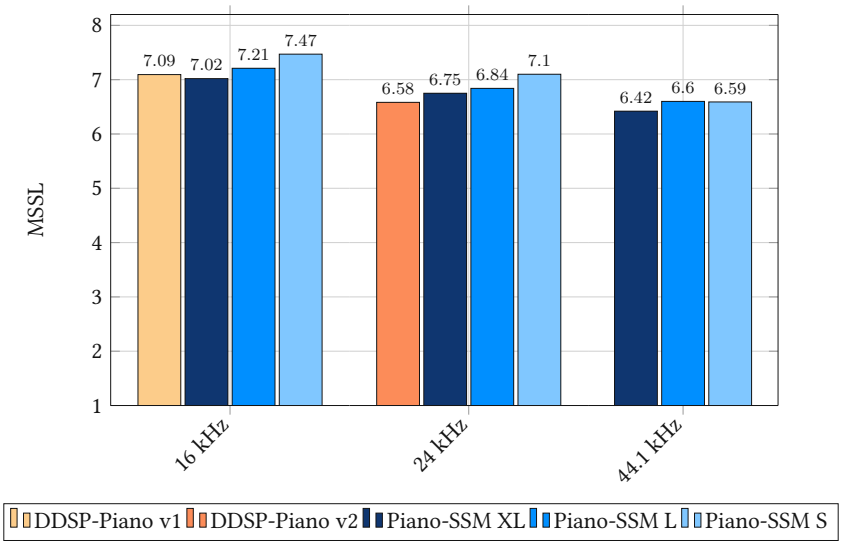



Figure 5.1: Comparison of MSSL values (lower is better) for Piano-SSM **XL**, **L**, **S** and DDSP-Piano models across different synthesis sampling rates. The Piano-SSM models were trained separately at each sampling rate, while **DDSP-Piano v1** was trained at 16 kHz, and **DDSP-Piano v2** at 24 kHz.

### 5.2.2 Piano-TTS Comparison

Unfortunately, it was not possible to resynthesize or gain access to raw audio files for Piano-TTS, making a MSSL-based evaluation not possible. However, DDSP-Piano outperformed Piano-TTS v2 as stated by Renault et al. [4]. The only available comparison metric is the chroma loss (see Section 3.4.2). While chroma loss is not as good as MSSL for evaluating piano audio quality, it still gives some information about pitch accuracy. All evaluations were conducted at a 24 kHz sampling rate for comparability.

Moreover, Piano-TTS has a significantly larger parameter count, approximately 117× more than the

Piano-SSM XL model, and requires more complex intermediate representations. In contrast, Piano-SSM relies only on MIDI-to-audio modeling via SSM models without any intermediate layers.

| Model | Acoustic Model | Waveform Model | Chroma Loss ↓ |
|---|---|---|---|
| joint-nsfg | Transformer-TTS [7] | NSF-GAN | 0.38 |
| joint-hfg | Transformer-TTS [7] | HiFi-GAN [8] | 0.38 |
| Piano-SSM XL | – | – | **0.28** |
| Piano-SSM L | – | – | **0.31** |
| Piano-SSM S | – | – | **0.30** |

Table 5.2: Comparison of chroma loss between Piano-SSM and Piano-TTS models. Lower Chroma Loss ↓ is better.

As shown in Figure 5.2, Piano-SSM exhibits a much simpler architecture than typical TTS-based models, without any complicated intermediate feature representations such as spectrograms. This simplicity enables efficient training and inference while still achieving state-of-the-art audio quality.

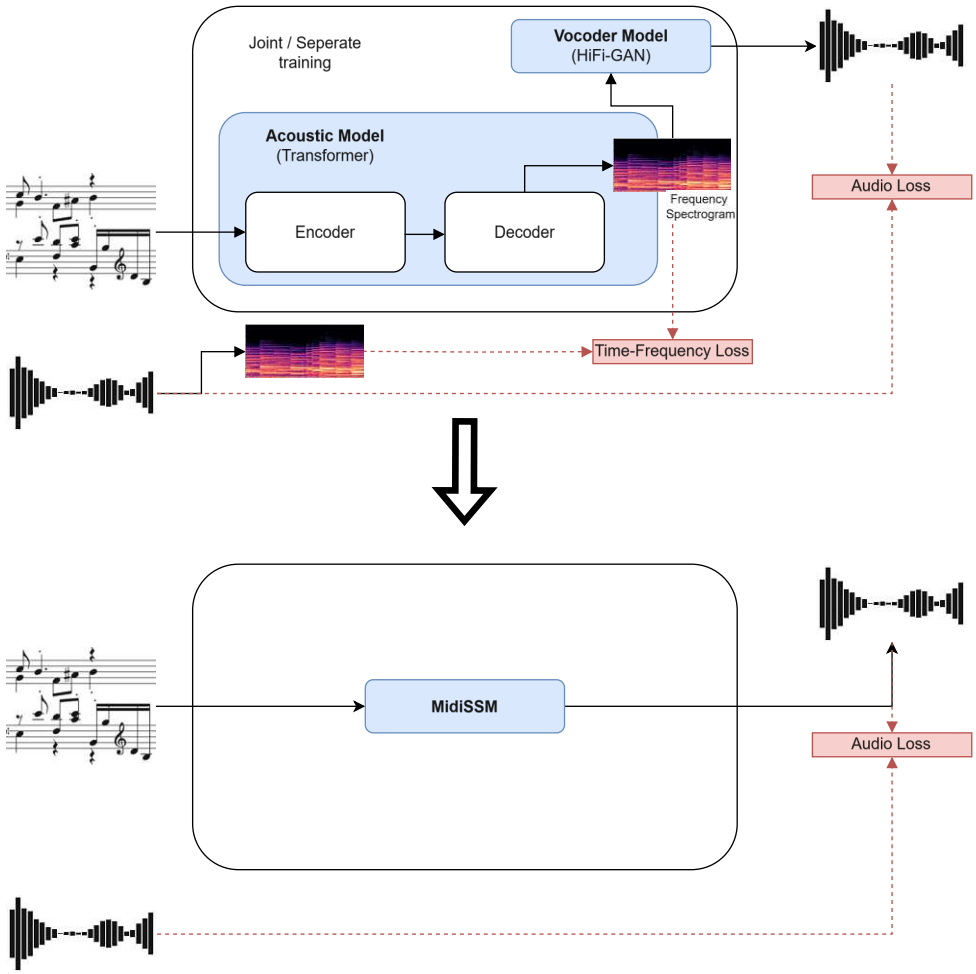



Figure 5.2: Comparison of a typical TTS-based architecture with the end-to-end Piano-SSM approach, which needs no intermediate representations

### 5.2.3 Variable Sampling Rates

To show the impact of different synthesis sampling rates on the performance of Piano-SSM, the models are evaluated and trained at different sampling rates and compared using the MSSL score when synthesizing audio at lower sampling rates. In detail, the effect of training at 44.1 kHz and downsampling to 24 kHz and 16 kHz, as well as training at 24 kHz and downsampling to 16 kHz, is examined. Additionally, the effects on MSSL when downsampling original audio is explored.

