

Ludwig-Maximilians-Universität München  
Oettingenstraße 67  
D–80538 München  
Institut für Informatik

Bachelorarbeit

# **Text-zu-spielbarem-Klang: Synthesizer basierend auf Latent-Diffusion-Technologie**

Pierre-Louis Wolfgang Léon Suckrow

suckrowpierre@gmail.com

**Studiengang:** Informatik

**Prüfer/in:** Prof. Dr. Sylvia Rothe

**Betreuer/in:** Christoph Weber

**Beginn am:** Mai 31, 2023

**Beendet am:** Oktober 19, 2023

„Text-zu-spielbarem-Klang: Synthesizer basierend auf Latent-Diffusion-Technologie“  
19. Oktober 2023

© 2023 Pierre-Louis Wolfgang Léon Suckrow

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 License (CC BY-SA 4.0):  
<http://creativecommons.org/licenses/by-sa/4.0/>  


Typesetting: PDF-L<sup>A</sup>T<sub>E</sub>X 2<sub>&</sub>

## Kurzfassung

In dieser Arbeit wurde die Integration und Anwendbarkeit generativer künstlicher Intelligenz im Bereich der Musikproduktion durch die Einführung eines digitalen Instruments analysiert. Mittels ausgewählter *Diffusions*-Modelle können Nutzer\*innen Klänge durch textliche Beschreibungen definieren und diese mit standardisierten Musikproduktionswerkzeugen spielen und manipulieren. Die verwendeten *Diffusions*-modelle wurden hinsichtlich ihrer Tauglichkeit im gegebenen Kontext bewertet und für die Integration in ein digitales Instrument modifiziert. Unter Verwendung bestimmter Frameworks wurde das digitale Instrument erstellt, das in einer Benutzeroberfläche resultiert. Diese ermöglicht es den Anwendern, modell- und instrumentspezifische Parameter zu bearbeiten. Die Analyse zeigte, dass die eingesetzten Modelle nicht immer adäquat auf den Kontext der Musikproduktion reagieren, was in unerwarteten Klangbildern oder abstrakten Artefakten resultierte. Aktuell bieten verfügbare Text-zu-Audio-Modelle keine qualitativ hochwertige Reproduktion vertrauter Klänge, bieten jedoch Möglichkeiten für experimentelle Anwendungen. Die Implementierung eines Prototypen des digitalen Instruments ermöglicht solche Experimente und die Exploration des innovativen Klangsyntheseverfahren. Aktuell fehlen jedoch Funktionen, ausgewählte Bereiche der erzeugten Klänge wiederzugeben oder sie beliebig lange zu spielen. Dennoch können bereits jetzt interessante und ungewöhnliche Klanglandschaften erzeugt werden, die Anwendung in musikalischen Kompositionen finden könnten.



## **Abstract**

In this work, the integration and applicability of generative artificial intelligence in the field of music production was analyzed through the introduction of a digital instrument. Using selected diffusion models, users can define sounds through textual descriptions and play and manipulate them with standardized music production tools. The diffusion models used were evaluated for their suitability in the given context and modified for integration into a digital instrument. Using certain frameworks, the digital instrument was created, resulting in a user interface. This allows users to edit model and instrument-specific parameters. The analysis showed that the models used do not always adequately respond to the context of music production, resulting in unexpected sound patterns or abstract artifacts. Currently, available text-to-audio models do not provide high-quality reproduction of familiar sounds but offer opportunities for experimental applications. The implementation of a prototype of the digital instrument allows for such experiments and the exploration of innovative sound synthesis methods. However, functions are currently missing to reproduce selected areas of the generated sounds or to play them indefinitely. Nevertheless, interesting and unusual soundscapes can already be produced, which could find application in musical compositions.



# Inhaltsverzeichnis

<b>1. Einleitung</b>	<b>11</b>
<b>2. Related Work</b>	<b>13</b>
<b>3. Grundlagen</b>	<b>21</b>
3.1. Klangsynthese und Musikproduktion . . . . .	21
3.2. Synthesizer mittels Diffusion . . . . .	26
<b>4. Methoden</b>	<b>37</b>
4.1. Implementierung des Neuronalen Synthesizers . . . . .	37
4.2. Verteilung und Bündelung . . . . .	39
4.3. Benutzeroberfläche . . . . .	39
4.4. Evaluation . . . . .	40
<b>5. Ergebnisse</b>	<b>43</b>
<b>6. Diskussion</b>	<b>47</b>
<b>7. Schlussfolgerung</b>	<b>49</b>
<b>Literaturverzeichnis</b>	<b>51</b>
<b>A. Veröffentlichung</b>	<b>61</b>



# Abbildungsverzeichnis

2.1. AudioLM Architektur . . . . .	14
2.2. Diffsound Architektur . . . . .	15
2.3. AudioGen Architektur . . . . .	16
2.4. Tango Architektur . . . . .	16
2.5. Make-An-Audio Module . . . . .	17
2.6. MusicLM Module . . . . .	18
2.7. MusicGen EnCodec . . . . .	19
3.1. Fourier Reihe . . . . .	22
3.2. <i>Spektren, Spektrogramme und Mel-Spektrogramme</i> . . . . .	23
3.3. Synth . . . . .	25
3.4. Hüllkurve . . . . .	25
3.5. Vorwärtsprozess Diffusion . . . . .	27
3.6. Rückwärtsprozess Diffusion . . . . .	27
3.7. LDM Architektur . . . . .	29
3.8. Clap Architektur . . . . .	30
3.9. AudioLDM Architektur . . . . .	32
3.10. AudioLDM2 Architektur . . . . .	34
4.1. AudiLDM Anbindung . . . . .	38
4.2. Benutzeroberfläche . . . . .	40



# 1. Einleitung

„Es gibt keine theoretischen Grenzen für den Computer als Quelle für musikalische Klänge, im Gegensatz zu herkömmlichen Instrumenten“<sup>1</sup> [72]. Mit diesen wegweisenden Worten publizierte Max Vernon Mathews im Jahr 1963 seinen Artikel „The Digital Computer as a Musical Instrument“ und legte durch seine Arbeit an den Bell Laboratories den Grundstein für die heutige Musikproduktion und Klangsynthese am Computer [73].

In Anlehnung an die Überlegungen von M. V. Mathews konzentriert sich die vorliegende Arbeit auf die Erforschung von Implementierungsmöglichkeiten eines *Synthesizers* unter Verwendung der neuartigen *Latent Diffusions*-Technologie [94]. Es werden sowohl das resultierende digitale Instrument als auch die potenziellen Vorteile und Limitationen dieser Implementierung betrachtet. Zudem wird das Potenzial der benutzten *Diffusions*-Modelle im Kontext musikalischer Anwendungen diskutiert. Die vorgestellte Realisierung beabsichtigt, Nutzer\*innen Klangbilder gemäß individueller Präferenzen zu generieren und diese folglich zu modifizieren und spielbar zu gestalten. Das zentrale Ziel dieser Arbeit besteht darin, die verschiedenen Implementierungsansätze sowie die Ergebnisse des digitalen Instruments und der eingesetzten *Diffusions*-Modelle systematisch zu analysieren, ihre jeweiligen Stärken und Schwächen herauszuarbeiten und die Tauglichkeit als innovatives Musikinstrument zu bewerten. Ein weiteres Ziel ist es, das Instrument so zu gestalten, dass es nahtlos in den zeitgenössischen Musikproduktionsprozess integriert werden kann, wodurch Musiker\*innen der Zugang zu neuen Klangwelten und Syntheseverfahren ermöglicht wird. Die Erzeugung eigens definierter Klänge könnte darüber hinaus bestimmte rechtliche Problematiken, die durch den Einsatz von Samples auftreten, obsolet machen. Im weiteren Verlauf dieser Arbeit wird die Machbarkeit eines solchen Instruments unter Berücksichtigung von zwei spezifischen *Diffusions*-modellen für die Audiosynthese erörtert.

Vereinzelte Projekte, darunter der von Google im Jahr 2017 entwickelte *NSynth* [36] oder das Audio-Plugin *Neutone* [88], haben bereits die Konzeption von Instrumenten mithilfe von künstlicher Intelligenz in Angriff genommen. Diese Ansätze erlernen auditive Charakteristika und wenden diese auf ein Signal an bzw. übertragen diese. Die jüngsten Fortschritte in der Entwicklung generativer Künstlicher Intelligenz, wie etwa *Diffusion* [24, 41, 76, 102], lassen sich auf die Audiosynthese übertragen. Durch die Möglichkeit, das Ergebnis dieser Modelle präzise durch eine Texteingabe zu bestimmen, erhalten Musiker\*innen, Musikproduzent\*innen und Sounddesigner\*innen eine neuartige Kontrollmethode zur Definition gewünschter Klangattribute. Ein ähnlicher Ansatz in Form einer Implementierung eines digitalen Instrumentes mit Audiosynthese durch generative Künstliche Intelligenz ist die Anwendung *VroomAI* [7]. Obwohl beide Ansätze ähnliche Modelle verwenden, bietet *VroomAI* keine Möglichkeit, Modell- und Audioparameter anzupassen oder zu modifizieren. Die hier dargestellte Implementierung schließt zudem neuere Modelle ein, die zum Zeitpunkt der Veröffentlichung von *VroomAI* noch nicht verfügbar waren.

In der vorliegenden Arbeit werden die Text-zu-Audio-*Diffusions*-modelle *Audio Latent Diffusion Model (AudioLDM)* [63] und *Audio Latent Diffusion Model 2 (AudioLDM2)* [64] verwendet, um das Potenzial von künstlicher Intelligenz im Bereich der Audiosynthese in einem musikalischen Kontext aufzuzeigen. Der Fokus liegt dabei auf der Untersuchung dieser neuen Methode zur Klangerzeugung und der Bewertung ihres

---

<sup>1</sup> "There are no theoretical limitations to the performance of the computer as a source of musical sounds, in contrast to the performance of ordinary instruments"

## 1. Einleitung

performativen Potenzials. Insbesondere die Möglichkeit einer interaktiven Bereitstellung von Einstellungsmöglichkeiten für Parameter des jeweiligen Modells und die Klangwiedergabe, welche in vergleichbaren Projekten bisher fehlte, soll Instrumentalist\*innen zusätzliche Kontrolle über die gewünschten Eigenschaften verleihen.

## 2. Related Work

Die Entwicklung eines Instruments, das die Soundsynthese durch künstliche Intelligenz ermöglicht, wurde im Jahr 2023 im Rahmen des von *Juce* [51] organisierten Wettbewerbs „Neural Audio Plugin“ verfolgt. Insbesondere zeichnete sich das Projekt *VroomAi* [7] durch die Nutzung desselben C++ Audio Frameworks wie im vorliegenden Projekt aus. In *VroomAi* wurde ein digitales Instrument entwickelt, welches über eine Benutzeroberfläche die Eingabe eines benutzerspezifischen *Prompts* zur Klangerzeugung ermöglicht. Anschließend kann dieser Klang durch den Einsatz eines *Samplers* spielbar gemacht werden.

*VroomAi* stellt zwei Modelle zur Verfügung: Das Modell *AudioLDM* [63], welches in dieser Arbeit implementiert ebenfalls benutzt wird, und das Modell *AudioLM* [8]. Das Modell *AudioLDM2* wurde bis zum Veröffentlichungsdatum dieser Arbeit noch nicht publiziert und findet daher keine Anwendung. Die in *VroomAi* implementierte Architektur ähnelt jener der in dieser Arbeit vorgestellten Umsetzung. Ihr Hauptzweck besteht darin, die Modelle auszuführen und Klänge zu generieren. Hierfür wird auf einen *Gradio* Server [37] zurückgegriffen, der das Modell lokal ausführt.

Das lokale Ausführen des Modells weist bestimmte Schwachstellen auf (siehe Kapitel 4.1.3). Insbesondere bei der Nutzung des von den *AudioLDM* Entwicklern bereitgestellten *Gradio*-Servers können Komplikationen entstehen. Das *AudioLDM* Modell lässt sich, ohne zusätzliche Anpassungen, nicht auf Computern ohne GPU betreiben. Dies kann für Anwender ohne Vorkenntnisse im Bereich ML-Modelle oder bei nicht geeigneter Hardware zu Herausforderungen führen. Ziel dieser Arbeit ist es, solche Hürden so weit wie möglich zu reduzieren und ein intuitiv zu bedienendes digitales Instrument zu entwickeln. Zusätzlich soll dem Anwender die Möglichkeit gegeben werden, modellspezifische sowie Audio-Parameter einzugeben – eine Funktion, die in *VroomAi* bisher nicht vorhanden ist.

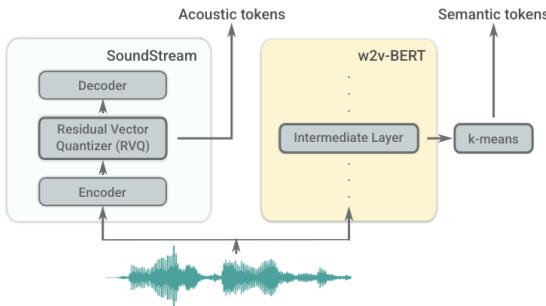
Neben *AudioLDM* [63] bieten sich folgende *Machine-Learning* Modelle an, um Klangerzeugung zu betreiben.

*AudioLM* [8] ermöglicht Audio-Synthese durch eine Kombination von *adversarial neural networks* [35], *self-supervised representation learning* [11] und *language modeling* [92] besteht. Es wurde auf der tokenisierten Darstellung von Wellenformen trainiert. Obwohl die Hauptfunktion von *AudioLM* in der Stimmgenerierung liegt, zeigte es, durch Training auf Klavieraufnahmen, ebenfalls Potenzial in der musikalischen Synthese. Das Modell besteht aus drei Kernelementen: einem *Tokenizer-Modell*, dem *Decoder* eines *Transformers-Sprachmodells* [110] und einem *Detokenizer-Modell*. Der Tokenizer wandelt kontinuierliche Audiowellenformen in eine diskrete Repräsentation um. Dabei verwendet es ein hybrides Tokenisierungsverfahren (siehe Abbildung 2.1a), das sowohl akustische als auch semantische Charakteristika (wie Sprachinhalte für Sprache oder Melodie und Rhythmus für Musik) der Wellenform erfasst. Während *SoundStream* [118] die akustischen Token generiert, stammen die semantischen Token von den Repräsentationen einer Zwischenschicht des *w2v-BERT* [11]. Der *Decoder* eines *Transformers-Sprachmodells* [110] lernt, Tokens basierend auf ihrem vorherigen Kontext zu antizipieren, indem die Abhängigkeiten und Struktur innerhalb der benutzen Sprache erfasst werden. Während der *Inferenz* prognostiziert das Modell aus einem gegebenen *Prompt autoregressiv* die Token-Sequenz. Dies impliziert, dass Tokens nacheinander generiert werden, wobei jeweils die zuvor erzeugten Tokens als Kontext dienen. Ein hierarchisches Vorgehen wird dabei angewendet (siehe Abbildung 2.1b): Zuerst werden semantische Token für die komplette Sequenz modelliert, welche dann als Konditionen zur Vorhersage der akustischen Token dienen. Abschließend konvertiert das Detokenizer-Modell die Token-Sequenz zurück in ein Audiosignal. [8]

Das Modell *Diffsound* [115] wurde mit der Absicht entwickelt, Soundeffekte zu generieren. Die Erzeugung eines Audiosignals erfolgt anhand eines *Prompts*, unter Zuhilfenahme eines *Text Encoders*, eines *Vector Quantized Variational Autoencoders* (VQ-VAE), eines *Token Decoders* und eines *Vocoders* (siehe Abbildung

## 2. Related Work

---



**(a)** Das hybride Tokenisierungsverfahren zur Generierung semantischer Tokens mittels *w2v-BERT* [11] und akustischer Tokens über *SoundStream* [118] [8]

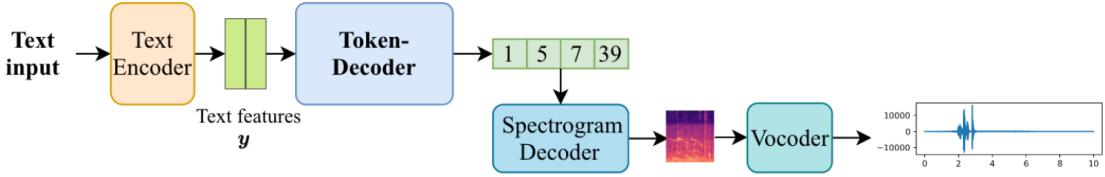


**(b)** Hierarchische Drei-Stufen-Modellierung von Tokens: Zunächst werden semantische Tokens für eine kohärente Struktur generiert. Diese werden anschließend durch grobe akustische Tokens ergänzt. Schließlich werden feine akustische Details in Tokens umgewandelt. Die abschließende Tokensequenz wird mittels *Decoder* in ein Audiosignal transformiert. [8]

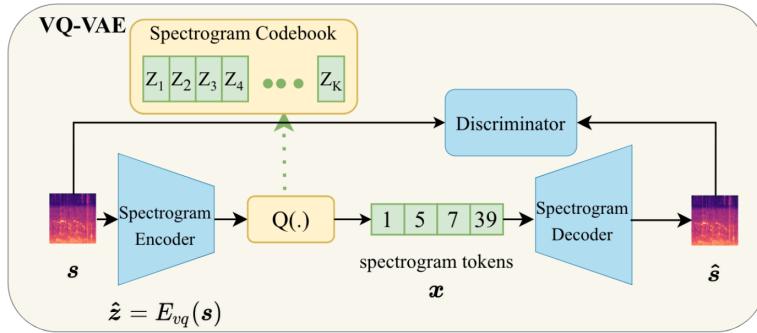
**Abbildung 2.1.:** AudioLM Module zur Extraktion von Tokens und Konstruktion eines Klanges aus diesen. [8]

2.2a). Der *Text Encoder* extrahiert relevante Audioinformationen aus dem eingegebenen Text und eliminiert dabei unwesentliche Daten. Dabei kommen ein vorab trainiertes *BERT* [23] und der *Text Encoder* eines bereits trainierten *Contrastive Language-Image Pre-Training (CLIP)* [89] zum Einsatz. Aus diesen Token wird ein *Mel-Spekrogramm* generiert, welches das künftige Audiosignal repräsentiert. Da *Mel-Spekrogramme* in eine Token-Sequenz überführt werden können, erstellt der *Token Decoder* diese basierend auf den vom *Text Encoder* generierten Token. Die Qualität des resultierenden Audiosignals hängt maßgeblich vom *Token Decoder* ab. Frühere Arbeiten [48, 65] setzten auf einen *autoregressiven* Ansatz, welcher aufeinanderfolgende Token basierend auf ihren Vorgängern vorhersagte. Ein solcher Ansatz kann allerdings zu kumulierten Fehlern führen, die sowohl die Rechenzeit als auch die Qualität beeinträchtigen. Zur Überwindung dieser Schwäche präsentiert das Paper einen *nicht autoregressiven Token Decoder*, der auf *Diskreter Diffusion* [6, 102] basiert. Anstatt die *Mel-Spekrogramm*-Token sequenziell vorherzusagen, prognostiziert *Diffsound* alle Token simultan und optimiert sie iterativ. Der *VQ-VAE* [80] lernt mithilfe eines *Discriminators* das Vokabular der Spektrogramm-Token, um ein Spektrogramm zu generieren (siehe Abbildung 2.2b). In einem abschließenden Schritt konvertiert der *Vocoder* das erstellte Spektrogramm in ein Audiosignal. Hierzu wurde der *Vocoder MelGAN* [60] mit dem *AudioSet* [31] Datensatz trainiert. Zusätzlich kam der Datensatz *AudioCaps* [53] sowohl für Training als auch Validierung zum Einsatz. Trotz der Fortschritte besitzt diese Arbeit Limitierungen: So ist das Generierungssystem nicht vollständig End-to-End und sowohl der *VQ-VAE*, der *Token-Decoder* als auch der *Vocoder* werden separat trainiert. [115]