Figure 5.3 presents the MSSL values for the three Piano-SSM variants (XL, L, and S) trained on the MAESTRO 2009 dataset. Each model variant was trained separately at 16 kHz, 24 kHz, and 44.1 kHz, and the MSSL score is evaluated across these rates. The Figure shows a general trend of decreasing MSSL when downsampling from the training sampling rate to a smaller synthesis sampling rate. While the smallest model S shows the biggest loss in MSSL, the biggest model XL keeps a relative low MSSL score even when downsampling from 44.1 kHz to 16 kHz.

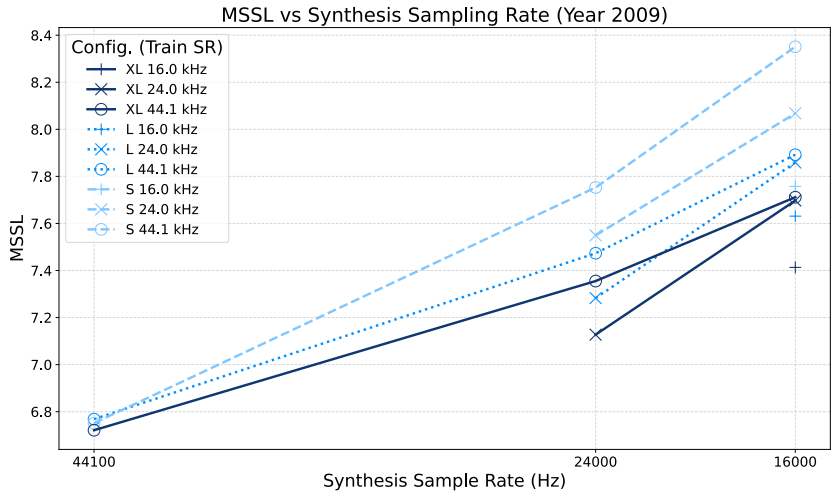



Figure 5.3: Comparison of **MSSL** values for different synthesis sampling rates (16 kHz, 24 kHz, and 44.1 kHz) of the year 2009. The plot shows results for the Piano-SSM model three variants: **XL** (**solid**), **L** (**dashed**), and S (dotted). Each model variant was trained separately at 16 kHz (+), 24 kHz (×), and 44.1 kHz (○). Lower MSSL values indicate better performance, with a general trend of increasing MSSL as the synthesis sampling rate decreases.

Figure 5.4 shows the MSSL values for the three Piano-SSM variants (XL, L, and S) trained on the full MAESTRO dataset, evaluated across different evaluation sampling rates. In this setup, the model is trained at a given *Train Sampling Rate* and used to synthesize audio at a specific *Synthesis Sampling Rate*. Both the ground truth and the synthesized audio are then resampled to a common *Evaluation Sampling Rate* to enable consistent comparison across different configurations.

When the *Evaluation Sampling Rate* is set to 16 kHz, the models generally retain higher MSSL scores. This is primarily due to the limited spectral bandwidth at lower sampling rates, which reduces the

amount of high-frequency detail that must be reconstructed. As a result, even models that were originally trained at higher sampling maintain their performance when downsampling. In contrast, the task becomes significantly more demanding when evaluating at the full 44.1 kHz resolution. The broader spectral content requires the model to accurately reconstruct high-frequency information that is often more challenging to learn and generalize. However, a general trend emerges across all evaluation sampling rates. MSSL tends to decrease when downsampling to smaller synthesis sampling rates.

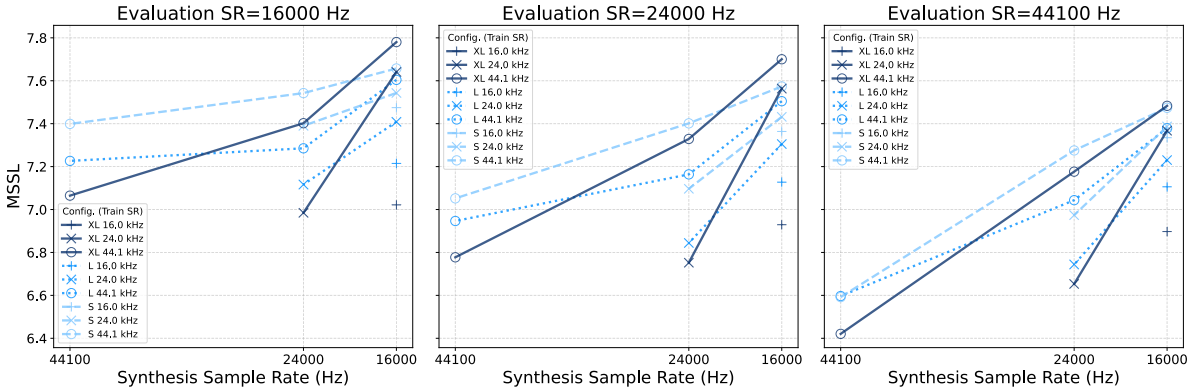



Figure 5.4: Comparison of MSSL values for different **Synthesis Sampling Rate**s (16 kHz, 24 kHz, and 44.1 kHz) on the MAESTRO dataset. The plot shows results for the three Piano-SSM variants: **XL** (**solid**), **L** (**dashed**), and **S** (**dotted**). Each model variant was trained separately at 16 kHz (+), 24 kHz (×), and 44.1 kHz (○). Lower MSSL values indicate better performance. Each plot shows the model's performance when evaluated at a specific **Evaluation** Sampling Rate **SR**.