Einen *autoregressiven* Ansatz verfolgend, verspricht *AudioGen* [59] eine verbesserte Audiogenerierung auf der Grundlage textueller Eingaben im Vergleich zu *Diffsound*. *AudioGen* operiert auf einer gelernten diskreten Audio-Repräsentation und ist in der Lage daraus Audiosignale zu generieren. Die Architektur dieses Modells besteht aus zwei Hauptkomponenten (siehe Abbildung 2.3). Die erste Komponente kodiert Audio mittels *neural audio compression* [118] in eine diskrete Tokensequenz. Das hierfür verwendete Modell wurde darauf trainiert, komprimierte Audiosignale zu rekonstruieren. Auf die durch diese erste Komponente erzeugten Audio-Tokens versucht ein *autoregressiven Transformer-decoder Sprachmodell*, das dem *GPT2*



(a) Umwandlung einer Texteingabe in ein Audiosignal: Mithilfe des *Text Encoder* und des *Token-Decoder* werden Tokens für ein Spektrogramm generiert. Dieses kann anschließend durch einen *Vocoder* in ein Audiosignal transformiert werden. [115]



(b) *Diffsound Vector Quantized Variational Autoencoder* für das Erlernen einer latenten Repräsentation von Spektrogrammen: Die Spektrogramme werden kodiert und anschließend rekonstruiert. Die Optimierung erfolgt mithilfe eines *Discriminator*. [115]

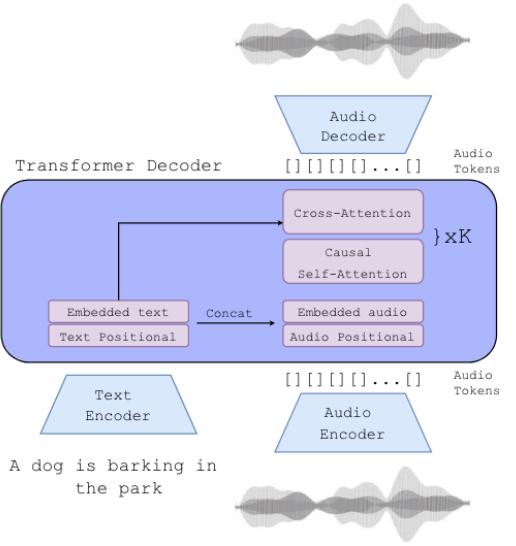
**Abbildung 2.2.: Diffsound Architektur: Ablauf der *Inferenz* und der *Vector Quantized Variational Autoencoder* [115]**

[4] ähnelt, unter Berücksichtigung von Textinformationen abzubilden. Während der *Inferenz* extrahiert das *Transformer*-Modell die zur Eingabe passenden Tokens, die anschließend durch den *Decoder* der ersten Komponente in Audio umgewandelt werden können. Das entwickelte Modell zeigt jedoch Schwierigkeiten, temporale Umformulierungen korrekt zu verarbeiten. [59]

Das Modell *Tango* [32] zieht Inspiration aus *AudioLDM* und verfolgt das Ziel, Text in Audio mittels *latenter Diffusion* zu transformieren. Es hebt sich in bestimmten Aspekten von *AudioLDM* ab und strebt an, überlegene Resultate trotz reduziertem Datensatz zu erzielen. Die Konstruktionsweise setzt sich aus drei zentralen Bestandteilen zusammen (siehe Abbildung 2.4): einem *Text Encoder* zur Verarbeitung der Eingabe-*Prompts*, einem *Latent Diffusionsmodell (LDM)* [94], und einem *Variational Autoencoder (VAE)* [54]. Der *Text Encoder*, angereichert durch das *Large Language Model (LLM)* *FLAN-T5-LARGE* [13], formt eine textbasierte Darstellung des gegebenen Inputs. Untersuchungen [20] zufolge vermögen *FLAN-T5-Modelle* durch ihr extensives Vortraining - reich an Gedankenstrukturen, Argumentationslinien und Anweisungen - rasch und wirkungsvoll neue Aufgaben zu erlernen. Dieses intensive Vortraining, das älteren Textmodellen wie *RoBERTa* [68] fehlte, könnte die Betonung essenzieller Details vereinfachen, was eine optimierte Transformation von Texten in ihre auditiven Pendants begünstigen könnte. Das *LDM*, adaptiert von *AudioLDM*, wandelt die textbasierte Darstellung des *Prompts* in eine latente Audio-Abbildung mittels *Spektrogramm*-Tokens um. Dieser Vorgang beruht auf dem synchronisierten Wechselspiel von Vorwärts- und Rückwärts-*diffusion*: Dem Audio wird *Rauschen* beigemischt, welches anschließend unter Anleitung des Textes wieder entfernt wird. Bei der Auswertung wird der trainierte Rückwärts *Diffusions*-prozess genutzt, um die Audio-Darstellung zu formen. Der *VAE*-Decoder wandelt dies dann in ein *Mel-Spektrogramm* um. Der *Vocoder HiFi-GAN* [57], identisch zu dem in *AudioLDM*, konvertiert das *Spektrogramm* letztlich in ein Audiosignal. Eine Schwäche des Modells liegt in der Erstellung nuancierter Audio-Outputs für ähnliche

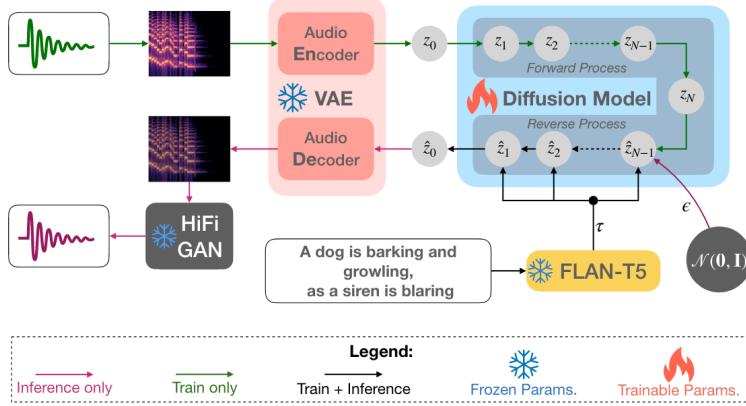
## 2. Related Work

---



**Abbildung 2.3.:** Architektur von *AudioGen*: Zusammensetzung aus *Encoder* und *Decoder* zum Erlernen des latenten Raums sowie dem *Transformer*-Modell, das auf diesen operiert. [59]

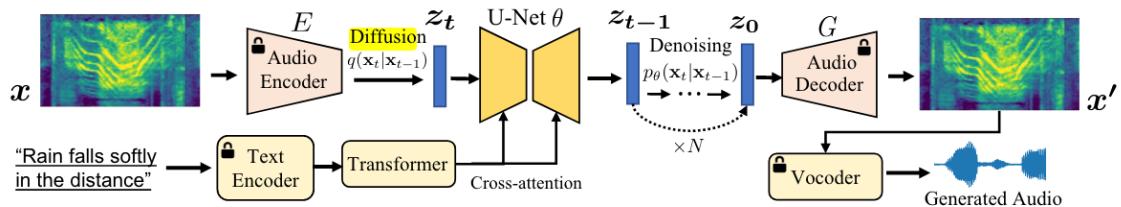
Texte, besonders wenn es nur auf kleineren Datensätzen trainiert wurde. Die Autoren empfehlen daher, dieses Hindernis durch Training auf umfangreicheren Datensätzen zu überbrücken, um die Kapazitäten des Modells in Bezug auf Differenzierung und Generierung von diversen Text-Audio-Kombinationen zu steigern. [32]



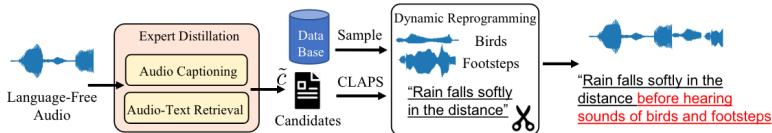
**Abbildung 2.4.:** Architektur von *Tango*: Ein Spektrogramm-*Encoder* und *Decoder* zur Erstellung einer latenten Repräsentation, ergänzt durch den *Diffusions*-Prozess und den *Text-Encoder*. [68]

Ein zusätzliches Modell, das sich der Methode der *Latent Diffusion* [94] bedient, um Text in Audio zu konvertieren, ist *Make-An-Audio* [43]. Parallel zur Entwicklung dieses Modells (siehe Abbildung 2.5a) wurde eine Methode konzipiert, um ungelabelten Audiosignalen eine Beschreibung zuzuweisen (siehe Abbildung 2.5b). Diese Methode soll den Mangel an gelabelten Audiosignalen adressieren. Der Prozess zur Umschreibung ungelabelter Audiosignale gliedert sich in zwei Stufen. In der ersten Stufe kommen vortrainierte Expertenmodelle [22, 55, 114] zum Einsatz, um Wissen zu destillieren. Diese Modelle generieren

Texte, die das Audiosignal beschreiben. Der Text mit der höchsten *CLAP*-Bewertung [28] wird als finaler Text ausgewählt. Zur Vermeidung von Overfitting im Modell wird in der zweiten Stufe aus einer Datenbank mit gelabelten Audioereignissen bis zu zwei Signale extrahiert und mit dem aus dem ersten Schritt generierten Paar kombiniert. Der *Diffusions*-Prozess arbeitet auf einer latenten Repräsentation des Audios, die durch einen Spektrogramm-*Autoencoder* erlernt wurde. Ein *Encoder*-Netz empfängt ein Spektrogramm und komprimiert dieses in den *Latentenraum*. Ein *Decoder* zielt darauf ab, das komprimierte Audiosignal aus dem latenten Raum zu rekonstruieren. Das Modell wird daraufhin trainiert, den durch die Rekonstruktion entstandenen Verlust zu minimieren und mit einem zusätzlichen *Diskriminator*, der das Unterscheiden zwischen echten und generierten Audiosignalen erlernt. Ein *Vocoder* kann schließlich das rekonstruierte Spektrogramm in ein Audiosignal umwandeln. Abseits von textbasierten Eingaben können Audiosignale auch aus anderen Quellen, wie Bildern und Videos, generiert werden. [43]



(a) Architektur von Make-An-Audio: Während des Trainings bleiben die Parameter der mit einem Schloss gekennzeichneten Module, darunter Text-Encoder, Audio-Encoder, Decoder und Vocoder, konstant. U-Net und der Diffusions-Prozess werden trainiert. [43]



(b) Destillation von Make-An-Audio zur Erweiterung des verfügbaren Trainingsdatensatzes [43]

**Abbildung 2.5.:** Module von Make-An-Audio zur Audio-Synthese und Erweiterung des Datensatzes umschriebener Audioereignisse. [43]

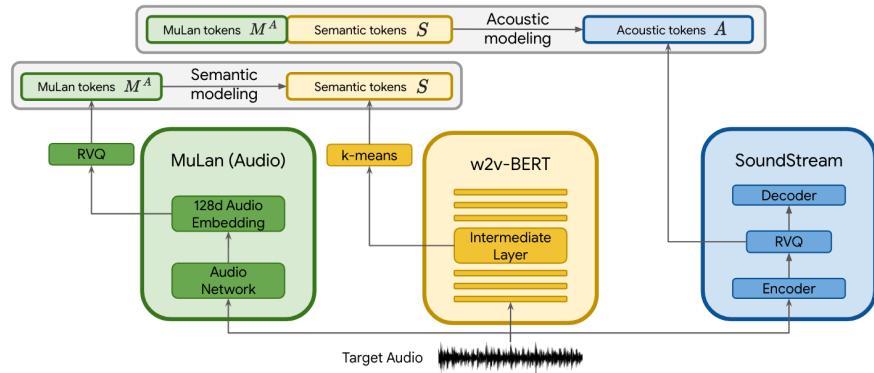
Die musikalische Anwendung neuronaler Netze beschränkt sich nicht ausschließlich auf die konditionierte Synthese von Klängen. Vielmehr ermöglichen solche Netzwerke auch eine zielgerichtete musikalische Generierung. Modelle wie *MusicLM* [3] und *MusicGen* [15] stützen sich auf bewährte Modelle und Ansätze der akustischen Klangsynthese. Sie erweitern diese jedoch durch die Fähigkeit, vollständige musikalische Audiosignale zu generieren, die kohärente Strukturen über längere Zeiträume beibehalten.

Das Modell *MusicLM* [3] basiert auf den Konzepten von *AudioLM*. Es integriert akustische Informationen über *Soundstream* [118], semantische Informationen mittels *w2v-Bert* [12] und Audio-Informationen mithilfe von *Mulan* [42] (siehe Abbildung 2.6a). Im Training dieses Modells erfolgte eine Abbildung des latenten Raums von *Mulan* zum latenten Raum von *w2v-Bert* und ebenso eine Abbildung von *w2v-Bert* zu *Soundstream* unter Verwendung von *Transformers*. Während der *Inferenz* können die erlernten Abbildungen genutzt werden. Eine von *Mulan* generierte Texteingabe wird hierbei in den latenten Raum von *SoundStream* transformiert, um anschließend in ein Audiosignal kodiert zu werden (siehe Abbildung 2.6b). [3]

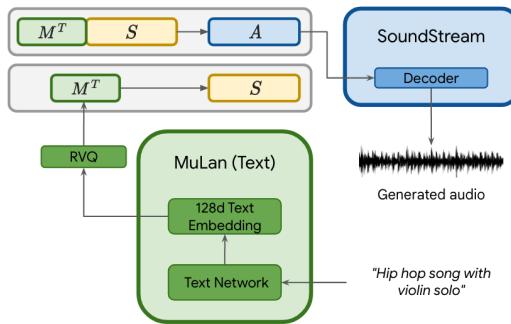
Im Gegensatz zu *MusicLM* [15] setzt *MusicGen* nicht auf mehrere kaskadierende Modelle, sondern verwendet einen *autoregressiven Transformer-basierten Decoder* [110], der auf Text oder Melodien, wie beispielsweise Summen, konditioniert werden kann. Dieses Modell operiert auf einer komprimierten Darstellung von gesampelten Audiosignalen, die durch *EnCodec* [21] (siehe Abbildung 2.7) generiert wird. *EnCodec* ist ein *convolutional auto-encoder*, der einen *Latentenraum* mithilfe von *Residual Vector Quantization* [118] erstellt. Abhängig von der Anzahl der in dem Quantifizierungsprozess verwendeten *codebooks* werden

## 2. Related Work

---



**(a)** Modellierung semantischer und akustischer Tokens erzeugt durch *MuLan*, *w2v-Bert* und *SoundStream*. [3]

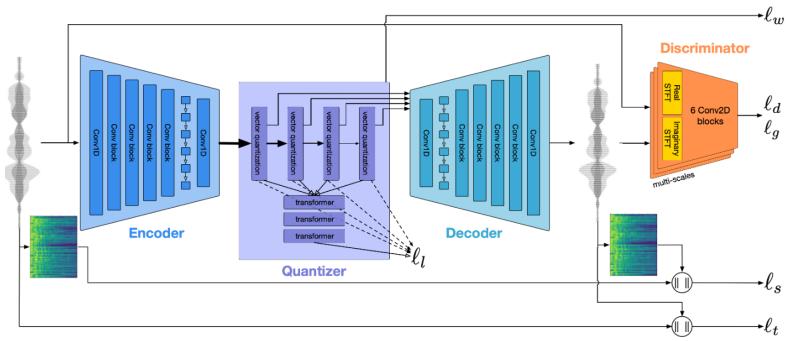


**(b)** *Inferenz* von MusicLM für Text-zu-Audio-Konversion mittels des Text-*Encoders* *MuLan* und der im Training erlernten Modellierungsverfahren. [3]

**Abbildung 2.6.:** Module von MusicLM für *Inferenz*-prozesse und Training. [3]

entsprechend viele Tokens generiert, die speziell angeordnet und anschließend dem *Decoder* von *EnCodec* übergeben werden. Dieser kann unter der gegebenen Konditionierung eine Audiodatei erstellen. Für die Textkonditionierung wurden Modelle wie *T5* [91], *FLAN-T5* [92] und *CLAP* evaluiert, während für Melodien eine Konditionierung über ein *Chromogramm* erfolgt. [15]

In dieser Arbeit wurden die generativen Audiomodelle *AudioLDM* [63] und *AudioLDM2* [64] für die Auswahl herangezogen. Durch die Integration in das von *Huggingface* [45] veröffentlichte Python-Paket *Diffusers* [85] mittels Anbindung [46, 47] ist eine effiziente Implementierung dieser Modelle gewährleistet. Die zuvor in *VroomAI* nicht vorhandene Möglichkeit Parameter einzustellen wurde ebenfalls berücksichtigt und implementiert.



**Abbildung 2.7.:** Einsatz von *EnCodec* [21] in *MusicGen* [15]: Eine *Encoder-Decoder*-Architektur basierend auf dem latenten Raum der *Residual Vector Quantization*, optimiert durch einen *Discriminator*. [21]



# 3. Grundlagen

## 3.1. Klangsynthese und Musikproduktion

### 3.1.1. Mathematische und physikalische Modellierung von Musik und Klang

„Musik ist eine Kunstform und kulturelle Aktivität, deren Medium der in Zeit organisierte Klang ist“<sup>1</sup> [109]. Im Gegensatz zu *Rauschen* weisen die Klänge, die Musik aufbauen, Strukturen und Zusammenhänge auf, welche für das menschliche Gehör als angenehm wahrgenommen werden [82].