A complementary evaluation is presented in Figure 5.5. In this setup, all synthesized audio is compared directly to the full-resolution ground truth at 44.1 kHz, which remains unaltered throughout the whole experiment. Each model is trained at different sampling rates and then synthesized at the corresponding synthesis sampling rates like in the previous experiment. Then, synthesized audios are sampled to 44.1 kHz and compared to the original 44.1 kHz ground truth without resampling the ground truth audio. This setup shows each model's synthesis quality relative to the original audio's full spectral content and the general advantages of training at higher sampling rates.

The plot also includes a baseline comparison between downsampled and re-upsampled versions of the ground truth audio (mean over the whole MAESTRO dataset). The *Original 16 kHz* and *Original 24 kHz* points represent ground truth audio first downsampled to the respective sampling rate, then upsampled back to 44.1 kHz before comparison. These reference points illustrate the inherent spectral loss introduced by low sampling rates, even without synthesis artifacts.

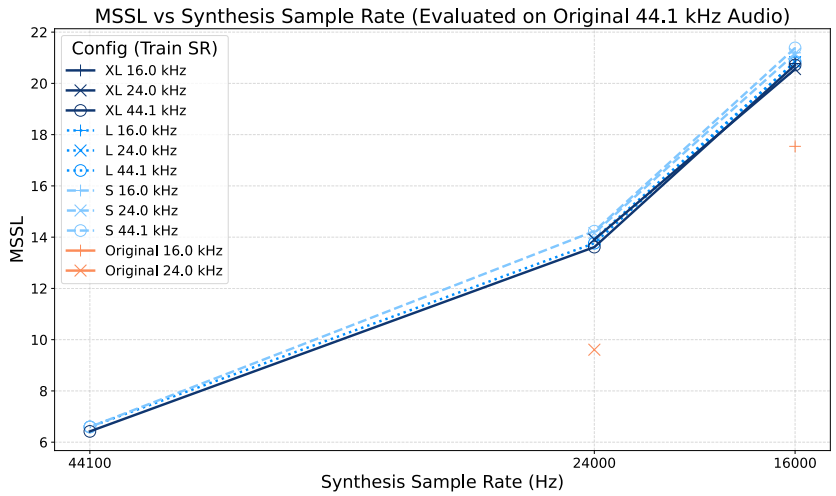
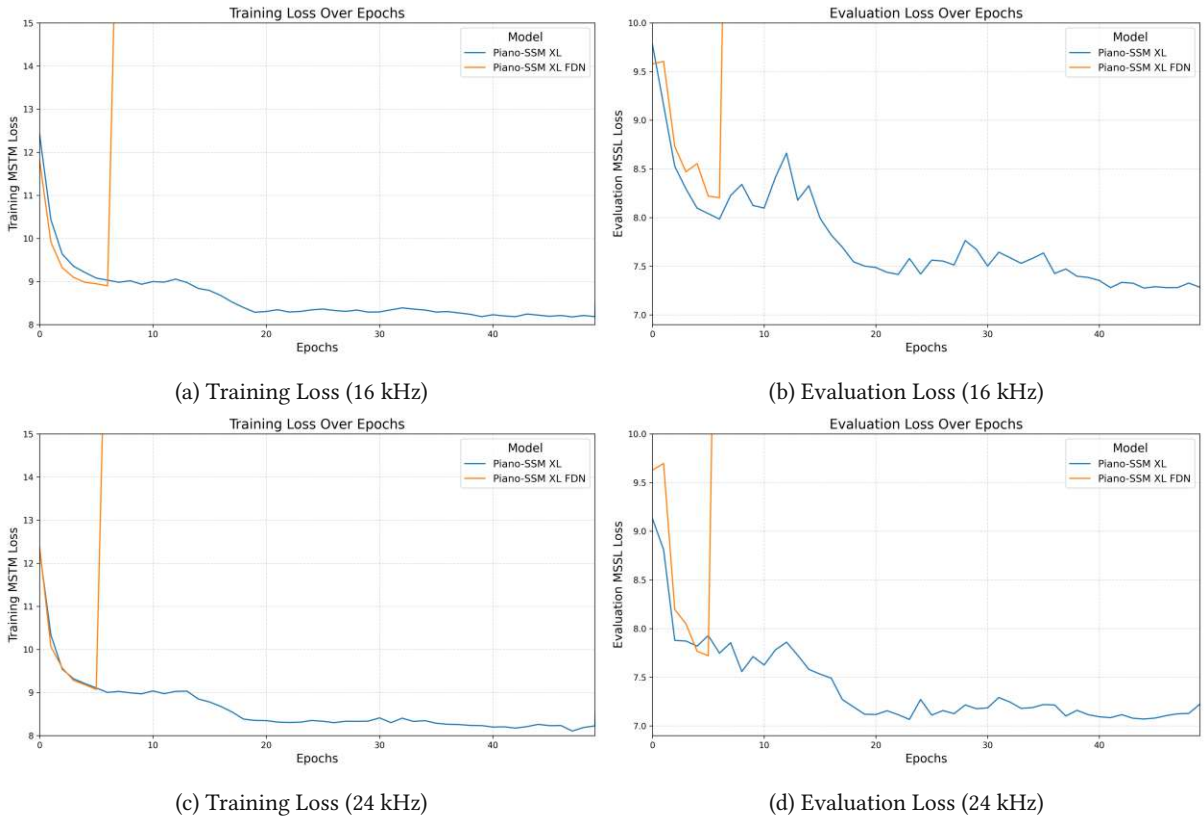



Figure 5.5: Comparison of MSSL values for different synthesis sampling rates (16 kHz, 24 kHz, and 44.1 kHz) on the MAESTRO dataset. The plot shows results for the three Piano-SSM variants: **XL** (**solid**), **L** (**dashed**), and **S** (**dotted**). Each model variant was trained separately at 16 kHz (+), 24 kHz (×), and 44.1 kHz (○). Lower MSSL values indicate better performance. The plot shows the evaluation of the synthesized audio against the original 44.1 kHz ground truth. Additional markers indicate the performance of downsampled and re-upsampled versions of the ground truth audio for reference.

### 5.2.4 Reverberation Evaluation

Reverberation might be more promising for room acoustics modeling than multiple linear layers. However, it currently shows significant instability during training. Further research is needed to identify the root causes of this instability and develop methods to prevent the instability. Figure 5.6 shows the training and evaluation loss curves over epochs. The configuration Piano-SSM XL refers to the previously described model in which each year is represented by a distinct linear layer. In contrast, Piano-SSM XL FDN integrates a FDN reverberation layer of size 10 for each year to simulate room acoustics more realistically.




(a) Training Loss (16 kHz)

(b) Evaluation Loss (16 kHz)

(c) Training Loss (24 kHz)

(d) Evaluation Loss (24 kHz)

Figure 5.6: Training and evaluation loss of Piano-SSM XL and Piano-SSM XLFDN for different sampling rates. (a) Training Loss at 16 kHz, (b) Evaluation Loss at 16 kHz, (c) Training Loss at 24 kHz, (d) Evaluation Loss at 24 kHz.

## 5.3 Experimental Evaluations MAPS

The Piano-SSM XL model is trained on isolated notes and chords using the ISOL, RAND, and UCHO sets of the MAPS dataset. The evaluation is done on the MUS set of the MAPS dataset, which consists of pieces of piano performances. This configuration makes it a more challenging task than the MAESTRO dataset. Table 5.3 shows the dataset durations in hours used for training and evaluating different datasets. *MAPS close* describes the MAPS Yamaha Disklavier Mark III part in condition *Close*, and *MAPS ambient* is the *ambient* condition of the MAPS dataset. MAESTRO 2009 and MAESTRO Full, consisting of complete piano performances, offer a larger training set compared to the MAPS dataset, which has a smaller duration and variation in the dataset.

Despite being trained only on single isolated notes and chords, the model was tested on full piano performances, demonstrating its ability to generalize beyond the training data. While the results do not match those models trained on the MAESTRO dataset, they remain remarkable, given the limited training set. Table 5.4 shows the MSSL score evaluations for the MAPS Yamaha Disklavier Mark III in Close, Ambient, and both (Ambient & Close) settings compared to models trained on MAESTRO 2009 and full MAESTRO dataset, but evaluated on MAPS. The linear layer in the Piano-SSM XL trained on

| 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 2009 | Train | **18 hours** |
| | Test | 2.7 hours |
| MAESTRO Full | Train | 159.2 hours |
| | Test | 20.4 hours |

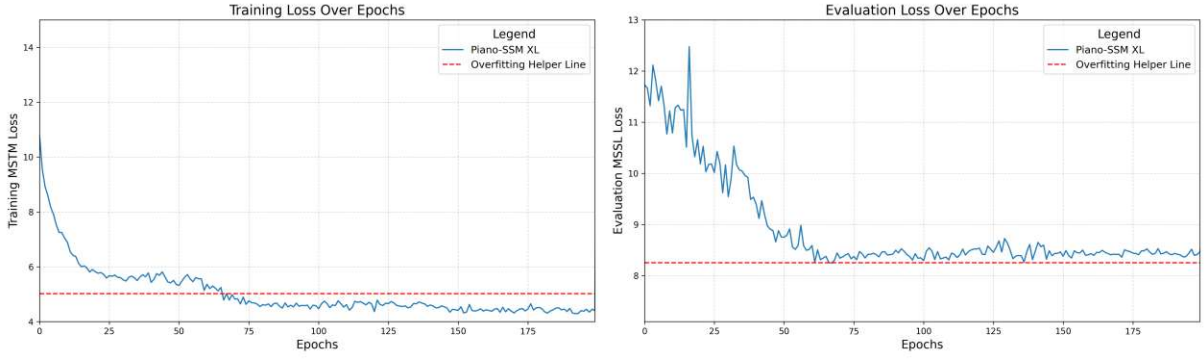
Table 5.3: Total duration of datasets used for training and evaluation. MAPS datasets train on isolated notes and chords and evaluate on pieces of piano performances, while MAESTRO datasets contain complete piano performances for both training and evaluation

the full MAESTRO dataset is set to the year 2011.

| Configuration | Training | Evaluation | MSSL ↓ |
|---|---|---|---|
| Piano-SSM XL | Close | Close | 8.71 |
| Piano-SSM XL | Ambient | Ambient | 8.30 |
| Piano-SSM XL | Ambient & Close | Ambient & Close | 8.23 |
| Piano-SSM XL | MAESTRO 2009 | Ambient & Close | 8.77 |
| Piano-SSM XL | MAESTRO Full | Ambient & Close | **7.95** |

Table 5.4: Results of Piano-SSM XL at 24 kHz, comparing models trained on MAPS with isolated notes and chords to a model trained on MAESTRO 2009 and full MAESTRO dataset, which includes full piano performances. Lower MSSL ↓ is better

Due to the minimal amount of training data (see Table 5.3), the Piano-SSM model tends to overfit, as shown in Figure 5.7. The red dotted line in the plot acts as an indicator of overfitting. While the training loss continues to decrease steadily, the evaluation loss plateaus, signaling that the model has begun to overfit.




(a) Training Loss (24 kHz)

(b) Evaluation Loss (24 kHz)

Figure 5.7: Training and Evaluation over Epochs for Piano-SSM-XL, trained on MAPS Ambient. Red is the *Overfitting Helper Line*

## 5.4 Auto-regressive Real-Time Capabilities

A key advantage of SSMs is their ability to efficiently infer in an auto-regressive mode. The discrete version of Piano-SSM can be seen as multiple IIR filters stacked with nonlinearities, allowing for fast causal autoregressive audio synthesis. For comparison, DDSP-Piano v2 [4] reports a real-time factor (RTF) of 1.9±0.1 on a 2.6 GHz Intel Xeon E5-2623 v4 CPU. The RTF performance of Piano-SSM on an Intel Core i7-12700 processor at ∼4.5 GHz using single-core across multiple sampling rates and model sizes is evaluated. 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 head-only implementation [2, 3] optimized for low-latency SSM based audio synthesis. In addition to RTF, the 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 is reported. The results are shown in Table 5.5.