Klang charakterisiert sich durch ein intrinsisches Zusammenspiel aus physikalischen und perzeptiven Elementen. Auf der physikalischen Ebene manifestiert sich der Klang als Welle, die durch einen schwingenden Körper erzeugt und von einem Ort zum anderen propagiert. Diese Welle setzt sich aus einem *Grundton* sowie mehreren resonierenden Einzeltönen zusammen. Der resultierende Klang weist eine Vielzahl von *Obertönen* auf, die die charakteristischen klanglichen Eigenschaften, auch *Klangfarben* genannt, hervorbringen. [82, 109]

Diese diversen *Töne* sind periodische Schwingungen, definiert durch eine *Tonhöhe/Frequenz*  $\omega$  (in Hz). Diese kann als Anzahl der Kompressionen an einem festgelegten Punkt pro Sekunde verstanden werden. Der Kehrwert der *Frequenz*, bezeichnet als *Periode*  $T = \frac{1}{\omega}$  (in s), gibt die Dauer an, die eine Kompression benötigt, um zwei identische Punkte zu durchlaufen. Töne mit einer niedrigen *Frequenz* erscheinen in unserer Wahrnehmung als tief und dumpf, wohingegen Töne mit hohen *Frequenzen* als leicht, schwebend und durchdringend empfunden werden. Die *Amplitude* der Schwingung beschreibt die transportierte Energie und demzufolge die Lautstärke eines Tones. Aufgrund des Abstandsgesetzes wird diese von einer Schallquelle über die logarithmische *Dezibel-Skala* (dB) angegeben. [82, 109]

Die Analyse der Struktur eines Klangs konzentriert sich auf die Beziehungen und Interaktionen der diversen Töne, die den Klang konstituieren. Die elementarste Darstellung eines Tones ist der Sinuston, welcher durch eine Sinuskurve repräsentiert wird. Gemäß dem *Fourier-Theorem* besteht jeder Ton aus einer Kombination verschiedener Sinustöne (siehe Abbildung 3.1). Dabei wird die tiefste dominierende *Frequenz* als *Grundton* und die höheren *Frequenzen* als *Obertöne* klassifiziert. Diese *Obertöne* variieren in ihrer *Frequenz*, *Amplitude* sowie ihrem zeitlichen Auf- und Abbau. Ein Oberton, dessen *Frequenz* ein ganzzahliges Vielfaches des *Grundtones* darstellt, wird als *harmonischer Oberton* definiert. Ist dies nicht der Fall, wird der *Oberton* als *inharmonisch* bezeichnet. Daraus folgt, dass verschiedene Instrumente, obwohl sie denselben Ton erzeugen, identische Grundschwingungen, aber unterschiedliche *harmonische* und *inharmonische Obertöne* aufweisen. [82, 97, 112]

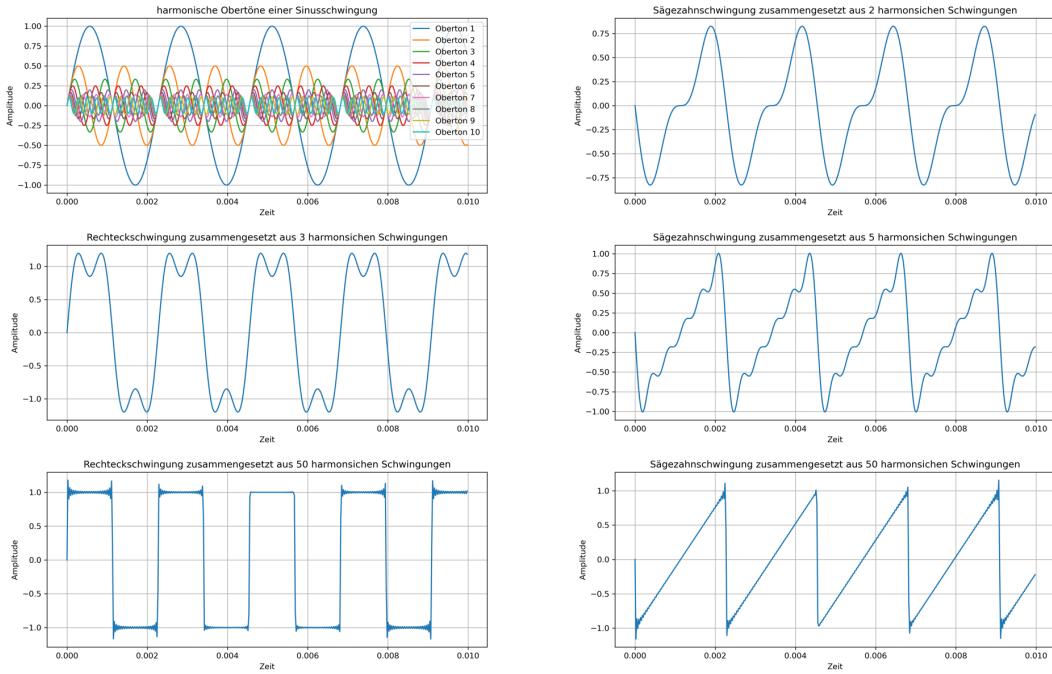
Die Beschaffenheit eines Klangs lässt sich durch die *Amplituden-faktoren* und die korrespondierenden *Frequenzen* der jeweiligen Töne beschreiben und in einem *Frequenzspektrum* (siehe Abbildung 3.2) darstellen. Diese Visualisierung ermöglicht die Identifikation dominanter *Frequenzen* oder *Frequenz*-bereiche innerhalb des Signals und bietet eine Illustration der *Klangfarbe* sowie der spezifischen Charakteristik des Klangs. Mittels einer *Fourier-Transformation* kann das zugehörige Spektrum für jedes gegebene Signal ermittelt werden. Umgekehrt erlaubt die *Inverse Fourier-Transformation* die Rekonstruktion des Signals basierend auf seinem Spektrum. [90]

---

<sup>1</sup> "Music is an art form and cultural activity whose medium is sound organized in time"

### 3. Grundlagen

---



**Abbildung 3.1.:** Zusammensetzung von Sägezahn- und Rechteckschwingungen aus harmonischen Oberschwingungen einer Sinuswelle.

Zur Berechnung einer *Diskreten Fourier-Transformation (DFT)* hat sich die *Fast Fourier-Transformation (FFT)* als effizienter Algorithmus etabliert [40]. Zur Realisierung der *Fourier-Transformation* in Echtzeit wird auf die *Short Time Fourier-Transformation* zurückgegriffen [107].

Ein *Spektrogramm* erlaubt die Analyse eines Signals sowohl im zeitlichen als auch im *Frequenz*-bereich (siehe Abbildung 3.2). Jedoch ist die Genauigkeit nicht stets gleichbleibend. Bei der Zerlegung des Signals in kurze zeitliche Segmente zur Spektrumsberechnung zeigen längere Segmente eine reduzierte zeitliche Auflösung, profitieren jedoch von einer präziseren Frequenzauflösung. Im Gegensatz dazu bieten kürzere Segmente eine höhere zeitliche Präzision, weisen aber eine geringere Frequenzauflösung auf. [90]

*Spektogramme*, die sich auf die *Mel-Skala* stützen, nennt man *Mel-Spektrogramm*. Diese Skala stellt akustische *Frequenzen* in einer annähernd logarithmischen Weise dar, sodass Tonhöhenintervalle, die ähnlich wahrgenommen werden, über den gesamten Hörbereich konstant abgebildet werden [62].

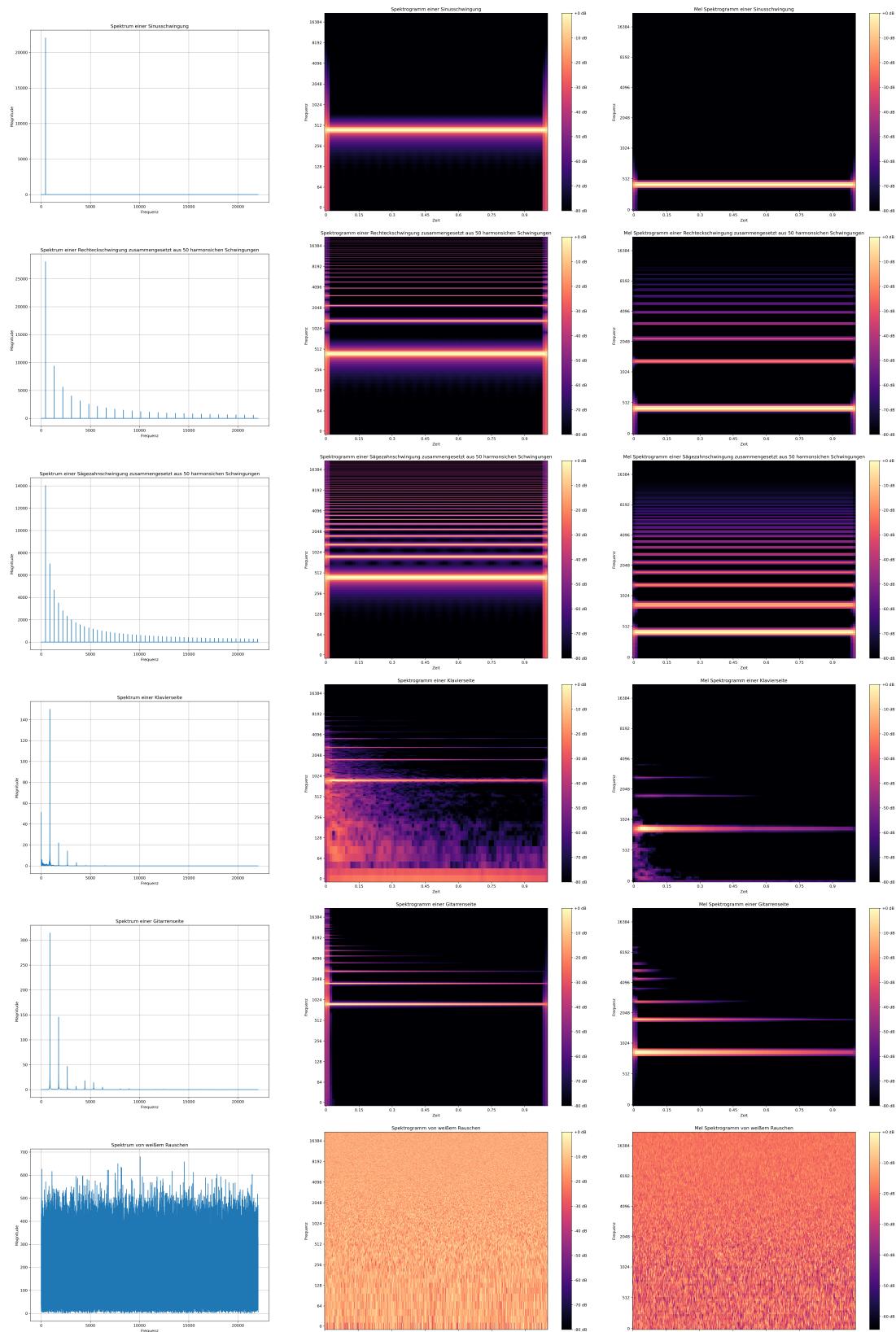
Ein Klang, der ein kontinuierliches *Frequenzspektrum* (siehe Abbildung 3.2) aufweist und viele Arten von Geräuschen aus unterschiedlichen Quellen beinhaltet, wird als *Rauschen* definiert [109].

Der für das menschliche Gehör wahrnehmbare *Frequenz*-bereich, welcher von 20Hz bis 20kHz reicht, lässt sich in drei Segmente unterteilen: *Bässe*, *Mitten* und *Höhen*. Die *Bässe* erstrecken sich von 20Hz bis 250Hz. Die tiefen *Mitten* umfassen den Bereich von 250Hz bis 2kHz, während sich die hohen *Mitten* von 2kHz bis 4kHz erstrecken. *Frequenzen* oberhalb von 4kHz werden als *Höhen* bezeichnet. [90]

#### 3.1.2. Digitale Audiorepräsentation und Sampling

Aufgrund der binären Beschaffenheit digitaler Computer erfordert die Darstellung und Verarbeitung von Audiosignalen eine Umwandlung des Signals von einem kontinuierlichen in einen diskreten Wertebereich. Das *Abtasttheorem* postuliert, dass analoge Signale redundante Informationen beinhalten, die bei einer Übertragung

### 3.1. Klangsynthese und Musikproduktion



**Abbildung 3.2.:** Spektren, Spektrogramme und Mel-Spektrogramme unterschiedlicher Klangcharakteristika.

### 3. Grundlagen

---

eliminiert werden können. Daher genügt es, das Audiosignal durch Erfassen von *Amplituden*-werten in regelmäßigen Intervallen zu reproduzieren und zu übermitteln. Um das Signal adäquat zu charakterisieren, sollte die Abtastrate wenigstens das Doppelte der maximalen *Frequenz* des Signals betragen. Diese Schranke wird als *Nyquist-Frequenz* bezeichnet und ist gleich  $2\omega_{\text{Hz}}$ . Da das menschliche Ohr lediglich *Frequenzen* bis zu 20kHz detektiert, wird in der Praxis oft eine *Abtastrate* von 44.1kHz angewendet. Unterschreitet die Abtastrate die *Nyquist-Frequenz*, resultiert ein Informationsverlust, und die rekonstruierte Wellenform kann von der ursprünglichen abweichen. In solch einem Szenario ist der Vorgang der Signalrekonstruktion nicht länger deterministisch, was als *Unterabtastung* bezeichnet wird. [61, 97, 101]

Die *Samplingtiefe* (engl. *Bitdepth*) definiert den Wertebereich, innerhalb dessen die erfassten *Amplituden*-werte dargestellt werden. Ein größerer Wertebereich ermöglicht eine präzisere Repräsentation des ursprünglichen Signals, erfordert jedoch auch einen erhöhten Speicherbedarf. Ist der Wertebereich hingegen eingeschränkt, müssen die *Amplitudenwerte* intensiver gerundet werden, was zu Artefakten und Unregelmäßigkeiten im Signal führen kann. [106]

*Sound-Sampling* oder schlicht *Sampling* in Verbindung mit digitaler oder analoger Klang-*Modifikation* stellt eine eigenständige Technik der Klangsynthese dar. Obwohl es im Wesentlichen eine Reproduktion des entnommenen Signals darstellt, geht es über eine bloße Speicherung hinaus. Prinzipien und Verfahren aus der analogen Synthese und Klanggestaltung können auf die Reproduktion des Klangs angewendet werden. Das resultierende Instrument wird als *Sampler* bezeichnet. Abhängig von seiner Komplexität ermöglicht er es Nutzer\*innen, das Signal in einem definierten Bereich zu wiederholen, zu verlangsamen, zu beschleunigen, zu invertieren oder spezifische Intervalle neu zu ordnen. Dies bietet auch die Gelegenheit zum musikalischen Zitieren, was jedoch rechtliche Fragen bezüglich des klanglichen Diebstahls für Musiker\*innen, Verlage und Plattenlabels aufwirft. [52, 97, 98]

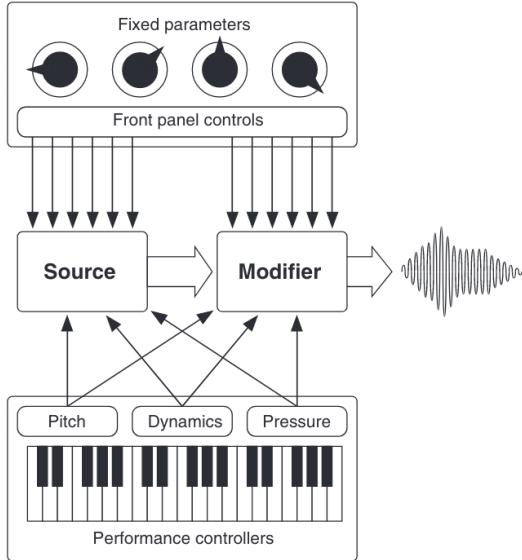
Die Wiederverwendung und Aufarbeitung von Teilen oder vollständigen, bereits produzierten Werken ist in der Industrie gängige Praxis. Dieses Vorgehen könnte durch den Einsatz generativer künstlicher Intelligenz vermieden werden.

#### 3.1.3. Synthesierung und Klangformung

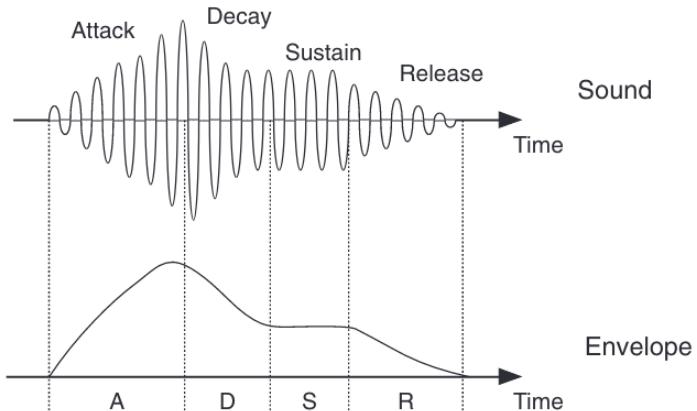
Instrumente, die dazu dienen, musikalische Klänge durch elektronische Spannung zu generieren und zu manipulieren, werden als *Synthesizer* bezeichnet [27, 84]. Die Bestandteile eines solchen Instruments lassen sich in drei Hauptkategorien einteilen (siehe Abbildung 4.2): *Erzeuger* (engl. *source*), *Modifikatoren* (engl. *modifier*) und *Kontrollinstanzen* (engl. *controls*). *Erzeuger*, wie beispielsweise Oszillatoren, generieren den Grundklang, welcher in diesem speziellen Projekt durch ein generatives *Diffusions*-Modell erzeugt wird. *Modifikatoren*, zu denen Filter und Effekte gehören, bearbeiten diesen generierten Klang. *Kontrollinstanzen* ermöglichen es Nutzer\*innen, die Parameter der *Erzeuger* und *Modifikatoren* zu definieren und flexibel anzupassen. Die Veränderung eines Parameters wird als *Modulieren* bezeichnet, während die spezifischen Einstellungen der Parameter als *Patch* gelten. [84]

Die für die Realisierung des Projekts erforderlichen Komponenten werden im Nachfolgenden erläutert.

In dem Bestreben, sein fundiertes wissenschaftliches Verständnis und seine Expertise in den Bau musikalischer Instrumente zu integrieren, konzipierte der Physiker und Komponist Hugh Le Caine erste Apparaturen zur elektronischen Klangsynthese - und das 20 Jahre vor der Kommerzialisierung der ersten analogen *Synthesizer* [116]. In diesem Zusammenhang formulierte Le Caine mehrere Grundsätze, die ein Instrumentenentwickler nach seiner Meinung verfolgen sollte. Er war der festen Überzeugung, dass ein Musiker maximale Kontrolle über die zentralen Parameter eines Klangs haben sollte. Dabei identifizierte er, dass die Ausdruckskraft eines Instruments insbesondere durch die Steuerung der *Anschlagsdynamik* über den gesamten Lautstärkebereich eines Tones, durch die Regulierung seiner *An- und Ausschwingzeiten* sowie durch die *Klangfarbe* bestimmt wird. [97]



**Abbildung 3.3.:** Bestandteile eines *Synthesizers*: *Erzeuger*, *Modifikatoren* und *Kontrollinstanzen*. [98]



**Abbildung 3.4.:** Beispiel einer Hüllkurve, definiert durch *Anschwellzeit*, *Abschwellzeit*, *Lautstärke-niveau* und *Ausklingzeit*. [98]

Allgemein wird das Ziel verfolgt, Töne zu generieren, die sich sowohl zeitlich als auch in ihrer *Frequenz* verändern oder übergehen, um akustisch ansprechende Ergebnisse zu erzielen [84]. Die Steuerung von *Anschlagsdynamik* sowie An- und Ausschwingszeiten wird durch einen *Hüllkurvengenerator* (engl. *envelope generator*) ermöglicht. Dieser Generator besteht in der Regel aus vier Teilen. Die *Anschwellzeit* (engl. *attack time*) gibt die Dauer an, die nach dem Anschlagen einer Taste vergeht, bis ein Ton seine höchste Lautstärke erreicht. Entsprechend bestimmt die *Abschwellzeit* (engl. *decay time*) den Zeitraum, in dem der Ton bis zu einem voreingestellten *Lautstärkeniveau* (engl. *sustain level*) abfällt. Dieses Niveau wird beibehalten, bis die Taste losgelassen wird. Danach kann eine Zeitspanne festgelegt werden, in der der Ton komplett ausklingt (engl. *release time*). Aufgrund der Bezeichnungen dieser Teile wird der Generator oft als *ADSR*-Generator (engl. *attack, decay, sustain, release*) (Abbildung 3.4) genannt. Instrumente wie Streichinstrumente, die mit

### 3. Grundlagen

---

einem Bogen gespielt werden, haben lange *Anschwell-, Abschwell- und Ausschwingzeiten*. Im Gegensatz dazu haben gezupfte Instrumente kürzere *Anschwellzeiten*. Klaviere und Schlaginstrumente zeichnen sich durch sehr schnelle *Anschwellzeiten* aus. [97, 98]

Die *Klangfarbe*, die durch den speziellen *Erzeuger* entsteht, kann durch Filter weiter verändert werden. In der Elektrotechnik werden diese verwendet, um bestimmte *Frequenzen* zu entfernen. Insbesondere der *Tiefpassfilter* (engl. *low-pass*) und der *Hochpassfilter* (engl. *high-pass*) sind wichtig, da sie entweder die höheren oder tieferen Teile des *Frequenzspektrums* reduzieren. Der Punkt, an dem diese Reduzierung beginnt, wird als *Beschneidungsfrequenz* (engl. *cut-off-frequency*) bezeichnet. Mit diesen Filtern kann man bestimmte *Obertöne* verstärken oder abschwächen. Wenn man diese Filter hintereinander schaltet, erhält man einen *Bandpassfilter*. Es ist auch möglich, *Obertöne* in der Nähe der Beschneidungsfrequenz zu betonen, was eine *Resonanz* (engl. *resonance*) nachahmt. [97]

*Kontrollinstanzen* können in zwei Gruppen eingeteilt werden. Die erste Gruppe beinhaltet Steuerelemente, die durch die Performance beeinflusst werden, wie z.B. ein Klaviaturkeyboard. Dies ermöglicht den Musiker\*innen, die *Tonhöhe* nach Wünschen zu verändern und so das Instrument zu steuern. Die zweite Gruppe umfasst feste Parameter, die spezifisch für den *Synthesizer* sind. Diese werden verwendet, um die Klangcharakteristik zu bestimmen. Dazu gehören zum Beispiel die Drehregler und Tasten eines *Synthesizers*. [98]

Das Kommunikationsprotokoll *MIDI (Musical Instrument Digital Interface)* [74] wurde eingeführt, um eine einheitliche Steuerung von Parametern und Kommunikation zwischen elektronischen Geräten zu gewährleisten. Es hat sich als globaler Standard in der elektronischen Musikbranche etabliert und erlaubt *MIDI-Instrumenten*, Daten auszutauschen. [97]

Die Einführung des *MIDI*-Protokolls hatte einen erheblichen Einfluss auf die Entwicklung und das Design von *Synthesizern*. Aufgrund der durch *MIDI* bedingten Standardisierung vieler Designaspekte von *Synthesizern* wurde die Klangerzeugungsmethode prominenter, während das funktionale Instrumentendesign weniger betont wurde. [98]

## 3.2. Synthesierung mittels Diffusion

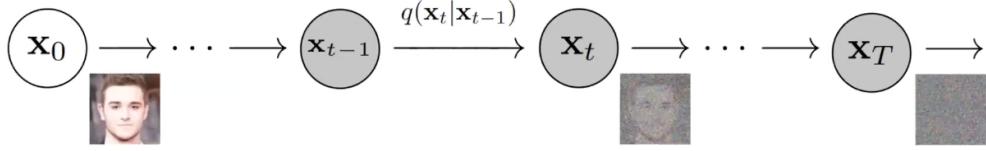
„Die Methoden der Klangerzeugung mit traditionellen mechanischen Musikinstrumenten waren seit ihrer Entstehung nur unwesentlichen Veränderungen unterworfen. [...] Der Instrumentenbau wandelte sich lediglich deshalb im Laufe der Zeit, weil man die Spielbarkeit der Instrumente zu verbessern und ihren Klangcharakter den wechselnden musikalischen Idealvorstellungen anzupassen wünschte. Anders dagegen im Bereich elektronischer Klangerzeugung. Hier werden ständig neue Methoden zur Synthese von Klängen entwickelt.“ [97]

Die innovative Technik der Soundsynthese durch *Diffusion* bietet verschiedene Vorteile. Sie könnte die Abhängigkeit von aufgezeichneten Klängen, deren Aufnahme oft kostenintensiv ist, reduzieren. Künstliche Intelligenz in der Sound-Synthese könnte diesen Sektor demokratisieren und mehr Menschen Zugang zu benötigten Ressourcen bieten. Zudem könnten Nutzer\*innen mehr Kontrolle über *Klangfarben* und -eigenschaften erhalten. Kreativ gesehen ermöglicht diese Methode die Schaffung einzigartiger Klänge, die Künstlern als Inspirationsquelle dienen könnten. [39]

### 3.2.1. Diffusion

Unter der Prämisse, dass die untersuchten Daten einer bestimmten Verteilung entstammen, zielen generative *Machine-Learning*-Modelle darauf ab, diese Verteilung zu identifizieren und zu approximieren. Die Erstellung neuer Daten erfolgt durch das Ziehen einer Probe aus dieser nachgebildeten Verteilung. [70]

Der Prozess der *Diffusion* im Maschinellen Lernen [24, 41, 76, 102] wurde von der statistischen Thermodynamik [49] und der sequenziellen Monte-Carlo-Methodik [75] inspiriert. Ziel ist es, iterative Verzerrungen einer Datenverteilung  $x_0 \sim q(\mathbf{x}_0)$  (*forward diffusion*) durch einen umgekehrten Prozess zu korrigieren (*reverse diffusion*). Das Ergebnis ist ein generatives Modell, das effizienter trainiert und bewertet werden kann als vergleichbare Ansätze. [76, 102]



**Abbildung 3.5.:** Vorwärts *Diffusions*-prozess: Schrittweise Korruption betrachteter Daten hin zu *isotropischem Rauschen*. [70]

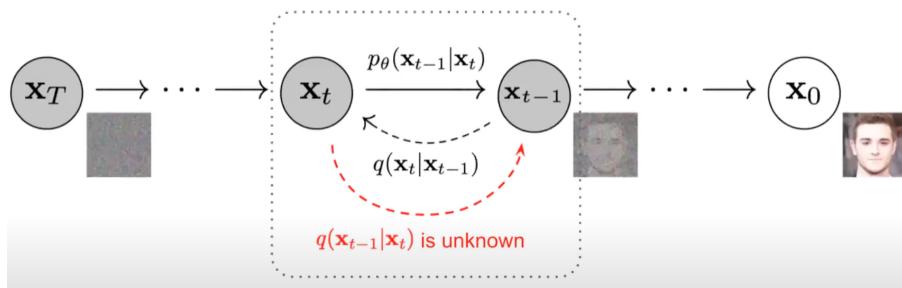
Der Vorwärtsprozess (Abbildung 3.5) ist als Markov-Kette gestaltet [41, 102]. Einem Datenpunkt  $x_0$  wird ein geringes *Rauschen* hinzugefügt, das durch den Hyperparameter  $\{\beta_t \in (0, 1)\}_{t=1}^T$ , auch als *noise schedule* bekannt, bestimmt wird [41, 70]. Hierbei wird *Gauss'sches Rauschen* [101] verwendet, repräsentiert durch eine Normalverteilung  $\mathcal{N}$ . Wiederholt man diesen Vorgang über  $T$  Schritte, werden sukzessive Informationen entfernt, bis schließlich nur *isotropisches Rauschen* verbleibt [70].

$$(3.1) \quad q\left(\mathbf{x}^{(t)} \mid \mathbf{x}^{(t-1)}\right) = \mathcal{N}\left(\mathbf{x}^{(t)}; \mathbf{x}^{(t-1)} \sqrt{1 - \beta_t}, \mathbf{I}\beta_t\right)$$

$$(3.2) \quad q\left(\mathbf{x}_{1:T} \mid \mathbf{x}_0\right) = \prod_{t=1}^T q\left(\mathbf{x}_t \mid \mathbf{x}_{t-1}\right)$$

Unter Verwendung der Relation  $\alpha_t := 1 - \beta_t$  und  $\bar{\alpha}_t := \prod_{s=1}^t \alpha_s$  lässt sich die Gleichung für den Vorwärtsprozess wie folgt formulieren [41]:

$$(3.3) \quad q\left(\mathbf{x}_t \mid \mathbf{x}_0\right) = \mathcal{N}\left(\mathbf{x}_t; \sqrt{\bar{\alpha}_t} \mathbf{x}_0, (1 - \bar{\alpha}_t) \mathbf{I}\right)$$



**Abbildung 3.6.:** Rückwärts *Diffusions*-prozess: Schrittweise Entrauschung der Daten. [70]

Während des Rückwärtsprozesses (siehe Abbildung 3.6) wird versucht, das eingefügte *Rauschen*  $\epsilon \sim \mathcal{N}(0, 1)$  schrittweise zu entfernen [70]. Dies lässt den umgekehrten Prozess so erscheinen, als würde er aus dem *Rauschen* neue Daten generieren [70]. Bei einem niedrigen Wert von  $\beta$  stimmt die Rauschverteilung des Rückwärtsschritts  $q(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$  mit dem Vorwärtsprozess überein [102]. Der erlernte Rückwärtsprozess

### 3. Grundlagen

---

$p_\theta(\mathbf{x}_{t-1} | \mathbf{x}_t)$  kann dementsprechend approximiert werden [41, 70, 76]. Das Training fokussiert sich darauf, kleinere Abweichungen zu schätzen, anstatt den gesamten Ablauf in einem einzigen Schritt durch eine Funktion zu repräsentieren [102].

$$(3.4) \quad p_\theta(\mathbf{x}_{t-1} | \mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \mu_\theta(\mathbf{x}_t, t), \Sigma_\theta(\mathbf{x}_t, t))$$

$$(3.5) \quad p_\theta(\mathbf{x}_{0:T}) = p(\mathbf{x}_T) \prod_{t=1}^T p_\theta(\mathbf{x}_{t-1} | \mathbf{x}_t)$$

Das Erlernen des Rückwärtsprozesses wird durch ein neuronales Netzwerk durchgeführt. In jedem Schritt können das Netzwerk den Erwartungswert  $\mu_\theta$ , das ursprüngliche Bild  $\mathbf{x}_0$  oder das hinzugefügte *Rauschen*  $\epsilon$  vorhersagen [41, 76]. Untersuchungen zeigten, dass die Vorhersagen des Bildes weniger präzise waren. Daher wurde die Entscheidung getroffen, das hinzugefügte *Rauschen*  $\epsilon$  unter Verwendung der vereinfachten Verlustfunktion 3.6 vorherzusagen [41].

$$(3.6) \quad L_{\text{simple}} = E_{t, \mathbf{x}_0, \epsilon} [\|\epsilon - \epsilon_\theta(\mathbf{x}_t, t)\|^2]$$

Der Erwartungswert  $\mu_\theta$  kann aus dem vorhergesagten *Rauschen*  $\epsilon$  mithilfe der nachstehenden Gleichung abgeleitet werden [41]:

$$(3.7) \quad \mu_\theta(\mathbf{x}_t, t) = \frac{1}{\sqrt{\alpha_t}} \left( \mathbf{x}_t - \frac{\beta_t}{\sqrt{1 - \bar{\alpha}_t}} \epsilon_\theta(\mathbf{x}_t, t) \right)$$

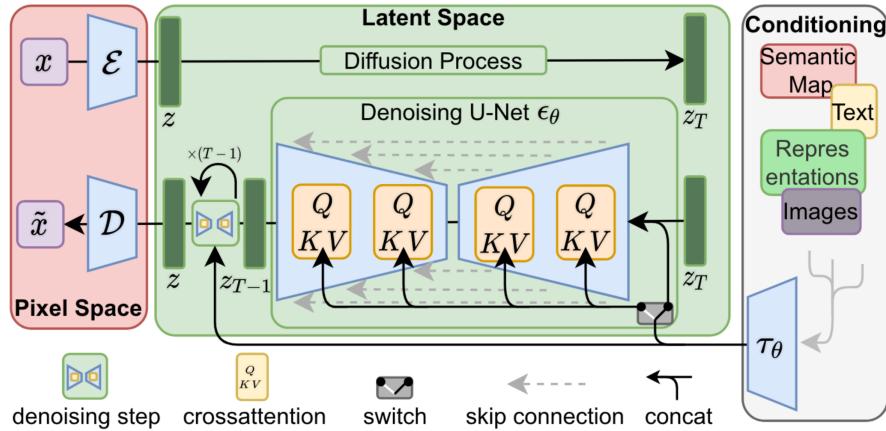
Die in [41] präsentierte Architektur des neuronalen Netzwerks basiert auf einem *U-Net* [95], welches auf einem *Wide ResNet* [117] aufbaut [41].

In [76] wurden signifikante Verbesserungen im Vergleich zu den Arbeiten von [102] und [41] vorgenommen. Eine dieser Neuerungen war das Lernen der Varianz anstelle ihrer festen Einstellung. Zudem wurde der *schedule* Parameter optimiert. Statt einer konstanten Einstellung, die dazu führte, dass die Datenpunkte am Ende übermäßig verrauscht waren und die anfänglichen Schritte zu viel Information verloren, wurde ein *cosinus schedule* eingeführt. [76]

#### 3.2.2. Latente Diffusion

In [94] wurde das *Latente Diffusions-modell (LDM)* vorgestellt, welches den *Diffusions*-prozess [24, 41, 76, 102] erweitert, um die Generierung von Bildern basierend auf einer Eingabe zu ermöglichen. Anders als bei früheren *Diffusions*-prozessen fokussiert sich dieser Ansatz nicht auf die Pixelwerte der Bilder, sondern auf ihre *latente* Darstellung. Dies reduziert den Rechenaufwand erheblich und ermöglicht sowohl das Training als auch die *Inferenz* auf eingeschränkter Hardware. Die in Abbildung 3.7 dargestellte Architektur besteht aus drei Hauptteilen: Ein *Variational Autoencoder*, der die Daten im *Latentenraum* kodiert und dekodiert, der *Diffusions*-teil und ein zusätzliches Modul, das den *Diffusions*-teil mithilfe von Text, Bildern oder anderen Eingaben konditioniert. [94]

Durch das vorausgehende Training eines *Variational Autoencoders* [54] gemäß [29] wird der *latente* Raum für den *Diffusions*-prozess erzeugt. Dabei transformiert der resultierende *Encoder*  $\mathcal{E}$  ein RGB-Bild  $x \in \mathbb{R}^{H \times W \times 3}$  in seine *latente* Darstellung  $z \in \mathbb{R}^{h \times w \times c}$ , dargestellt als  $z = \mathcal{E}(x)$ . Nach dem *Rückwärtsdiffusions*-prozess generiert ein *Decoder*  $\mathcal{D}$  das Bild zurück, wobei  $\tilde{x} = \mathcal{D}(z) = \mathcal{D}(\mathcal{E}(x))$ . [94]



**Abbildung 3.7.:** LDM-Architektur: *Diffusion* mittels *U-Net* im durch den *Encoder*  $\mathcal{E}$  erzeugten *latenten* Raum. Rekonstruktion in die ursprüngliche Darstellung erfolgt durch den *Decoder*  $\mathcal{D}$ . [94]

Dank der niedrigen Dimensionalität und Kompression des *latenten* Raumes eignet sich der *Diffusionsprozess* besser für *likelihood*-basierte generative Modelle als die Verwendung der ursprünglichen Pixelwerte. Dies liegt daran, dass semantisch wichtige Informationen stärker hervorgehoben werden und der Rechenaufwand effizienter gestaltet wird. Das zugrundeliegende *U-Net* [95] verwendet hauptsächlich 2D-Konvolutionsschichten, um die Bildverarbeitung zu optimieren. [94]

Um eine von  $y$  abhängige Generierung zu ermöglichen, muss das Netzwerk  $\epsilon_\theta(z_t, t, y)$  konditioniert werden. Dazu wird das verwendete *U-Net* durch *Cross Attention* [110] erweitert. Ein spezifischer *Encoder*  $\tau_\theta$  passt sich an verschiedene Eingabemodalitäten an und erstellt eine Repräsentation  $T_\theta(y) \in \mathbb{R}^{M \times d_\tau}$ . Diese Repräsentation wird mittels *Cross Attention* auf die Schichten des *U-Net* angewendet. Die dabei verwendete Aufmerksamkeitsfunktion ist durch  $\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right)$  definiert, wobei  $Q = W_Q^{(i)} \cdot \varphi_i(z_t), K = W_K^{(i)} \cdot \tau_\theta(y), V = W_V^{(i)} \cdot \tau_\theta(y)$  entsprechend gegeben sind. Die zugehörige Verlustfunktion wird im Folgenden beschrieben. [94]

$$(3.8) \quad L_{LDM} := \mathbb{E}_{\mathcal{E}(x), y, \epsilon \sim \mathcal{N}(0, 1), t} [\|\epsilon - \epsilon_\theta(z_t, t, \tau_\theta(y))\|_2^2]$$

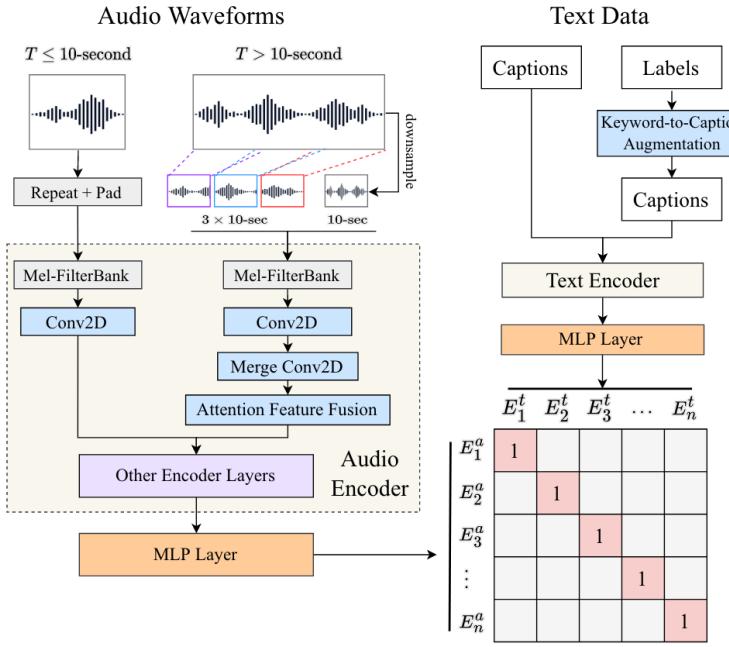
### 3.2.3. Clap

*Large-Scale Contrastive Language-Audio Pretraining (Clap)* [113] erschafft, mittels *Contrastive Learning* eine Vorverarbeitung auf Sprach-Audio-Daten durchzuführen. Ziel ist es, eine *latente* Repräsentation für Audio zu erstellen. Inspiriert wurde dieses Modell von *Contrastive Language-Image Pretraining (CLIP)* [89], welches den Zusammenhang zwischen Bild- und Textdaten in ähnlicher Weise lernt. Analog zum bildlichen Kontext weisen Audio und Text überlappende Informationen auf. [113]

Das *Contrastive Learning*-Modell wurde mit dem spezifisch veröffentlichten Datensatz *LAION-Audio-630K* trainiert. Dieser umfasst 633,526 Paare aus Text und Audio ( $X_i^a, X_i^t$ ), die aus verschiedenen Online-Quellen stammen. Dazu gehören menschliche Stimmen, Naturklänge und Audiomaterial. Zusätzlich wurden die Datensätze *AudioCaps+Clotho* [53] [26] und *AudioSet* [31] verwendet. Alle Audiodateien wurden in ein *Mono*-Signal umgewandelt und haben eine Abtastrate von 48kHz. [113]

### 3. Grundlagen

---



**Abbildung 3.8.: CLAP-Architektur für latente Repräsentation von Text-Audio-Paaren:** Der Audio-Encoder verarbeitet Signale variabler Länge und adaptiert sie für das MLP. Der Text-Encoder behandelt textuelle Eingaben; bei Verwendung von Labels erfolgt zuerst eine Umschreibung. [113]

Die Modellarchitektur, dargestellt in Abbildung 3.8, generiert Embeddings  $E_i^a$  und  $E_i^t$  für die Audio- $X_i^a$  und Texteingabe  $X_i^t$ . Hierfür wird zunächst ein Encoder  $f(\cdot)$  eingesetzt, dessen Ergebnis anschließend durch ein zweischichtiges *Multi-Layer-Perceptron (MLP)* mit *ReLU* [2] als Aktivierungsfunktion weiterverarbeitet wird. [113]

$$(3.9) \quad E_i^a = \text{MLP}_{\text{audio}}(f_{\text{audio}}(X_i^a))$$

$$(3.10) \quad E_i^t = \text{MLP}_{\text{text}}(f_{\text{text}}(X_i^t))$$

Um die Rechenzeit bei längeren Audiosignalen zu minimieren, wurde ein Ansatz entwickelt, der es ermöglicht, Audioeingaben unterschiedlicher Länge in konstanter Zeit zu verarbeiten. Dabei werden sowohl globale als auch lokale Signalinformationen berücksichtigt. Bei Signalen unter 10 Sekunden wird das Signal dreifach wiederholt und mit Nullen ergänzt, bis es 10 Sekunden erreicht. Für Signale über 10 Sekunden werden vier Eingaben erstellt: Einmal wird das gesamte Signal auf 10 Sekunden *downsampled* und zusätzlich werden drei zufällige 10-Sekunden-Abschnitte aus verschiedenen Signalbereichen entnommen. Der Audio-Encoder kombiniert anschließend diese globalen und lokalen Daten, um die relevanten Informationen zu extrahieren. [113]