| **Config.** | **Sampling Rate [kHz]** | **Model Size [MB]** / **L2 Cache [MB]** | **Est. RTF** | **RTF ↓** | **Workload [GFLOP/s]** |
|---|---|---|---|---|---|
| XL | 44.1 | 1.07 / 1.25 | 0.41 | 0.44 | 23.105 |
| L | 44.1 | 0.57 / 1.25 | 0.22 | 0.22 | 11.996 |
| S | 44.1 | 0.32 / 1.25 | 0.12 | 0.12 | 6.442 |
| XL | 24.0 | 1.07 / 1.25 | 0.22 | 0.24 | 12.574 |
| L | 24.0 | 0.57 / 1.25 | 0.12 | 0.12 | 6.529 |
| S | 24.0 | 0.32 / 1.25 | 0.07 | 0.06 | 3.506 |
| XL | 16.0 | 1.07 / 1.25 | 0.15 | 0.16 | 8.383 |
| L | 16.0 | 0.57 / 1.25 | 0.08 | 0.08 | 4.352 |
| S | 16.0 | 0.32 / 1.25 | 0.04 | 0.04 | 2.337 |

Table 5.5: Real-Time factor (**RTF**) of different Piano-SSM **Config**urations at different **Sampling Rates**, measured on an Intel Core i7-12700 processor. **Model Size** is the size of the model in megabytes, while **L2 Cache** indicates the available L2 cache per performance core. Lower (↓) **RTF** indicates faster inference time. **RTF** correlates with the **Est**imated **RTF** which is the calculated data bound inference speed based on the L2 cache. The **Workload** required to synthesize one second of raw audio is reported.

The processing time required for one second of audio using the Fused Multiply-Add (FMA) (AVX2) instruction at ∼4.5 GHz is estimated to be around 120 GFLOP/s for a single performance core. The estimation is based on 15 FMA instructions with 16 FLOPs per 7.5 cycles with an additional 0.3 cycles for broadcasting and ∼0.5–1 cycle for loop math.[1] Based on this, the expected RTF for the Piano-SSM XL model is approximately 0.19. However, the actual measured RTF is significantly higher at 0.44, indicating that there is a performance bottleneck unrelated to computational throughput. Next, the memory bandwidth of the L2 cache is considered. The throughput has been measured[2] at ∼115 GB/s,

[1] https://www.intel.com/content/www/us/en/docs/intrinsics-guide/index.html
[2] https://github.com/travisdowns/uarch-bench

and all models fit into the 1.25 MB L2 Cache size. Estimating the RTF based on memory limitations using the equation: Estimated RTF $=$ Model Size $\cdot \frac{f_s}{\text{L2 Speed}}$, the estimated RTF is 0.41 for the Piano-SSM XL model at 44.1 kHz. Looking at Figure Table 5.5, the L and S models also closely correlate with the estimated RTF. This suggests that memory access speed, rather than computational throughput, is the main limiting factor.

## 5.5 Interpretability

Figure 5.8 shows frequencies $\mathbf{f}$=$(\boldsymbol{\beta} \cdot \mathrm{f}_{s\_train}) \ / \ 2\pi$ over exponential decay time constants $\boldsymbol{\tau} = 1/(\boldsymbol{\alpha}_{\mathbf{r}} \cdot \mathrm{f}_{s\_train})$ of all the eigenvalues across the layers for the there Piano-SSM model configurations trained at all sampling rates (16 kHz, 24 kHz and 44.1 kHz). Each point corresponds to an eigenvalue of a state matrix within a specific model layer, with color coding representing different layers.

Layers later in the architecture tend to have smaller time constants and higher frequencies, indicating faster dynamics and shorter memory. Most frequency components lie within the range of standard MIDI notes, suggesting that the model learned the dynamics are well aligned with the Pianos dynamics. It is also important to note that a nonlinearity, in this case, `tanh` activation, is applied after each SSM layer. This transforms the linear dynamics into a nonlinear behavior, introducing harmonic contents and an amplitude-dependent response. While the linear analysis provides valuable insights into the model's latent modes, the overall system behavior is shaped by this nonlinearity. It cannot be fully described by the eigenstructure alone.

Figure 5.9 shows how the model transforms the MIDI input into an audio signal over its different layers. After the `tanh` nonlinearity, the summed output of each state space layer is shown. The last plot normalizes the audio back to the dataset distribution.