Einige der genutzten Datensätze enthalten für Audiosignale lediglich Keywords oder Tags statt vollständiger Beschreibungen. Um diesem Mangel zu begegnen, wurde das *T5* [91] *Language*-Modell verwendet, um aus diesen Stichworten und Tags umfassende Beschreibungen zu erstellen. Zusätzlich wurden die generierten Beschreibungen von Voreingenommenheiten befreit, wie zum Beispiel durch das Entfernen geschlechtsspezifischer Verzerrungen (eng. *gender de-biasing*). [113]

Die gleiche Verlustfunktion mit einem Temperaturparameter  $\tau$ , wie in *Clip* [89], wurde in den *MLPs* verwendet [113].

$$(3.11) \quad L = \frac{1}{2N} \sum_{i=1}^N \left( \log \frac{\exp(E_i^a \cdot E_i^t / \tau)}{\sum_{j=1}^N \exp(E_i^a \cdot E_j^t / \tau)} + \log \frac{\exp(E_i^t \cdot E_i^a / \tau)}{\sum_{j=1}^N \exp(E_i^t \cdot E_j^a / \tau)} \right)$$

Als Audio-Encoder wurden die Modelle *PANN* [58], basierend auf einem *Convolutional Neural Network* (*CNN*), und *HTSAT* [10], basierend auf einem *Transformer*-Modell, evaluiert. Für den Text-Encoder wurden der Text-Encoder von *Clip* [89], *Bert* [23], und *RoBERTa* [68] ebenfalls ausgewertet. [113]

Das trainierte Modell eignet sich sowohl für die Audio-Klassifikation als auch zur Bestimmung eines zugehörigen Audiosignals für einen gegebenen Text ( $T \rightarrow A$ ) [113]. Insbesondere für die Klangsynthese ist die Konversion von Text zu Audio von großer Relevanz. Die Untersuchungsergebnisse, die durch die Kombination verschiedener Encoder aus Tabelle 3.1 erzielt wurden, verdeutlichen, dass *HTSAT* [10] als Audio-Encoder in Kombination mit *Bert* [23] oder *RoBERTa* [68] als Text-Encoder, abhängig vom verwendeten Datensatz, die optimalsten Ergebnisse erbrachte. [113]

Model	AudioCaps (mAP@10)		Clotho (mAP@ 10)	
	A → T	T → A	A → T	T → A
PANN+CLIP Trans.	4.7	11.7	1.9	4.4
PANN+BERT	34.3	44.3	10.8	17.7
PANN+RoBERTa	37.5	45.3	11.3	18.4
HTSAT+CLIP Trans.	2.4	6.0	1.1	3.2
HTSAT+BERT	43.7	49.2	<b>13.8</b>	<b>20.8</b>
HTSAT+RoBERTa	<b>45.7</b>	<b>51.3</b>	<b>13.8</b>	20.4

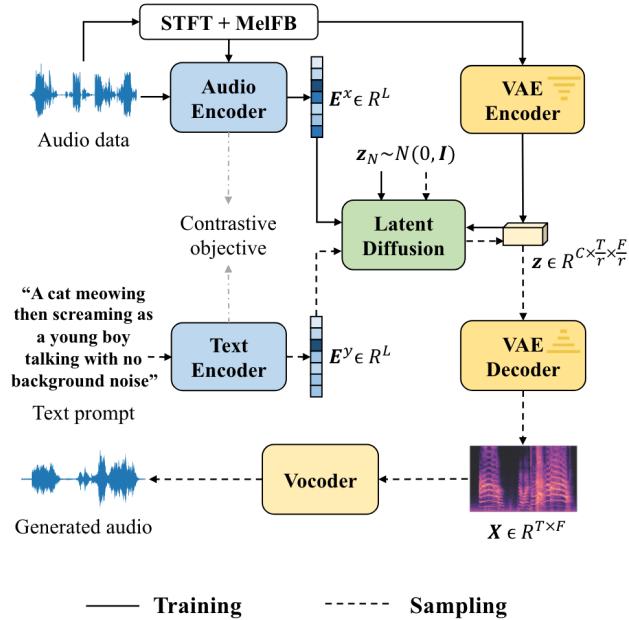
**Tabelle 3.1.:** Evaluation der Encoder [113]

### 3.2.4. AudioLDM

*AudioLDM* [63] wurde entwickelt, um sowohl die Synthese von Text zu Audio als auch die textgesteuerte Audio-Manipulation mittels *Latenter Diffusion* [94] durchzuführen. Es bietet den Vorteil qualitativ hochwertiger Ergebnisse bei minimalem Rechenaufwand. Während *Diffsound* [115] Audio-Text-Paare für Trainingszwecke nutzt, hat *AudioLDM* nachgewiesen, dass durch die Verwendung einer *CLAP*-Repräsentation [113] hochwertige Resultate erlangt werden können. [63].

Die Entscheidung, Natürliche Sprache anstelle von Labels für Eingabe und Konditionierung zu nutzen, wurde getroffen, um akustische Merkmale in einer flexibleren, deskriptiveren und präziseren Weise darstellen zu können. Frühere Ansätze im Bereich Text-zu-Audio (TTA) waren durch das Fehlen großer Mengen an qualitativ hochwertigen Audio-Text-Daten limitiert [67]. Obwohl Methoden der Textvorverarbeitung [31, 115] versuchten, dieses Problem zu adressieren, konnten sie die komplexen Beziehungen zwischen Klangereignissen nicht vollständig erfassen, was zu einer Beeinträchtigung der Generierungsleistung führte. Das Modell *AudioLDM* begegnete dieser Herausforderung, indem es im Training ausschließlich auf Audiodaten zurückgriff und somit überzeugendere Ergebnisse im Vergleich zu Ansätzen mit gepaarten Audio-Text-Daten erzielte [63].

### 3. Grundlagen



**Abbildung 3.9.:** Architektur von AudioLDM für Text-zu-Audio-Generierung: Die VAE-Encoder und Decoder bilden auf Grundlage von Spektrogrammen die latente Repräsentation, die mittels Diffusion modifiziert wird. Text- und Audio-Encoder dienen der Konditionierung, während der Vocoder ein Audiosignal aus dem Spektrogramm generiert. [63]

Die in Abbildung 3.9 dargestellte Architektur integriert mehrere Komponenten: Einen VAE [54], welcher Mel-Spektrogramme in einen *latenten* Raum transformiert; einen nach dem CLAP-Prinzip [113] entworfenen Audio- und Text-Encoder zur Erzeugung von Embeddings, die zur Konditionierung des LDMs dienen; einen LDM, der die inhärente Verteilung modelliert; sowie einen Vocoder, der das generierte Spektrogramm in ein hörbares Audiosignal konvertiert. [63]

Die CLAP-Module erzeugen Embeddings sowohl für Text, repräsentiert durch  $E^y \in \mathbb{R}^L$ , als auch für Audio, dargestellt durch  $E^x \in \mathbb{R}^L$ . In diesem Kontext steht  $x$  für eine Audioprobe und  $y$  für eine Textbeschreibung. Die Funktionen  $f_{\text{text}}(\cdot)$  und  $f_{\text{audio}}(\cdot)$  agieren hierbei als Encoder. Als Audio-Encoder kam HTSAT [10] zum Einsatz, während für den Text-Encoder RoBERTa [68] verwendet wurde. Während des Trainingsprozesses des LDMs fungierte  $E^x$  des jeweiligen Trainingsdatenpunkts als Konditionierungselement, wohingegen  $E^y$  zur Generierung des gewünschten Signals herangezogen wurde [63].

Der für den LDM verwendete Latentre Raum, beschrieben durch  $z \in \mathbb{R}^{C \times \frac{T}{r} \times \frac{F}{r}}$ , wurde mittels eines VAE auf der Grundlage von Spektrogrammen trainiert. In dieser Darstellung bezeichnet  $r$  das Kompressionsniveau. Die Variablen  $T$  und  $F$  repräsentieren die Zeit- und Frequenzdimensionen, während  $C$  die Anzahl der Kanäle angibt. Ein gegebenes Audiosignal wird durch die STFT-Methode (siehe Abschnitt 3.1.1) in ein Spektrogramm  $X \in \mathbb{R}^{T \times F}$  überführt. Im Generierungsprozess wird aus der *latenten* Variable  $\hat{z}_o$  ein Spektrogramm  $\hat{X}$  erzeugt. Dieses generierte Spektrogramm wird anschließend durch den Vocoder, welcher auf HiFi-GAN [57] basiert, in ein Audiosignal  $\hat{x}$  konvertiert. [63]

Das konzeptuelle Modell LDM verfolgt die Absicht, die fundamentale Verteilung  $q(z_0 | E^y)$  durch die Annäherung an  $p_\theta(z_0 | E^y)$  zu repräsentieren. In diesem Kontext wurden sowohl die Verlustfunktion als auch der Rückwärtsprozess entsprechend definiert: [63]

$$(3.12) \quad L_n(\theta) = \mathbb{E}_{z_0, \epsilon, n} \|\epsilon - \epsilon_\theta(z_n, n, E^x)\|_2^2$$

$$(3.13) \quad p_\theta(z_{n-1} | z_n, E^y) = \mathcal{N}(z_{n-1}; \mu_\theta(z_n, n, E^y), \sigma_n^2 I)$$

$$(3.14) \quad p_\theta(z_{0:N} | E^y) = p(z_N) \prod_{n=0}^{N-1} p_\theta(z_{n-1} | z_n, E^y)$$

Der Erwartungswert wird wie folgt parametrisiert und ergibt sich aus dem im Rückwärtsverfahren ermittelten Rauschen:  $\epsilon_\theta(z_n, n, E^y)$  [63].

$$(3.15) \quad \mu_\theta(z_n, n, E^y) = \frac{1}{\sqrt{\alpha_n}} \left( z_n - \frac{\beta_n}{\sqrt{1 - \bar{\alpha}_n}} \epsilon_\theta(z_n, n, E^y) \right)$$

Ferner wurde nachgewiesen, dass der *Cross-Attention*-Mechanismus gemäß [94] entbehrlich ist [63].

### 3.2.5. AudioLDM2

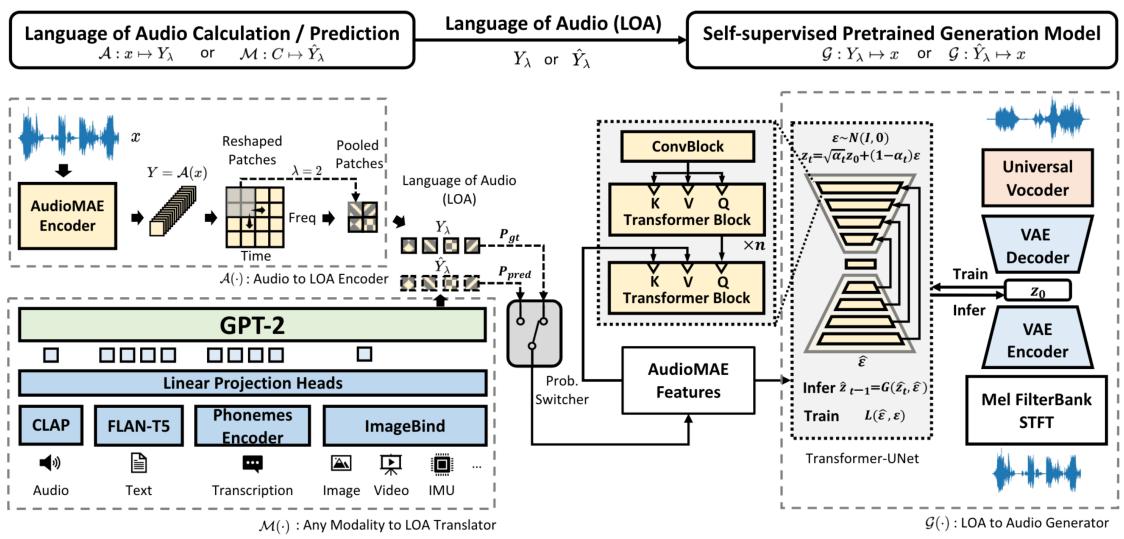
*AudioLDM2* [64] baut auf den erarbeiteten Konzepten von *AudioLDM* [63] auf und verspricht durch eine neuartige Architektur (siehe Abbildung 3.10) in Verbindung mit innovativen Konzepten verbesserte Resultate in der Audiogenerierung mittels neuronaler Netze. Die treibende Kraft hinter dieser Entwicklung war die Erkenntnis, dass im Bereich der Audiosynthese zwar verschiedene Modelle für unterschiedliche Anwendungen wie Spracherzeugung, Musikgenerierung und Klangsynthese existieren, diese jedoch häufig so spezifisch und beschränkt für eine bestimmte Aufgabe oder Domäne entwickelt werden, dass ihre Anwendbarkeit in einem übergeordneten Kontext limitiert ist. Das vorgestellte Modell verfolgt das Ziel, diese diversen spezifischen Herausforderungen in einem einzigen Modell zu integrieren. Zu diesem Zweck wurde eine Audio-Repräsentation namens *language of audio* (*LOA*) konzipiert, die sich effizient über verschiedene Domänen und Modalitäten, wie Text, Audio und verschiedene Medien, generalisieren lässt. Auf Basis dieser Repräsentation wurde ein *Latente-Diffusion*-Modell trainiert, um die Audiosynthese zu realisieren. [64]

Die Synthese eines Audiosignals  $x \in \mathbb{R}^{L_s}$ , wobei  $L_s$  die Dauer des Signals in Sekunden angibt, kann als Funktion  $\mathcal{H} : C \mapsto x$  im Hypothesenraum  $\mathcal{H}$  dargestellt werden. In dieser Funktion stellt  $C$  die Eingabe dar, auf deren Basis das Signal generiert wird. Diese Eingabe kann sowohl Text als auch diverse Medienformate umfassen. Das Erlernen dieser Funktion stellt aufgrund der signifikanten Unterschiede in den Verteilungen von  $C$  und  $x$  eine besondere Herausforderung dar. Ein vorgeschlagener Lösungsweg besteht in einer Abstraktion durch *LOA*. Die Darstellung eines Audiosignals im *LOA*-Format wird als  $Y = \mathcal{A}(x)$  definiert, wobei  $\mathcal{A}$  einen *Audio-zu-LOA-Encoder* darstellt.  $\mathcal{A}$  kann entweder durch vordefinierte Regeln definiert oder mittels *self-supervised learning* [104] trainiert werden. Um verschiedenste Eingabeformate zu unterstützen und in *LOA* zu konvertieren, wurde der Übersetzer *any modality to LOA translator*  $\mathcal{M}$  als  $\hat{Y} = \mathcal{M}(C)$  definiert. Infolgedessen kann die Signalgenerierungsfunktion entsprechend formuliert werden. [64]

$$(3.16) \quad \mathcal{H}_0 = \mathcal{G} \circ \mathcal{M} : C \mapsto \hat{Y} \mapsto x$$

Hierbei extrahiert  $\mathcal{G}$  ein Audiosignal aus der *LOA*-Repräsentation. Falls  $\mathcal{M}$  durch  $\mathcal{A}$  ersetzt wird (wie in Gleichung 3.17 gezeigt), tritt nur  $x$  als Trainingsdaten für  $\mathcal{G}$  auf. Dementsprechend kann  $\mathcal{G}$  *self-supervised* ohne den Einsatz von Labels trainiert werden, wodurch das Problem einer geringen Anzahl an gelabelten Audiopaaren umgangen wird. [64]

### 3. Grundlagen



**Abbildung 3.10.:** Architektur von AudioLDM2: Das *Diffusions*-netz integriert *U-Net* und *Transformer*-Konzepte und operiert auf der *latenten* Repräsentation von *Spektrogrammen*. Die Konditionierung basiert auf der *LOA*-Repräsentation, abgeleitet von *AudioMAE*-Informationen. Diese kann entweder durch den *Audio-zu-LOA* Encoder oder den *Any Modality-zu-LOA* Konverter gewonnen werden. [64]

$$(3.17) \quad \mathcal{H}_0 = \mathcal{G} \circ \mathcal{A} : C \mapsto Y \mapsto x$$

Um eine vielfältige Audio-Repräsentation  $Y$  für Sprache, Musik und Soundeffekte zu erstellen, sollte diese in der Lage sein, sowohl semantische als auch akustische Informationen zu erfassen. Zur Erfüllung dieser Kriterien wurde für den *Audio zu LOA-Encoder* das mittels *self-supervised* Verfahren vortrainierte Modell *AudioMAE* [44] herangezogen. Das Modell trainiert eine Repräsentation von ungelabelten Audiosignalen mithilfe eines *Encoder-Decoder*-Ansatzes. Hierbei werden dem *Encoder Mel-Spektrogramme* mit maskierten Bereichen übergeben, welche der *Decoder* anschließend zu rekonstruieren versucht. Nachdem die Repräsentation  $Y$  generiert wurde, kann diese durch den Einsatz von *average-max pooling* [66] weiter zu  $Y_\lambda$  verfeinert werden. [64]

Das Erlernen von  $\mathcal{G}$  wurde in Anlehnung an *AudioLDM* [63] mittels *Latenter Diffusion* [94] durchgeführt (siehe Abschnitt 3.2.4). Der *Diffusions*-vorgang ereignet sich in einem *latenten* Raum, der mittels eines *VAE* [54] erlernt wurde. Die *Diffusion* findet nicht direkt auf  $Y$  statt, da sich *AudioMAE* nicht in erster Linie auf den Erhalt der Qualität während des Rekonstruktionsvorgangs konzentriert. Ein *VAE* hingegen kennzeichnet sich durch eine überlegene Rekonstruktionsfähigkeit und ein intensiveres Kompressionsverhältnis. Während *AudioMAE* primär semantische Aspekte hervorhebt, legt der *VAE* sein Augenmerk verstärkt auf akustische Details. [64]

Die durch den *VAE* generierte *latente* Repräsentation  $z$  wird mittels eines *Encoders*, der *Downsampling* verwendet, und eines *Decoders*, der *Upsampling* einsetzt, erlernt. Ein gegebenes Spektrogramm  $X$  wird in  $z$  umgewandelt und gemäß  $\mathcal{V} : X \mapsto z \mapsto \hat{X}$  rekonstruiert. Aus  $\hat{X}$  kann mithilfe eines vortrainierten *HiFiGAN* [57] ein Audiosignal generieren. Die Optimierung des *VAE* basiert auf den Differenzen zwischen  $X$  und  $\hat{X}$ . [64]

Der Vorwärtsprozess des *LDM* im *latenten* Raum mit dem Hyperparameter  $\beta_t \in [0, 1]$  ist wie folgt definiert [64]:

$$(3.18) \quad q(z_t | z_{t-1}) = \sqrt{1 - \beta_t} z_{t-1} + \sqrt{\beta_t} \epsilon_t, t \in 2, \dots, T$$

Der Rückwärtsprozess hingegen ist definiert als [64]:

$$(3.19) \quad q(z_t | z_0) = \sqrt{\alpha_t} z_0 + \sqrt{1 - \alpha_t} \epsilon_t$$

Die zugehörige Verlustfunktion lautet [64]:

$$(3.20) \quad \operatorname{argmin}_{\phi} \left[ \mathbb{E}_{z_0, Y, t \sim \{1, \dots, T\}} \left\| \mathcal{G} \left( \sqrt{\alpha_t} z_0 + \sqrt{1 - \alpha_t} \epsilon_t, t, Y; \phi \right) - \epsilon_t \right\| \right]$$

Für  $\mathcal{G}$  kam ein *Transformer-UNet* (*T-UNET*) zum Einsatz. Dieses Netzwerk vereint einen *Encoder*, der *Downsampling* nutzt, mit einem *Decoder*, der *Upsampling* einsetzt. Im Anschluss an einen *Konvolutions*-Block werden zudem *Transformer*-Blöcke eingebunden. [64]

Das Modell  $\mathcal{M}$  ist von Bedeutung, da während der *Inferenz*  $\mathcal{A}$  nicht verfügbar ist. Zur Generierung von  $\hat{Y}$  ist ein alternatives Modell  $\mathcal{M}_\theta : C \rightarrow \hat{Y}$  erforderlich. Dabei stellt  $\theta$  die anlernbaren Parameter dar. Um der Ausgabe des *AudioMAE* optimal zu entsprechen, wurde dieses Problem als Aufgabe der Sprachmodellierung definiert. Als Basis für dieses Modell dient der *Generative Pre-trained Transformer 2 (GPT-2)* [4]. Dieser wurde mittels eines *unsupervised* Verfahrens auf einem umfangreichen Textdatensatz trainiert. Sein Hauptziel ist die Bewältigung von Aufgaben im Kontext des *Natural Language Processing (NLP)*, wie Textvervollständigung, Frage-Antwort-Systeme oder Sprachübersetzung. Das Modell *GPT-2* wurde spezifisch für den Einsatz in *AudioLDM2* mittels *teacher forcing* [56] verfeinert. Gegeben ein Datensatz  $C$ , wird das Modell durch die Maximierung der Likelihood optimiert.  $C$  kann diverse Daten repräsentieren, darunter Audio, Text, Phoneme oder visuelle Daten. Zur Extraktion essentieller Merkmale kommt ein *Mischung von Experten*-Ansatz (engl. *mixture of experts*) [71] zum Einsatz, wodurch ein vielfältiges Informationsspektrum zugänglich gemacht wird. Ein *Linearer Adapter* (engl. *linear adaptor*) dient dazu, sämtliche Merkmale auf die einheitliche Dimension  $D$  zu bringen. Verschiedenste Systeme können zur Extraktion dieser Merkmale genutzt werden. [64]

Zur Nutzung von Text als Kondition  $C$  kamen sowohl *CLAP* [113] als auch *FLAN-T5* [13] zum Einsatz. Da *CLAP* Schwierigkeiten bei der angemessenen Verarbeitung temporaler und semantischer Informationen aufwies, fungierte *FLAN-T5* zusätzlich als zusätzlicher *Encoder*. Des Weiteren diente *CLAP* der Generierung von Paraphrasen für Audiosignale, welche ursprünglich keine entsprechenden Umschreibungen besaßen, wie es bei Text-zu-Sprach-Aufgaben vorkommt. Ein weiterer in der Arbeit implementierter *Encoder* ist der *Phoneme Encoder*. Dieser wurde speziell konzipiert, um Informationen über Phoneme, die kleinsten sprachlichen Einheiten, zu kodieren. Für die Kodierung visueller Daten wurde *ImageBind* [33] eingesetzt. [64]

Während des Finetunings entschied ein probabilistischer Mechanismus über das Konditionierungssignal. In 25% der Fälle dienten die von *AudioMAE* extrahierten Informationen als *Ground Truth*. Demgegenüber wurden in den verbleibenden 75% der Szenarien die von *GPT* generierten Daten herangezogen. [64]



## 4. Methoden

### 4.1. Implementierung des Neuronalen Synthesizers

#### 4.1.1. Audioprogrammierung

Die Geschwindigkeit der Datenverarbeitung stellt ein maßgebliches Kriterium bei der Wahl der Programmiersprache für die Entwicklung und Implementierung von digitalen Audioprozessen dar. Für Echtzeitanwendungen gilt insbesondere, dass der Code so effizient wie möglich gestaltet sein sollte, um jegliche Latenz zu minimieren. Unter Berücksichtigung dieser Prämissen fällt die Wahl häufig auf eine Realisierung in *C/C++*. [9, 25]

*C++* wurde als Erweiterung der Programmiersprache *C* entwickelt und behält dennoch *C* als eine seiner Untermengen bei. Es baut auf den fundamentalen Prinzipien von *C* auf, insbesondere auf der hardwarenahen Programmierung und der Fähigkeit, auf den meisten Systemen zu operieren. Ergänzend erweitert *C++* das Repertoire um Facetten der Datenabstraktion sowie um objektorientierte und generische Programmieransätze. [103]

#### 4.1.2. JUCE Framework

„*JUCE* ist das am häufigsten verwendete Framework für die Entwicklung von Audioanwendungen und Audio-Plugins. Es handelt sich dabei um eine Open-Source-C++-Codebasis, die zur Erstellung eigenständiger Software auf Windows, macOS, Linux, iOS und Android sowie VST-, VST3-, AU-, AUv3-, AAX- und LV2-Plugins verwendet werden kann.“<sup>1</sup> [51]

Es bietet eine Abstraktion für die Verarbeitung von Audiosamples und *MIDI* von den nativen Audiogeräten auf jeder Plattform oder einer *Host-DAW*. Die von *JUCE* angebotene Bibliothek für *digitale Signalverarbeitungs (DSP)*-Bausteine ermöglicht eine rasche Prototypisierung und Implementierung verschiedener Audioeffekte, Filter, Instrumente und Generatoren. [51] Zudem bietet *JUCE* eine Vielzahl von Klassen, die gängige Herausforderungen in der Entwicklung von Audioprojekten adressieren. Dies schließt die Verwaltung von Grafiken, Sound, Benutzerinteraktion und Netzwerkkommunikation mit ein. [93]

In dieser Arbeit wird das *JUCE*-Framework mittels *CMake*, „eine Open-Source-, plattformübergreifende Werkzeugfamilie, die zur Erstellung, zum Testen und zum Verpacken von Software entwickelt wurde“<sup>2</sup> [14] eingesetzt, um die durch das *Diffusions*-netz erstellten Klänge spielbar und manipulierbar zu gestalten.

Das *Juce*-Modul *PluginGuiMagic* [111] erleichtert und beschleunigt die Gestaltung von Benutzeroberflächen für die zu entwickelnde Anwendung.

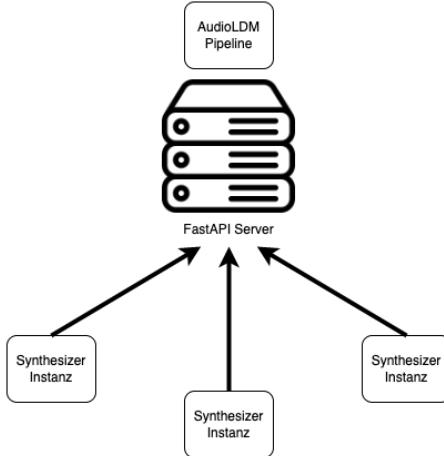
---

<sup>1</sup>*JUCE* is the most widely used framework for audio application and plug-in development. It is an open source C++ codebase that can be used to create standalone software on Windows, macOS, Linux, iOS and Android, as well VST, VST3, AU, AUv3, AAX and LV2 plug-ins.

<sup>2</sup>*CMake* is an open source, cross-platform family of tools designed to build, test, and package software.

## 4. Methoden

---



**Abbildung 4.1.:** Anbindung an den *AudioLDM*-Server

### 4.1.3. AudioLDM-Anbindung

Um ein von den *Diffusions*-modellen *AudioLDM* und *AudioLDM2* erzeugtes Sample, mithilfe des im *JUCE*-Framework entwickelten digitalen Instruments, wiederzugeben, muss *Inferenz* der Modelle aus dem C++-Quellcode des Instruments möglich sein.

Um eine Binäre Repräsentation des Modells zu generieren wäre eine *ONNX*-Modell [78] eine geeignete Methode. Mittels der *ONNX-Runtime* [79] ließe sich diese in C++ integrieren und abrufen, um *Inferenz*-operationen auszuführen und so die Soundgenerierung zu realisieren. Dieser Ansatz könnte insbesondere das Kompilieren und die Verbreitung des finalisierten Instruments samt des trainierten Modells fördern. Eine alternative Herangehensweise wäre, das Modell in ein *Torchscript*-Modell [108] zu konvertieren und dieses dann in C++ zu laden. [77]

Bislang ließ sich weder ein *ONNX*- noch ein *Torchscript*-Modell für das *AudioLDM*-/*AudioLDM2*-Modell erstellen. Dies liegt an der komplexen Interaktion verschiedener eingesetzter Modelle und Module, von denen einige gegenwärtig weder von *ONNX* noch von *Torchscript* unterstützt werden. Daher wurde für die Realisierung dieses Projekts die Erstellung einer *API* (*Application Programming Interface*) bevorzugt. Durch definierte Endpunkte ermöglicht diese *API* die Generierung einer lokalen Instanz des Modells, welche zur Erzeugung benötigter Samples dient. Als Basis für diese Implementierung dient das *FastAPI*-Framework [30]. Auf dem *Python*-Server wird eine Instanz des *AudioLDM*-/*AudioLDM2*-Modells betrieben, welche mithilfe der *Diffusers*-Bibliothek [85] von *Huggingface* [45] als *Pipeline* [46, 47] genutzt wird. Die Anwendung dieser Pipeline erlaubt *Inferenzen* mit einer gewissen Abstraktionsebene und ihre Nutzung auf unterschiedlichen Systemen. Zuvor waren die veröffentlichten Modelle etwa nicht auf Rechnern ohne GPU, wie den M-Modellen von Apple, lauffähig. Doch durch die Konfigurierbarkeit des in der Pipeline genutzten Prozessors ließ sich dieses Hindernis überwinden. Zudem verhindert der Einsatz einer *API*, dass bei mehreren *Synthesizer*-Instanzen jeweils ein neues Modell initialisiert werden muss, was eine parallele Nutzung mehrerer Modelle unnötig macht.

Die implementierten Endpunkte beinhalten einen zur Initialisierung eines Modells. Für diese Initialisierung sind Angaben zum *torch*-Gerät, welches den auszuführenden Prozessor bestimmt, und zum *Modell*, welches das zu verwendende vortrainierte Modell festlegt, erforderlich. Der Endpunkt für die Klanggenerierung benötigt Eingaben wie *Prompt*, *negativer Prompt*, *Audiolänge*, *Anzahl der Inferenzschritte* und *Leitskala* und optional einen *Seed*. Als Rückgabe erfolgt ein in *Base64* kodiertes Audiosignal.

#### 4.1.4. Sampler

Für die Implementierung des *Samplers* wurde der von *Juce* bereitgestellten *Sampler* [100] verwendet. Darüber hinaus wurde die Funktionalität erweitert, um gezielte Tonverschiebungen durchzuführen und den *Pitch-Bender* eines Keyboards zu verwenden.

## 4.2. Verteilung und Bündelung

Für *macOS* konnten eine *.app*-Applikationsdatei, eine *VST3*- sowie eine *AU*-Datei generiert werden. Um die Installation der Dateien für Nutzer\*innen zu vereinfachen, wurden diese in eine *pkg*-Installationsdatei gebündelt.

Der *FastAPI*-Server wurde mittels *pyinstaller* [87] zu einer Applikation kompiliert, die Nutzer\*innen unkompliziert öffnen kann. *pyinstaller* integriert alle erforderlichen Abhängigkeiten in einem Paket, sodass Nutzer\*innen die Applikation ausführen kann, ohne einen *Python*-Interpreter oder sonstige Module installieren zu müssen [87]. Da jedoch alle benötigten Module inkludiert werden, resultiert beispielsweise für den *Silicon*-Server ein ursprünglich 5 KB großes Skript in einer Dateigröße von 450 MB. Die Vergrößerung des Speicheraufwandes ist somit erheblich, jedoch unabdingbar, um den Server und damit ein benutzbare Instrument zu verteilen und zu veröffentlichen. Falls die generierten Server-Applikationen, jedoch auf einem System instabile Performance oder Probleme aufweisen, lässt sich auch das Server-Skript in einer festgelegten *Conda*-Umgebung ausführen.

Das digitale Instrument wurde sowohl unter *macOS* mit *Silicon*- als auch *Intel*-Chips getestet. Bei der Nutzung der Server-*Applikation* auf Intel-Macs wurde mit zwingendem *torch*-Gerät *cpu* eine mangelhafte und instabile Performance festgestellt. Hierbei empfiehlt sich das Script für den Server, in einer *Cuda*-Umgebung mit den vorgegebenen *Python*-Modulen laufen zu lassen (siehe Kapitel A)

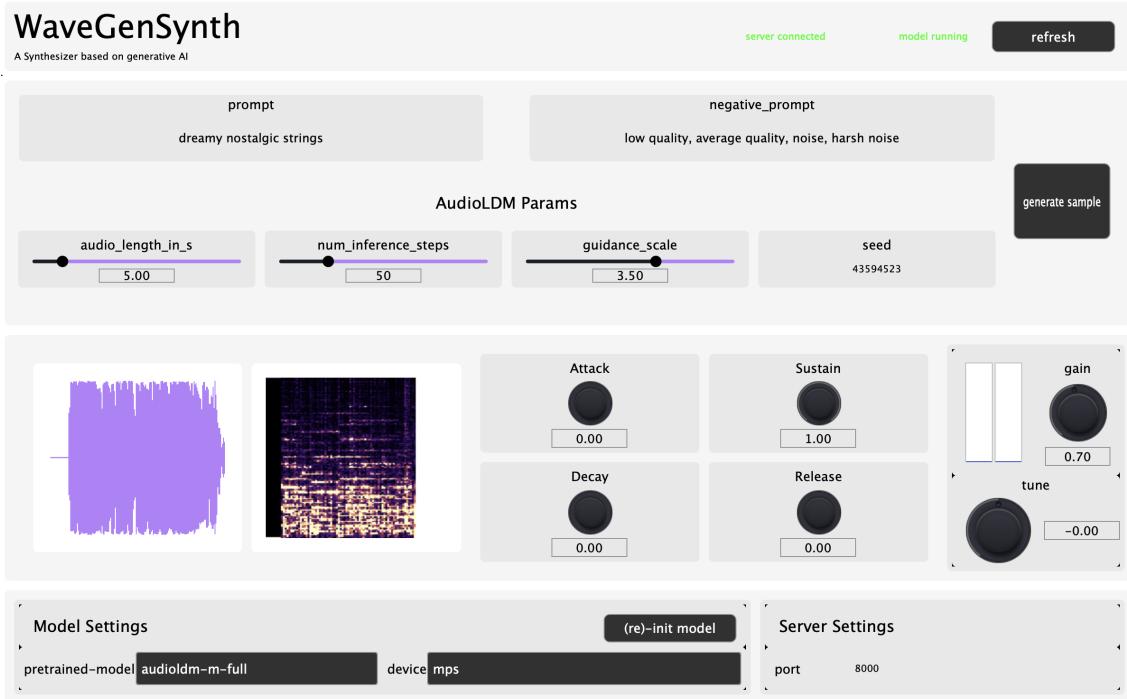
## 4.3. Benutzeroberfläche

Die resultierende Benutzeroberfläche (Abbildung 4.2) erlaubt es Nutzer\*innen, eine Verbindung mit der *API* (siehe Abschnitt 4.1.3) auf einem gewählten Port herzustellen und ein Modell zu initialisieren, indem aus einer Palette von vortrainierten Modellen ausgewählt wird. Die Nutzung des Modells auf diversen Hardwarekonfigurationen (mit und ohne GPU) sowie Betriebssystemen wird durch die Auswahl an unterstützten Geräten ermöglicht. Nach der Initialisierung eines Modells kann ein gewünschter Klang, der durch die Texteingabefelder *Prompt* und *Negative Prompt* spezifiziert wird, synthetisiert werden. Zudem lassen sich die *Länge* des zu erzeugenden Klangs in Sekunden (*audio length*), die Anzahl der *Inferenzschritte* (*number of inference steps*) und die *Leitskala* (*guidance scale*) anpassen. Um die Erzeugung deterministisch zu gestalten, kann ein *Seed* in Form eines 8-Byte großen Integers im Wertebereich von 0 bis  $2^{64} - 1$  angegeben werden.

Die *Hüllkurve* des *Samplers* (siehe 3.1.3) kann über vier Parameter angepasst werden. Hierbei lassen sich die Zeitspannen in Sekunden für die *Anschwell-* (engl. *attack*), *Abschwell-* (engl. *decay*) und *Ausschwingphasen* (engl. *release*) sowie das Niveau, auf das die Lautstärke absinken soll (engl. *sustain*), festlegen. Der von dem Instrument generierte Klang kann durch den Schieberegler *Verstärkung* (engl. *gain*) auf eine spezifische Lautstärke justiert werden.

Da die *Tonhöhe* des generierten Klangs nicht im Voraus festgelegt werden kann und der Klang standardmäßig der *Midi-Note C4* zugeordnet wird, bietet sich die Option, das Instrument mittels des Reglers *tune* so zu stimmen, dass die *Tonhöhe* des Klangs der gespielten Note entspricht. Hierbei lässt sich der Klang kontinuierlich um bis zu zwölf Halbtöne, also einer Oktave, nach oben oder unten modifizieren.

## 4. Methoden



**Abbildung 4.2.:** Entstandene Benutzeroberfläche

Mithilfe eines *MIDI*-Eingabegeräts lassen sich die verschiedene *Tonhöhen* spielen. Das an vielen Keyboards vorhanden *Pitch-Bend*-Rad, welches eine kontinuierliche *Modulation* des aktuell gespielten Tones um bis zu 2 Halbtöne ermöglicht, kann ebenso von dem Instrument verarbeitet werden.

Das erzeugte Signal wird mittels einer Wellenform- und Spektrogramm-Anzeige visualisiert. Dies zielt darauf ab, Nutzer\*innen ein besseres Verständnis der Modellergebnisse zu ermöglichen und das erfolgreiche Generieren zu erkennen. Zudem ermöglicht diese Visualisierung die Identifikation bestimmter Klangstrukturen.

## 4.4. Evaluation

Eine Darstellung des Verhaltens verschiedener Modelle in Reaktion auf spezifische Parameter und Eingaben ist unter <https://suckrowpierre.github.io/TtPS.github.io/> [83] visualisiert ermöglicht eine einfache Evaluierung dieser. Es ist zu beachten, dass diese Zusammenstellung einen Einblick und einen ersten Eindruck in die Nutzbarkeit der verfügbaren Modelle in Kombination verschiedener Parameter und *Prompts* bieten soll. Aufgrund der begrenzten Anzahl an Beispielen und dem Fehlen einer Studie können jedoch keine definitiven Aussagen über Qualität der Modelle getroffen werden. Die Untersuchungen der Modelle und daraus folgende Ergebnisse beziehen sich auf subjektive empfundene Beobachtungen und Eindrücke.

Die in <https://suckrowpierre.github.io/TtPS.github.io/> [83] dargestellten Ergebnisse wurden mittels des auf <https://github.com/suckRowPierre/ThesisModelsResultsGenerator/> veröffentlichten Codes generiert.

Ergebnisse, bei denen die *torch*-Geräte *mps* und *cpu* zum Einsatz kamen, wurden auf einem *Macbook Pro* mit einem *Apple M1 Pro*-Chip unter *macOS Ventura 13.2.1* erzeugt. Ergebnisse, die mithilfe von *Cuda* erzeugt wurden, entstanden in einem *Google-Colab* unter Verwendung einer *T4 GPU*.