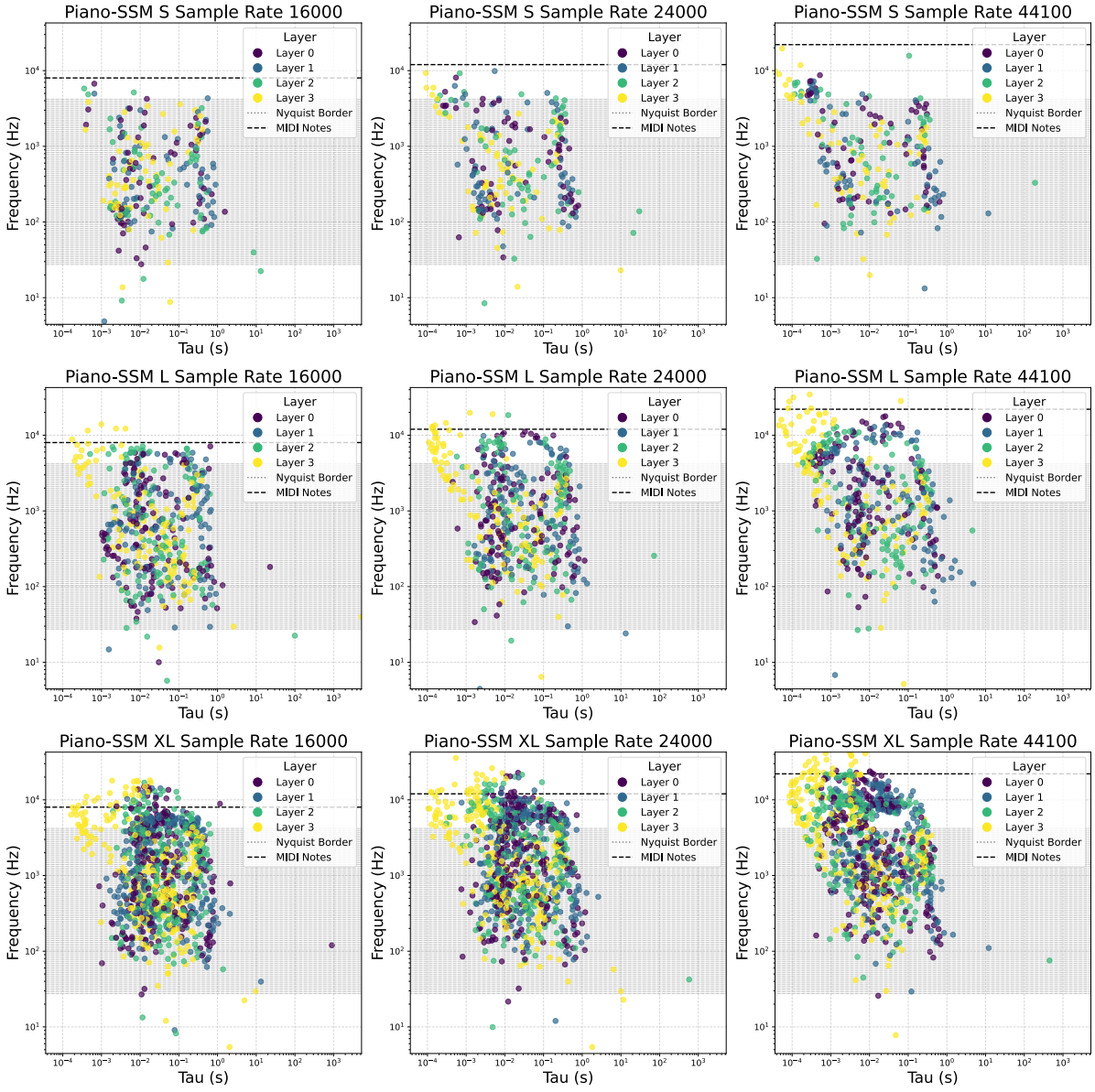



Figure 5.8: Frequencies $\mathbf{f}$ over exponential decay time constants Tau $\boldsymbol{\tau}$ for all eigenvalues across the layers of all Piano-SSM models (XL, L, S) trained on MAESTRO. Each point represents an eigenvalue in a specific layer, highlighting the learned dynamics. Smaller time constants $\boldsymbol{\tau}$ indicate faster internal states/memory decay. Each color represents a different layer within the architecture. The bold dotted horizontal line marks the Nyquist frequency (sampling_rate/2). Light-dotted horizontal lines indicate the positions of standard MIDI note frequencies. Equation 2.2.

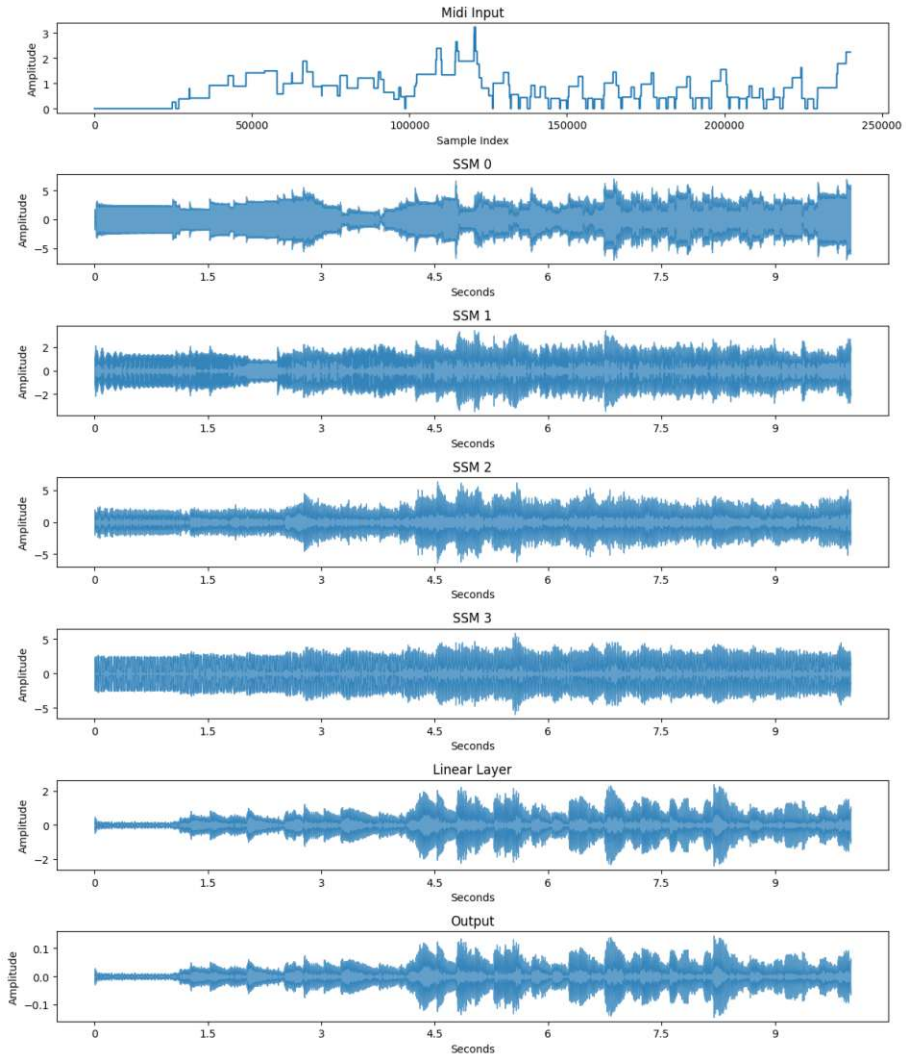



Figure 5.9: Audio signal evolution across layers. The first row shows the input MIDI sequence summed to one channel. After the nonlinearity, each row shows the summed audio output of the SSM layer. The shown piano performance is Domenico Scarlatti's *Sonata in B Minor, K. 27*

# Chapter 6

# Conclusion and Outlook

## 6.1 Conclusion

This work introduces a novel state-of-the-art neural network architecture, Piano-SSM, which leverages the advantages of SSMs without relying on intermediate representations or domain-specific expert knowledge. The Piano-SSM model is evaluated on the MAESTRO benchmark dataset, achieving comparable results to DDSP-Piano v1 and DDSP-Piano v2. Moreover, the model's capabilities to train at a high sampling rate and synthesize at a lower sampling rate with a slight decrease in audio quality are shown. The model is evaluated on the MAPS dataset to underline its generalization capabilities and ability to learn complex patterns from very limited training data. Furthermore, SSMs' efficient autoregressive and causal inference capabilities are shown, which enable real-time inference, even with the largest configuration at the highest sample rate. Moreover, the interpretability of the SSMs is shown by interpreting the eigenvalues of the state matrices, and the evolution of the audio sample over different layers.