Die verwendeten *Prompts* wurden unter anderem durch *GPT-4* [81] auf <https://chat.openai.com/> mittels des folgenden *Prompts* (siehe 4.1) am 8. September 2023 generiert.

---

**Listing 4.1 Benutzer GPT-4 Prompt**

---

For a scientific work of mine, I'm evaluating the music abilities of diffusion models that can generate audio. For that, I have a script that generates audio for different models based on a list of prompts. My list looks like the following. Please add more prompts to test the musical and instrumental abilities of the models. Keep in mind the people using the models would be musicians, studio engineers, and music producers.

```
A single kickdrum  
A kickdrum  
A distorted kickdrum  
A kickdrum with a lot of reverb  
A clap  
A weird clap  
A snare  
A string orchestra  
A analog synth  
A distorted analog synth  
A string synth  
dreamy nostalgic strings  
Ambient pads  
A guitar string  
A distorted guitar string
```

---



## 5. Ergebnisse

Die Integration der erörterten Komponenten und Elemente resultierte in einem digitalen Instrument mit den Namen *WaveGenSynth*, das entweder als eigenständige Anwendung oder als Erweiterung in den Formaten VST3 oder AU innerhalb einer Digital Audio Workstation (*DAW*) wie *Ableton* [1] oder *Logic* [69] verwendet werden kann. Das digitale Instrument ist unter *Github* [34] veröffentlicht<sup>1</sup> (siehe Anhang A).

Die Soundqualität des Instrumentes und die Fähigkeit die gewünschten musikalischen Merkmale aus der textuellen Eingabe zu generieren hängt von dem zur Verwendung kommenden Modell, und der Nutzer spezifischen Eingabe der Parameter und des *Promptes* ab.

Es lässt sich feststellen, dass bei gleichen Parametern und *Prompt* für unterschiedliche *Torch*-Geräte diverse Audiosignale erzeugt werden. Ein großer Unterschied in der Qualität zwischen den Geräten lässt sich anhand der 6 verwendeten *Prompts* nicht eindeutig ermitteln, jedoch zeigt sich eine Tendenz, dass, wenn ein *Prompt* auf den Geräten *mps* und *cuda* ein schlechtes Ergebnis erzielt, das mittels *cpu* generierte Audiosignal für den gleichen *Prompt* noch mehr unerwünschte Artefakte und unangenehme Geräusche aufweist.

Aus [46, 47] geht der erwartete Einfluss des Werts der *Leitskala* (engl. *guidance-scale*) auf die Signalqualität hervor. Obwohl höhere Werte den beschriebenen *Prompt* präziser abbilden, führen sie zu einer verminderten Audioqualität. Zur Untersuchung dieses Sachverhalts wurden Audiosignale der *Prompts* („Gentle guitar strum, Tribal African drum circle, vocal harmonies“) mittels sämtlicher verfügbarer vortrainierter Modelle und *Leitskala*-Werten von 1, 2, 3, 4, 5 generiert. Es zeigt sich, dass Audiosignale, die mit einem Wert von 1 erzeugt wurden, eine verminderte Audioqualität aufweisen. Werte zwischen 2 – 4 scheinen hingegen eine bessere Audioqualität zu gewährleisten. Die bedeutenden Audiocharakteristika scheinen sich in diesem Bereich zu kristallisieren, und tiefe *Frequenzen* treten in Erscheinung. Weiterhin bestätigt sich, dass höhere Werte den gegebenen *Prompt* detaillierter repräsentieren. Ein exemplarischer Fall hierfür ist der *Prompt* „Gentle guitar strum“, generiert mit *audioldm-m-full* [17]. Bei diesem zeigen Signale mit einem Wert unter 4 diverse Audioereignisse, die an eine Gitarre erinnern. Erst bei einem Wert von 4 oder höher ist ein einzelnes, eindeutiges Gitarrenzupfen zu identifizieren.

Die *Inferenzschritte* (engl. *Inference steps*) geben die Anzahl der Entrauschungsschritte im *Diffusions*-prozess an. Eine höhere Anzahl dieser Schritte soll zu verbesserten Ergebnissen bei gleichzeitig längerer *Inferenz*-zeit führen [46, 47]. Um die Auswirkungen der *Inferenzschritte* zu analysieren, wurden wieder Audiosignale aus drei unterschiedlichen *Prompts* („ambient texture“, „techno kickdrum“, „Long evolving drone“) mithilfe aller vorab trainierten Modelle und für *Inferenzschritte*-Anzahlen von 5, 10, 20, 50, 100, 200, 400 generiert. Hierbei zeigten die Werte 5 und 400 die ungünstigsten Ergebnisse: Signale mit 5 Schritten resultierten in dumpfen und wenig strukturierten Klängen. Hingegen schienen Werte im Bereich von 10 – 200 adäquate Resultate zu liefern. Es wurden oftmals nur marginale Qualitätsveränderungen pro Schritt beobachtet. Die Ergebnisse weisen zudem darauf hin, dass *AudioLDM2* im Vergleich zu *AudioLDM* eine größere Anzahl an Schritten benötigt, um qualitativ hochwertige Signale zu produzieren. Bei 400 Schritten hingegen erscheint die Anzahl überdimensioniert, da das resultierende Signal verstärkt abstrakte Klangereignisse und unerwünschte Artefakte aufweist.

Um die Auswirkungen der gewünschten Audiosignallänge auf die Resultate zu untersuchen, wurden für die drei *Prompts* („Chirping birds at dawn“, „A choir pad“, „An ambient electronic pad“) mittels der Modelle *audioldm-m-full* [17] und *audioldm2* [18] Signale mit den Längen 5, 10, 20, 30 generiert. Bei ausgedehnteren

---

<sup>1</sup><https://github.com/suckrowPierre/WaveGenSynth>

## 5. Ergebnisse

---

Signalen scheint die Qualität leicht abzunehmen. Offenbar haben die Modelle Schwierigkeiten, die Konsistenz über längere Zeiträume hinweg beizubehalten. In einigen dieser Beispiele manifestiert sich ein Phänomen, das als *Pitchdrifting* bezeichnet wird, wodurch die wahrgenommene *Tonhöhe* variiert. Diese leicht reduzierte Qualität könnte jedoch gezielt als stilistisches Mittel verwendet werden.

Die Einwirkung verschiedener *Negative-Prompts* (engl. *negative prompt*) auf die *Prompts* („Soft flute note“, „Choir“, „A muted trumpet“) wurde analysiert. Mittels der Modelle *audioldm-m-full* [17] und *audioldm2-music* [19] wurden Signale unter Verwendung der *Negativen Prompts* („low quality“, „average quality“, „harsh noise“, „dissonant chords“, „distorted sounds“, „clashing frequencies“, „feedback loop“, „clattering“, „inharmonious“, „noise“, „high pitch“, „artefacts“) generiert. Die Beobachtungen zeigen, dass einige *Negative-Prompts* das Ergebnis positiv beeinflussen, während andere das Signal mit zusätzlichem *Rauschen* beeinträchtigen. Insbesondere „low quality“, „noise“, „harsh noise“ und „average quality“ schienen das Resultat am meisten zu verbessern. Der *Negative-Prompt* „distorted sounds“ zeigte in einigen Fällen positive Effekte, bewirkte jedoch im Zusammenhang mit dem *Prompt* „Soft flute note“ und dem Modell *audioldm-m-full* eine Verschlechterung. „high pitch“ scheint hohe *Obertöne* zu unterdrücken und tiefere Töne bekommen mehr Präsenz, was eine bewusste Entscheidung seien sollte. Andere *Negative-Prompts*, wie „dissonant chords“, „distorted sounds“, „clashing frequencies“, „feedback loop“, „clattering“, „inharmonious“ und „artefacts“, scheinen das Resultat zu beeinträchtigen. Die anfängliche Annahme, dass *AudioLDM2* aufgrund von *LOLA* effizienter mit *Negative-Prompts* interagieren könnte, wurde durch die ungünstigen Ergebnisse von *audioldm2-music* in den gegebenen Tests nicht bestätigt.

Erwartungen, dass *Prompts* wie „A kickdrum“, „A snare“, „Loud clap sound“, „A gong hit“ ein singuläres, nicht wiederholendes Audioereignis erzeugen, werden nicht erfüllt. In den meisten Fällen produzieren die Modelle Signale, in denen Audioereignisse mehrfach auftreten. Die Modelle scheinen nicht darauf ab zu zielen, das gewünschte einzelne Ereignis zu synthetisieren, sondern generieren das angestrebte Ergebnis in einem musikalischen Rhythmus. Eine präzise Formulierung, die ausschließlich ein einzelnes Audioereignis und einen Ton erwartet – ausgedrückt durch „A single“ –, scheint bei *audioldm2* [18] effektiver zu sein. Im Gegensatz dazu zeigt *audioldm2-music* [19] auch mit dieser Formulierung keine Ergebnisse mit weniger wiederholenden Strukturen. Es wird vermutet, dass *audioldm2-music* [19] dazu neigt, wiederholende Strukturen zu generieren, da sein Hauptziel die Generierung musikalischer Ergebnisse und nicht eines isolierten Klangs ist.

Der Versuch, instrumententypische Klänge durch die *Prompts* „FM synthesis bells“, „mellotron chords“, „A bagpipe melody“, „A guitar string“ und „A piano chord“ zu generieren, brachte nicht die gewünschten Ergebnisse. Die erzeugten Signale klingen abstrakt und stellen keine akkurate Audiorepräsentation der jeweiligen Eingabe dar. Zudem ist festzustellen, dass weniger zufriedenstellende Ergebnisse von *AudioLDM2* eine Neigung haben, ähnliche zu klingen.

Das Schmücken des *Prompts* mit Emotionen und zusätzlichen Beschreibungen, wie beispielsweise „Dark pad sound“, „An ethereal shimmering synth pad“, „An angelic choir“, „dreamy nostalgic strings“ und „a sad violin solo“, scheint die generierten Signale interessanter zu gestalten. Dies ermöglicht den Anwendern, den zu erzeugenden Klang stärker in eine bestimmte Richtung zu beeinflussen. Hierbei wurde wieder beobachtet, dass die Ergebnisse von *AudioLDM2* trotz variierender *Prompts* eine Tendenz zu ähnlichen Klangstrukturen zeigen.

Die Modelle scheinen auf bestimmte Beschreibungen musikalischer und akustischer Effekte adäquat reagieren zu können. Das langanhaltende Ausklingen eines Tones, umschrieben durch den Ausdruck „long sustain“, funktioniert offenbar zufriedenstellend. Ebenso scheint das Hinzufügen von Hall durch „reverb“, Verzerrung mittels „distortion“ und das Umkehren eines Tones durch „reverse“ gut zu funktionieren. Komplexere Effekte wie Echo („echo“), Verstimmung („detune“) und Verzögerung („delay“) können gelegentlich zu den gewünschten Ergebnissen führen. Interessanterweise scheinen die *AudioLDM*-Modelle in diesem Aspekt besser abzuschneiden als die *AudioLDM2*-Modelle, obwohl man aufgrund der *LOA*-Repräsentation erwarten könnte, dass letztere bessere Ergebnisse erzielen könnten.

---

Bei dem Versuch, *Prompts* mit spezifischen, in der Musikproduktion oder der Pop-Kultur häufig verwendeten Klängen und Begriffen zu definieren scheiterte. Die Bemühungen, eine im Hip-Hop häufig verwendete *808-Kickdrum* oder eine in der elektronischen Musik verwendete *909-Kickdrum* zu erzeugen, lieferten Ergebnisse, deren *Klangfarbe* nicht den Vorgaben entsprach. Auch der Versuch, einen *Amen-Break*, das wohl am meisten verwendete *Sample* weltweit, zu erzeugen, brachte nicht das gewünschte Ergebnis. Eine Ähnlichkeit mit einer *303 Baseline* konnte lediglich durch *audioldm-m-full* erreicht werden, während die anderen Modelle scheiterten. Die *Prompts* „A Juno-106 pad“ und „Oberheimer OB-Xa string pads“ erzeugten verwendbare *Pads*, also lange gehaltene Töne oder Akkorde. Es lässt sich jedoch streiten, ob sie die klanglichen Feinheiten dieser Instrumente besitzen.



## 6. Diskussion

Das Aufzeigen, der Möglichkeit Instrumente basierend auf künstlicher Intelligenz zu entwickeln, stellt eine bedeutende Erweiterung in der musikalischen Klangerzeugung dar. Die Analyse des digitalen Instruments und die Bewertung der anwendbaren Modelle deuten darauf hin, dass die Generierung, Manipulation und Reproduktion von Klängen, die durch künstliche Intelligenz synthetisiert wurden, ein hohes kreatives Potential aufweisen. Es treten jedoch bestimmte Einschränkungen und Schwachstellen auf. Zu diesen zählen insbesondere die identifizierten Mängel, das gelegentlich unvorhersehbare Verhalten der Modelle sowie die begrenzte Implementierung des *Samplers*.

Es zeigt sich, dass die analysierten Modelle bisher nicht angemessen auf Eingabeauflforderungen im Bereich der Musikproduktion reagieren. Die Unfähigkeit einiger Modelle, einzelne Klangereignisse zu generieren, das Fehlen der Modellierung erwarteter Klangmerkmale spezifischer Instrumente sowie das Auftreten verrauschter Ergebnisse weisen auf bestehende Defizite hin. Diese Mängel könnten durch das Training mit umfangreicheren Audiodaten, die in der Musikproduktion eingesetzt werden, adressiert werden. Potenzielle Quellen für solche Trainingsdaten sind *Synthesizer*- oder *Sampler*-manufakturen, Audiobibliotheken wie *Splice* [96], Musikstudios oder auch Produzenten selbst. Privatpersonen, die sich mit Musikproduktion beschäftigen, verfügen häufig über eigene Klangbibliotheken, die in ihren Werken Anwendung finden. Auf Basis dieser Daten könnte ein Modell weiter verfeinert werden, um es individueller und persönlicher zu gestalten. Eine solche Personalisierung von *AudioLDM* [63] wurde in [86] untersucht und könnte in *generative AI*-Werkzeuge für Musikproduzenten integriert werden. Eine weitere Überlegung wäre, die Klangbibliotheken verschiedener Privatpersonen zu einer umfangreichen Datenbasis zusammenzuführen, die anschließend für das Finetuning oder Training verwendet werden könnte. Ein solcher, von der Community getriebener Ansatz, könnte ein demokratisches Mittel darstellen, um den Mangel an Datensätzen in diesem Bereich zu beheben.

Benutzte Modelle sollten darauf abzielen, Ergebnisse mit der von der *Audio Engineering Society* empfohlenen *Sampling*-Rate von 48kHz [5] zu erzeugen. Die Modelle *AudioLDM* [63] und *AudioLDM2* [64] produzieren Ergebnisse mit einer Rate von 16kHz und sind daher nicht geeignet, um Klänge in *High Fidelity* zu generieren. Eine kürzere *Inferenzzeit* könnte die *User-Experience* des Instruments positiv beeinflussen.

Es ist wünschenswert, dass eingesetzte Modelle Ergebnisse mit der von der *Audio Engineering Society* empfohlenen *Sampling*-Rate von 48kHz [5] liefern. Die Modelle *AudioLDM* [63] und *AudioLDM2* [64] generieren Ergebnisse mit einer Rate von 16kHz und sind daher für die Generierung von Klängen in *High Fidelity* nicht optimal. Eine verkürzte *Inferenzzeit* könnte zudem die *User-Experience* des Instruments verbessern.

Das Scheitern der Erstellung einer *ONNX*- bzw. *TorchScript*-Repräsentation des Modells für die *Inferenz* in C++ führt zu Komplikationen bei der Bündelung und Verteilung des Modells in Verbindung mit der *Python API*. Durch die Erstellung einer solchen *binären* Datei für das verwendete Modell würde die Notwendigkeit eines Servers und einer *API* entfallen. Ebenso wäre der Einsatz mehrerer Programmiersprachen für eine Implementierung nicht länger erforderlich.

Die rudimentäre Implementierung des *Samplers* ermöglicht nicht die Definition von Start- und Endpunkten für die Wiedergabe des generierten Signals. Solche Parameter könnten eine intuitive Möglichkeit bieten, das gewünschte Klangereignis aus mehreren oder wiederholten Ereignissen auszuwählen. Zudem fehlt die Möglichkeit, spezifische Klangsegmente zu wiederholen, um einen durchgängigen Klang zu erzeugen – eine Funktion, die in industriellen *Samplern* seit den 1990er Jahren üblich ist. Vor diesem Hintergrund stellt sich die Frage nach der Notwendigkeit, einen eigenen *Sampler* zu entwickeln. Alternativ könnte eine Benutzeroberfläche ausreichen, die sich in bestehende Musikproduktionssoftware integrieren lässt, um

## 6. Diskussion

---

Klänge zu generieren und mittels der Werkzeuge des jeweiligen Software-Ökosystems zu manipulieren. Eine solches Anwendung könnte, vorausgesetzt das Modell ist leistungsstark genug, Anwendungen wie *Splice* [96] überflüssig machen. Die in *AudioLDM* [63] präsentierten Generierungsfunktionen, wie Inpainting oder Style-Transfer, könnten in einem solchen Werkzeug integriert werden. Dies würde Nutzer\*innen beispielsweise erlauben, unerwünschte Abschnitte eines Spektrogramms zu eliminieren oder Audiodateien zu importieren und durch Eingabeaufforderungen zu verändern.

Sollte die Entwicklung eines speziell angefertigten *Samplers* fortgesetzt werden, könnten diesem auch Elemente des maschinellen Lernens hinzugefügt werden. Beim Spielen unterschiedlicher Noten auf einem Klavier erzeugen diese verschiedene *Klangfarben* und folglich unterschiedliche Spektren [82]. Es genügt nicht, den Klang lediglich auf die gewünschte *Frequenz* zu *modulieren*. Um ein realistischeres Ergebnis zu erzielen, müssen die feinen Unterschiede der *Oberschwingungen* für jede *Tonhöhe* angepasst werden. Ein solches Anpassungsverfahren könnte durch ein ML-Modell erlernt werden. Als Eingabe würde ein Audiosignal dienen, aus welchem das Modell Signale für alle möglichen *Tonhöhe* generiert. Für das Training eines solchen Modells wäre eine umfangreiche Sammlung von Audiosignalen erforderlich, die alle *Tonhöhe* verschiedener Instrumente abdeckt.

Einige Musiker\*innen bevorzugen es, sich von Computern zu distanzieren und spezialisierte Hardware zu nutzen, die ein haptischeres Feedback bietet. Die in dieser Arbeit präsentierte spielbare Klangsynthese könnte auf ein dediziertes Hardware-Gerät übertragen werden. Lösungen für *Deep Learning* auf Microcontrollern [99] können auf dedizierten Hardware-Plattformen, wie den *Coral*-Geräten [16] von Google oder den *Jetson*-Geräten [50] von Nvidia, betrieben werden. Der Code für das Instrument müsste dann lediglich für eine bestimmte Hardware entwickelt und gewartet werden. Dies würde für Musiker\*innen, die keine Kenntnisse in *Python* oder *C++* besitzen, den Vorteil bieten, dass sie sich nicht mit der Installation und Betreiben der Software auseinandersetzen müssen. Einige Microcontroller-Lösungen, wie *Coral*, erfordern jedoch, dass das verwendete Modell mit *TensorFlow Lite* [105] kompatibel ist. Hierbei stellt sich wieder die Frage, ob umfangreiche und komplexe Modelle, wie *AudioLDM* [63] und *AudioLDM2* [64], in dieses Format konvertiert werden können.

Weitere Mängel des Instruments, die adressiert und optimiert werden könnten, betreffen beispielsweise das manuelle Stimmen, welches sich automatisierten ließ. Ein weiteres Defizit besteht darin, dass die bisher generierten Klänge und Einstellungen sich nicht, wie bei vielen digitalen Instrumenten üblich, speichern und wieder abrufen lassen. Für die Erstellung dieses ersten Prototyps wurde auf die *Spezifikation und Verifikation* des *Quellcodes* verzichtet. Bei einer Weiterverfolgung der Implementierung sollte dieser Aspekt jedoch berücksichtigt werden, um potenzielle *Bugs* und Fehler im Code zu minimieren.