## 6.2 Outlook

Although the Piano-SSM already shows remarkable performance in the task of MIDI to raw audio synthesis, several open research questions could not be answered in this thesis. The following points are open as outlooks for future research:

- **Interpretability and Downsampling.** When downsampling from a higher training sampling rate to a lower inference sampling rate, some eigenvalues represented in the tau frequency plot are higher than the Nyquist border of the new synthesis sampling rate. This could lead to unwanted aliasing effects. This topic has to be researched in the future and should be regularized or constrained during the training of the network.

- **Reverberation.** Renault et al. improved the DDSP-Piano v1 model by introducing the reverberation layer; they proved that the layer can capture the room acoustics of the different years of the MAESTRO dataset. Because of the improvement in DDSP-Piano, it is possible that the reverberation layer would also improve the Piano-SSM model. However, the encountered instability of the reverberation layer must be determined, or other alternatives to the proposed reverberation have to be found.

# Bibliography


[1] D. Dallinger, M. Bittner, D. Schnöll, M. Wess, and A. Jantsch, "Piano-SSM: Diagonal State Space Models for Efficient MIDI-to-Raw Audio Synthesis," *Under Submission*, 2025.

[2] D. Schnöll, D. Dallinger, M. Wess, M. Bittner, and A. Jantsch, "Towards Optimal Implementations of Neural Networks on Micro-Controller," *ITEM: IoT, Edge, and Mobile for Embedded Machine Learning at ECML PKDD*, 2024.

[3] M. Bittner, D. Schnöll, M. Wess, and A. Jantsch, "Efficient and Interpretable Raw Audio Classification with Diagonal State Space Models," *Under Submission*, 2025.

[4] L. Renault, "Neural audio synthesis of realistic piano performances," Theses, Sorbonne Université, Jul. 2024, issue: 2024SORUS196. [Online]. Available: https://theses.hal.science/tel-04732963

[5] X. Shi, E. Cooper, X. Wang, J. Yamagishi, and S. Narayanan, "Can Knowledge of End-to-End Text-to-Speech Models Improve Neural MIDI-to-Audio Synthesis Systems?" Mar. 2023, arXiv:2211.13868 [cs]. [Online]. Available: http://arxiv.org/abs/2211.13868

[6] E. Cooper, X. Wang, and J. Yamagishi, "Text-to-Speech Synthesis Techniques for MIDI-to-Audio Synthesis," Feb. 2022, arXiv:2104.12292 [cs]. [Online]. Available: http://arxiv.org/abs/2104.12292

[7] N. Li, S. Liu, Y. Liu, S. Zhao, M. Liu, and M. Zhou, "Neural Speech Synthesis with Transformer Network," Jan. 2019, arXiv:1809.08895 [cs]. [Online]. Available: http://arxiv.org/abs/1809.08895

[8] J. Kong, J. Kim, and J. Bae, "HiFi-GAN: Generative Adversarial Networks for Efficient and High Fidelity Speech Synthesis," Oct. 2020, arXiv:2010.05646 [cs]. [Online]. Available: http://arxiv.org/abs/2010.05646

[9] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention Is All You Need," Aug. 2023, arXiv:1706.03762 [cs]. [Online]. Available: http://arxiv.org/abs/1706.03762

[10] 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," Sep. 2016, arXiv:1609.03499 [cs]. [Online]. Available: http://arxiv.org/abs/1609.03499

[11] A. Gu, K. Goel, and C. Ré, "Efficiently Modeling Long Sequences with Structured State Spaces," Aug. 2022, arXiv:2111.00396 [cs]. [Online]. Available: http://arxiv.org/abs/2111.00396

[12] A. Gu, A. Gupta, K. Goel, and C. Ré, "On the Parameterization and Initialization of Diagonal State Space Models," Aug. 2022, arXiv:2206.11893 [cs]. [Online]. Available: http://arxiv.org/abs/2206.11893

[13] J. T. H. Smith, A. Warrington, and S. W. Linderman, "Simplified State Space Layers for Sequence Modeling," Mar. 2023, arXiv:2208.04933 [cs]. [Online]. Available: http://arxiv.org/abs/2208.04933

[14] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to Sequence Learning with Neural Networks," Dec. 2014, arXiv:1409.3215 [cs]. [Online]. Available: http://arxiv.org/abs/1409.3215

[15] B. Hayes, J. Shier, G. Fazekas, A. McPherson, and C. Saitis, "A Review of Differentiable Digital Signal Processing for Music & Speech Synthesis," Aug. 2023, arXiv:2308.15422 [cs]. [Online]. Available: http://arxiv.org/abs/2308.15422

[16] D. Smith and C. Wood, "The 'USI', or Universal Synthesizer Interface," *journal of the audio engineering society*, no. 1845, Oct. 1981.

[17] C. E. Shannon, "Communication Theory of Secrecy Systems*," *Bell System Technical Journal*, vol. 28, no. 4, pp. 656–715, Oct. 1949. [Online]. Available: https://ieeexplore.ieee.org/document/6769090

[18] A. Gu, T. Dao, S. Ermon, A. Rudra, and C. Re, "HiPPO: Recurrent Memory with Optimal Polynomial Projections," Oct. 2020, arXiv:2008.07669 [cs]. [Online]. Available: http://arxiv.org/abs/2008.07669

[19] K. Goel, A. Gu, C. Donahue, and C. Ré, "It's Raw! Audio Generation with State-Space Models," Feb. 2022, arXiv:2202.09729 [cs, eess]. [Online]. Available: http://arxiv.org/abs/2202.09729

[20] C. Hawthorne, A. Stasyuk, A. Roberts, I. Simon, C.-Z. A. Huang, S. Dieleman, E. Elsen, J. Engel, and D. Eck, "Enabling Factorized Piano Music Modeling and Generation with the MAESTRO Dataset," Jan. 2019, arXiv:1810.12247 [cs]. [Online]. Available: http://arxiv.org/abs/1810.12247

[21] H.-W. Dong, C. Zhou, T. Berg-Kirkpatrick, and J. McAuley, "Deep Performer: Score-to-Audio Music Performance Synthesis," Feb. 2022, arXiv:2202.06034 [cs]. [Online]. Available: http://arxiv.org/abs/2202.06034

[22] Y. Wang, R. J. Skerry-Ryan, D. Stanton, Y. Wu, R. J. Weiss, N. Jaitly, Z. Yang, Y. Xiao, Z. Chen, S. Bengio, Q. Le, Y. Agiomyrgiannakis, R. Clark, and R. A. Saurous, "Tacotron: Towards End-to-End Speech Synthesis," Apr. 2017, arXiv:1703.10135 [cs]. [Online]. Available: http://arxiv.org/abs/1703.10135

[23] X. Wang, S. Takaki, and J. Yamagishi, "Neural source-filter waveform models for statistical parametric speech synthesis," Nov. 2019, arXiv:1904.12088 [eess]. [Online]. Available: http://arxiv.org/abs/1904.12088

[24] L. Renault, R. Mignot, and A. Roebel, "Differentiable Piano Model for MIDI-to-Audio Performance Synthesis," Sep. 2022, publisher: Zenodo. [Online]. Available: https://zenodo.org/record/7092602

[25] L. Renault, R. Mignot, and A. Roebel., "DDSP-Piano: A Neural Sound Synthesizer Informed by Instrument Knowledge," *Journal of the Audio Engineering Society*, vol. 71, no. 9, pp. 552–565, Sep. 2023. [Online]. Available: https://www.aes.org/e-lib/browse.cfm?elib=22231

[26] J. Engel, L. Hantrakul, C. Gu, and A. Roberts, "DDSP: Differentiable Digital Signal Processing," Jan. 2020, arXiv:2001.04643 [cs]. [Online]. Available: http://arxiv.org/abs/2001.04643

[27] S. Lee, H.-S. Choi, and K. Lee, "Differentiable Artificial Reverberation," Jul. 2022, arXiv:2105.13940 [cs]. [Online]. Available: http://arxiv.org/abs/2105.13940

[28] V. Emiya, N. Bertin, B. David, and R. Badeau, "MAPS - A piano database for multipitch estimation and automatic transcription of music," Research Report, Jul. 2010. [Online]. Available: https://inria.hal.science/inria-00544155

[29] N. Jonason, X. Wang, E. Cooper, L. Juvela, B. L. T. Sturm, and J. Yamagishi, "DDSP-based Neural Waveform Synthesis of Polyphonic Guitar Performance from String-wise MIDI Input," Sep. 2023, arXiv:2309.07658 [cs]. [Online]. Available: http://arxiv.org/abs/2309.07658

[30] B. McFee, M. McVicar, D. Faronbi, I. Roman, M. Gover, S. Balke, S. Seyfarth, A. Malek, C. Raffel, V. Lostanlen, B. van Niekirk, D. Lee, F. Cwitkowitz, F. Zalkow, O. Nieto, D. Ellis, J. Mason, K. Lee, B. Steers, E. Halvachs, C. Thomé, F. Robert-Stöter, R. Bittner, Z. Wei, A. Weiss, E. Battenberg, K. Choi, R. Yamamoto, C. Carr, A. Metsai, S. Sullivan, P. Friesch, A. Krishnakumar, S. Hidaka, S. Kowalik, F. Keller, D. Mazur, A. Chabot-Leclerc, C. Hawthorne, C. Ramaprasad, M. Keum, J. Gomez, W. Monroe, V. A. Morozov, K. Eliasi, nullmightybofo, P. Biberstein, N. D. Sergin, R. Hennequin, R. Naktinis, beantowel, T. Kim, J. P. Åsen, J. Lim, A. Malins, D. Hereñú, S. van der Struijk, L. Nickel, J. Wu, Z. Wang, T. Gates, M. Vollrath, A. Sarroff, Xiao-Ming, A. Porter, S. Kranzler, Voodoohop, M. D. Gangi, H. Jinoz, C. Guerrero, A. Mazhar, toddrme2178,

Z. Baratz, A. Kostin, X. Zhuang, C. T. Lo, P. Campr, E. Semeniuc, M. Biswal, S. Moura, P. Brossier, H. Lee, and W. Pimenta, "librosa/librosa: 0.10.2.post1," May 2024. [Online]. Available: https://zenodo.org/doi/10.5281/zenodo.11192913

[31] I. Loshchilov and F. Hutter, "SGDR: Stochastic Gradient Descent with Warm Restarts," May 2017, arXiv:1608.03983 [cs]. [Online]. Available: http://arxiv.org/abs/1608.03983

[32] N. Katsura, "Cosine Annealing with Warmup for PyTorch," Mar. 2025, publication Title: GitHub repository. [Online]. Available: https://github.com/katsura-jp/pytorch-cosine-annealing-with-warmup

[33] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Y. Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng, "TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems," 2015. [Online]. Available: https://www.tensorflow.org/

[34] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," Dec. 2019, arXiv:1912.01703 [cs]. [Online]. Available: http://arxiv.org/abs/1912.01703

[35] C. Steinmetz and J. D. Reiss, "auraloss: Audio-focused loss functions in PyTorch," 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:268231964

[36] J. Shen, R. Pang, R. J. Weiss, M. Schuster, N. Jaitly, Z. Yang, Z. Chen, Y. Zhang, Y. Wang, R. J. Skerry-Ryan, R. A. Saurous, Y. Agiomyrgiannakis, and Y. Wu, "Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions," Feb. 2018, arXiv:1712.05884 [cs]. [Online]. Available: http://arxiv.org/abs/1712.05884