Ungeachtet der genannten Defizite und Schwachstellen hat sich gezeigt, dass das digitale Instrument mithilfe der vorgestellten *Diffusions*-Modelle in der Lage ist, spielbare und manipulierbare Klänge aus benutzerspezifischen Eingaben zu generieren. Weiterhin wurde bestätigt, dass das Instrument in moderne Musikproduktionsprozesse und die damit verbundenen Anwendungen integriert werden kann. Es erschließt eine neue Form der Soundsynthese durch künstliche Intelligenz und bietet somit innovative Möglichkeiten, Klanglandschaften zu gestalten. Ein wesentlicher Vorteil gegenüber der Generierung anderer Medien besteht darin, dass auch unerwartete oder minderwertige Ergebnisse weiterverwendet werden können, indem sie durch diverse Effekte und Prozesse umfassend bearbeitet und transformiert werden. Ein Beispiel hierfür liefert der Musiker „Heinbach“, der in [38] erläutert, wie er den von „Bob Ross“ geprägten Begriff *Happy Little Accidents* in den Musikbereich überträgt und unerwartete Klänge musikalisch aufarbeitet, um neue *Klangfarben* und -landschaften zu schaffen. Zusammenfassend lässt sich feststellen, dass das entwickelte Instrument in seiner aktuellen Form mit den verfügbaren generativen Modellen weniger zur qualitativ hochwertigen Modellierung spezifischer Klänge geeignet ist, sondern vielmehr zum Experimentieren mit neuen, unerwarteten Klängen verlockt.

## 7. Schlussfolgerung

Diese Arbeit zielte darauf ab, musikalische Konzepte, elektronischer Klangerzeugung, digitaler Audiorepräsentation, *generative AI* mittels *Latenter Diffusion* und Softwareframeworks zu verknüpfen. Es wurde erfolgreich ein digitales Instrument entwickelt, das die Erzeugung innovativer, spielbarer und manipulierbarer Klangergebnisse ermöglicht. Dies unterstreicht die Machbarkeit eines solchen Unterfangens, auch wenn es bestimmten Einschränkungen und Defiziten erfasst wurden.

Die Komplexität der verwendeten Modelle *AudioLDM* [63] und *AudioLDM2* [64] ließ eine erfolgreiche Konvertierung in eine binäre Repräsentation scheitern und beschränkte dadurch die möglichen Implementierungsmethoden. Es wurde sich für die Implementierung des Instruments über eine lokal gehostete *API* entschieden, die die vortrainierten Modelle *AudioLDM* und *AudioLDM2* initialisieren und mit diesen *Inferenz* betreiben kann. Dieser Ansatz führte allerdings zu einer erhöhten Komplexität bei der Bündelung und Verteilung des Instruments, wodurch zwei separate ausführbare Applikationen entstanden.

Die Auswertung der Qualität der Ergebnisse von *AudioLDM* [63] und *AudioLDM2* [64] im Hinblick auf die erwarteten *Prompts* im Kontext der Nutzung des digitalen Instruments offenbarte Limitierungen. Diese Einschränkungen begrenzen die Anwendbarkeit des Instruments primär auf die Exploration neuer teils unerwarteter Klänge. Dennoch, trotz dieser Limitierungen liegt hierin das Potenzial, diese unvorhergesehenen und vielleicht befremdlichen Klänge als Basis für weitere Manipulationen und Umformung zu nutzen. Zukünftig könnten verbesserte und weiterentwickelte Modelle bei gesteigerter Klangqualität und reduzierten Artefakten im Klang konventionellen Instrumenten gleich kommen. Das Potenzial eines individuellen *finetuning* durch eigene Klangbibliotheken wurde erkannt und auf eine Verbesserung der *Abtastrate* auf 48kHz hingewiesen.

Die Abwesenheit von Funktionen, die es erlauben, Start- und Endpunkte der wiedergegebenen Töne anzupassen sowie Loop-Points zu definieren, für ein kontinuierliches Anhalten eines Klanges, limitiert die Bearbeitungsmöglichkeiten des generierten Klanges. Zudem würde das Instrument von zusätzlichen Optimierungen wie einer automatischen Stimmfunktion und der Möglichkeit, Klänge zu speichern und abzurufen, profitieren.

Trotz dieser Mängel ist es unbestreitbar, dass die Arbeit das Potenzial der musikalischen Anwendung von *Diffusions*-Modellen unterstreicht, und eine verbesserte und vielseitigere Integrationsmöglichkeit in moderne Musikproduktionsökosysteme im Vergleich zu ähnlichen Anwendungen bietet. Anwender können Klänge nach ihren Vorstellungen parametrisieren und erzeugen, das Klangverhalten dieser beim Spielen durch eine *Envelope* modifizieren, das Instrument stimmen und die Lautstärke justieren. Eingabegeräte können mittels *MIDI* angeschlossen und das *Pitch-Bend*-Rad kann ebenso verwendet werden.

Es ist zu hoffen, dass zukünftige Forschungen in diesem Bereich von den in dieser Arbeit gemachten Entdeckungen profitieren und diese weiterführen werden. Perspektiven für weitere diverse potenzielle Integration generativer Künstlicher Intelligenz in den Ablauf der Musikproduktion wurden aufgezeigt.



# Literaturverzeichnis

- [1] Ableton. URL: <https://www.ableton.com/> (besucht am 14. 08. 2023).
- [2] Abien Fred Agarap. *Deep Learning using Rectified Linear Units (ReLU)*. 7. Feb. 2019. arXiv: 1803.08375[cs, stat]. URL: <http://arxiv.org/abs/1803.08375> (besucht am 19. 08. 2023).
- [3] Andrea Agostinelli, Timo I. Denk, Zalán Borsos, Jesse Engel, Mauro Verzetti, Antoine Caillon, Qingqing Huang, Aren Jansen, Adam Roberts, Marco Tagliasacchi, Matt Sharifi, Neil Zeghidour, Christian Frank. *MusicLM: Generating Music From Text*. 26. Jan. 2023. doi: 10.48550/arXiv.2301.11325. arXiv: 2301.11325[cs, eess]. URL: <http://arxiv.org/abs/2301.11325> (besucht am 08. 10. 2023).
- [4] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever „Language Models are Unsupervised Multitask Learners“. In: 2019. URL: <https://api.semanticscholar.org/CorpusID:160025533>.
- [5] Audio Engineering Society, Inc. *AES5-2018 (Rev. AES5-2008) - AES recommended practice for professional digital audio — Preferred sampling frequencies for applications employing pulse-code modulation*. AES5-2018. 697 3rd Ave, New York, NY 10017, US: Audio Engineering Society, Inc., 22. Dez. 2018. URL: <https://www.aes.org/tmpFiles/aessc/20231014/aes05-2018-r2023-i.pdf> (besucht am 14. 10. 2023).
- [6] Jacob Austin, Daniel D. Johnson, Jonathan Ho, Daniel Tarlow, Rianne van den Berg. *Structured Denoising Diffusion Models in Discrete State-Spaces*. 22. Feb. 2023. arXiv: 2107.03006[cs]. URL: <http://arxiv.org/abs/2107.03006> (besucht am 11. 08. 2023).
- [7] Barney Hill. *VroomAI*. original-date: 2023-03-24T23:58:48Z. 13. Juli 2023. URL: <https://github.com/vroomai/vst> (besucht am 28. 07. 2023).
- [8] Zalán Borsos, Raphaël Marinier, Damien Vincent, Eugene Kharitonov, Olivier Pietquin, Matt Sharifi, Olivier Teboul, David Grangier, Marco Tagliasacchi, Neil Zeghidour. *AudioLM: a Language Modeling Approach to Audio Generation*. 7. Sep. 2022. arXiv: 2209.03143[cs, eess]. URL: <http://arxiv.org/abs/2209.03143> (besucht am 03. 04. 2023).
- [9] Richard Charles Boulanger, Victor Lazzarini, Hrsg. *The audio programming book*. OCLC: ocn503654559. Cambridge, Mass: MIT Press, 2011. 889 S. ISBN: 978-0-262-01446-5.
- [10] Ke Chen, Xingjian Du, Bilei Zhu, Zejun Ma, Taylor Berg-Kirkpatrick, Shlomo Dubnov. *HTS-AT: A Hierarchical Token-Semantic Audio Transformer for Sound Classification and Detection*. 1. Feb. 2022. arXiv: 2202.00874[cs, eess]. URL: <http://arxiv.org/abs/2202.00874> (besucht am 19. 08. 2023).
- [11] Yu-An Chung, Yu Zhang, Wei Han, Chung-Cheng Chiu, James Qin, Ruoming Pang, Yonghui Wu. *W2v-BERT: Combining Contrastive Learning and Masked Language Modeling for Self-Supervised Speech Pre-Training*. 13. Sep. 2021. arXiv: 2108.06209[cs, eess]. URL: <http://arxiv.org/abs/2108.06209> (besucht am 07. 10. 2023).

- [12] Yu-An Chung, Yu Zhang, Wei Han, Chung-Cheng Chiu, James Qin, Ruoming Pang, Yonghui Wu. *W2v-BERT: Combining Contrastive Learning and Masked Language Modeling for Self-Supervised Speech Pre-Training*. 13. Sep. 2021. arXiv: 2108.06209[cs, eess]. URL: <http://arxiv.org/abs/2108.06209> (besucht am 07.08.2023).
- [13] Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, Albert Webson, Shixiang Shane Gu, Zhuyun Dai, Mirac Suzgun, Xinyun Chen, Aakanksha Chowdhery, Alex Castro-Ros, Marie Pellat, Kevin Robinson, Dasha Valter, Sharan Narang, Gaurav Mishra, Adams Yu, Vincent Zhao, Yanping Huang, Andrew Dai, Hongkun Yu, Slav Petrov, Ed H. Chi, Jeff Dean, Jacob Devlin, Adam Roberts, Denny Zhou, Quoc V. Le, Jason Wei. *Scaling Instruction-Finetuned Language Models*. 6. Dez. 2022. arXiv: 2210.11416[cs]. URL: <http://arxiv.org/abs/2210.11416> (besucht am 06.10.2023).
- [14] *CMake*. URL: <https://cmake.org/> (besucht am 11.06.2023).
- [15] Jade Copet, Felix Kreuk, Itai Gat, Tal Remez, David Kant, Gabriel Synnaeve, Yossi Adi, Alexandre Défossez. *Simple and Controllable Music Generation*. 8. Juni 2023. doi: 10.48550/arXiv.2306.05284. arXiv: 2306.05284[cs, eess]. URL: <http://arxiv.org/abs/2306.05284> (besucht am 08.10.2023).
- [16] *Coral AI Products*. Coral AI. URL: <https://coral.ai/products/> (besucht am 15.10.2023).
- [17] *cvssp/audioldm-m-full*. Hugging Face. URL: <https://huggingface.co/cvssp/audioldm-m-full> (besucht am 05.10.2023).
- [18] *cvssp/audioldm2*. Hugging Face. URL: <https://huggingface.co/cvssp/audioldm2> (besucht am 05.10.2023).
- [19] *cvssp/audioldm2-music*. Hugging Face. URL: <https://huggingface.co/cvssp/audioldm2-music> (besucht am 05.10.2023).
- [20] Damai Dai, Yutao Sun, Li Dong, Yaru Hao, Shuming Ma, Zhifang Sui, Furu Wei. *Why Can GPT Learn In-Context? Language Models Implicitly Perform Gradient Descent as Meta-Optimizers*. 15. Mai 2023. arXiv: 2212.10559[cs]. URL: <http://arxiv.org/abs/2212.10559> (besucht am 13.08.2023).
- [21] Alexandre Défossez, Jade Copet, Gabriel Synnaeve, Yossi Adi. *High Fidelity Neural Audio Compression*. 24. Okt. 2022. doi: 10.48550/arXiv.2210.13438. arXiv: 2210.13438[cs, eess, stat]. URL: <http://arxiv.org/abs/2210.13438> (besucht am 09.10.2023).
- [22] Soham Deshmukh, Benjamin Elizalde, Huaming Wang. *Audio Retrieval with WavText5K and CLAP Training*. 28. Sep. 2022. doi: 10.48550/arXiv.2209.14275. arXiv: 2209.14275[cs, eess]. URL: <http://arxiv.org/abs/2209.14275> (besucht am 08.10.2023).
- [23] Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 24. Mai 2019. arXiv: 1810.04805[cs]. URL: <http://arxiv.org/abs/1810.04805> (besucht am 10.08.2023).
- [24] Prafulla Dhariwal, Alex Nichol. *Diffusion Models Beat GANs on Image Synthesis*. 1. Juni 2021. arXiv: 2105.05233[cs, stat]. URL: <http://arxiv.org/abs/2105.05233> (besucht am 04.09.2023).
- [25] Timur Doumler. „C++ in the Audio Industry“. CppCon 2015. CppCon, 21. Sep. 2015. URL: <https://www.youtube.com/watch?v=boPE02auJj4> (besucht am 16.06.2023).

- [26] Konstantinos Drossos, Samuel Lipping, Tuomas Virtanen. *Clotho: An Audio Captioning Dataset*. 21. Okt. 2019. arXiv: 1910.09387[cs, eess]. URL: <http://arxiv.org/abs/1910.09387> (besucht am 19. 08. 2023).
- [27] Dudenredaktion. *Synthesizer auf Duden online*. Duden online. URL: <https://www.duden.de/rechtschreibung/Synthesizer> (besucht am 16. 07. 2023).
- [28] Benjamin Elizalde, Soham Deshmukh, Mahmoud Al Ismail, Huaming Wang. *CLAP: Learning Audio Concepts From Natural Language Supervision*. arXiv.org. 9. Juni 2022. URL: <https://arxiv.org/abs/2206.04769v1> (besucht am 08. 10. 2023).
- [29] Patrick Esser, Robin Rombach, Björn Ommer. *Taming Transformers for High-Resolution Image Synthesis*. 23. Juni 2021. arXiv: 2012.09841[cs]. URL: <http://arxiv.org/abs/2012.09841> (besucht am 05. 09. 2023).
- [30] *FastAPI*. URL: <https://fastapi.tiangolo.com/> (besucht am 16. 07. 2023).
- [31] Jort F. Gemmeke, Daniel P. W. Ellis, Dylan Freedman, Aren Jansen, Wade Lawrence, R. Channing Moore, Manoj Plakal, Marvin Ritter. „Audio Set: An ontology and human-labeled dataset for audio events“. In: *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). New Orleans, LA: IEEE, März 2017, S. 776–780. ISBN: 978-1-5090-4117-6. doi: 10.1109/ICASSP.2017.7952261. URL: <http://ieeexplore.ieee.org/document/7952261/> (besucht am 12. 08. 2023).
- [32] Deepanway Ghosal, Navonil Majumder, Ambuj Mehrish, Soujanya Poria. *Text-to-Audio Generation using Instruction-Tuned LLM and Latent Diffusion Model*. 29. Mai 2023. arXiv: 2304.13731[cs, eess]. URL: <http://arxiv.org/abs/2304.13731> (besucht am 04. 08. 2023).
- [33] Rohit Girdhar, Alaaeldin El-Nouby, Zhuang Liu, Mannat Singh, Kalyan Vasudev Alwala, Armand Joulin, Ishan Misra. *ImageBind: One Embedding Space To Bind Them All*. 31. Mai 2023. arXiv: 2305.05665[cs]. URL: <http://arxiv.org/abs/2305.05665> (besucht am 06. 10. 2023).
- [34] *GitHub*. GitHub. URL: <https://github.com/github> (besucht am 12. 10. 2023).
- [35] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio. *Generative Adversarial Networks*. 10. Juni 2014. arXiv: 1406.2661[cs, stat]. URL: <http://arxiv.org/abs/1406.2661> (besucht am 20. 08. 2023).
- [36] Google AI. *NSynth: Neural Audio Synthesis*. Magenta. 6. Apr. 2017. URL: <https://magenta.tensorflow.org/nsynth> (besucht am 28. 07. 2023).
- [37] Team Gradio. *Gradio*. URL: <https://gradio.app> (besucht am 03. 08. 2023).
- [38] HAINBACH. *How Happy Accidents Can Inspire Your Music*. 23. Nov. 2021. URL: <https://www.youtube.com/watch?v=Ibajy9A91ls> (besucht am 22. 06. 2023).
- [39] Haohe Liu. *AudioLDM: Text-to-Audio Generation with Latent Diffusion Models*. 27. März 2023. URL: [https://www.youtube.com/watch?v=6qtL9\\_T8m3c](https://www.youtube.com/watch?v=6qtL9_T8m3c) (besucht am 25. 07. 2023).
- [40] Michael T. Heideman, Don H. Johnson, C. Sidney Burrus. „Gauss and the history of the fast Fourier transform“. In: *Archive for History of Exact Sciences* 34.3 (1985), S. 265–277. ISSN: 0003-9519, 1432-0657. doi: 10.1007/BF00348431. URL: <http://link.springer.com/10.1007/BF00348431> (besucht am 22. 06. 2023).

## Literaturverzeichnis

---

- [41] Jonathan Ho, Ajay Jain, Pieter Abbeel. *Denoising Diffusion Probabilistic Models*. 16. Dez. 2020. arXiv: 2006.11239[cs, stat]. URL: <http://arxiv.org/abs/2006.11239> (besucht am 06.04.2023).
- [42] Qingqing Huang, Aren Jansen, Joonseok Lee, Ravi Ganti, Judith Yue Li, Daniel P. W. Ellis. *MuLan: A Joint Embedding of Music Audio and Natural Language*. 25. Aug. 2022. DOI: 10.48550/arXiv.2208.12415. arXiv: 2208.12415[cs, eess, stat]. URL: <http://arxiv.org/abs/2208.12415> (besucht am 09.10.2023).
- [43] Rongjie Huang, Jiawei Huang, Dongchao Yang, Yi Ren, Luping Liu, Mingze Li, Zhenhui Ye, Jinglin Liu, Xiang Yin, Zhou Zhao. *Make-An-Audio: Text-To-Audio Generation with Prompt-Enhanced Diffusion Models*. 29. Jan. 2023. arXiv: 2301.12661[cs, eess]. URL: <http://arxiv.org/abs/2301.12661> (besucht am 19.03.2023).
- [44] Po-Yao Huang, Hu Xu, Juncheng Li, Alexei Baevski, Michael Auli, Wojciech Galuba, Florian Metze, Christoph Feichtenhofer. *Masked Autoencoders that Listen*. 12. Jan. 2023. arXiv: 2207.06405[cs, eess]. URL: <http://arxiv.org/abs/2207.06405> (besucht am 05.10.2023).
- [45] *Hugging Face – The AI community building the future*. 17. Okt. 2023. URL: <https://huggingface.co/> (besucht am 18.10.2023).
- [46] *Huggingface-AudioLDM Pipeline*. Huggingface. URL: <https://huggingface.co/docs/diffusers/main/en/api/pipelines/audioldm> (besucht am 16.07.2023).
- [47] *Huggingface-AudioLDM2 Pipeline*. Hugging Face. URL: <https://huggingface.co/docs/diffusers/main/en/api/pipelines/audioldm2> (besucht am 05.10.2023).
- [48] Vladimir Iashin, Esa Rahtu. *Taming Visually Guided Sound Generation*. 17. Okt. 2021. arXiv: 2110.08791[cs, eess]. URL: <http://arxiv.org/abs/2110.08791> (besucht am 10.08.2023).
- [49] C. Jarzynski. „Equilibrium free energy differences from nonequilibrium measurements: a master equation approach“. In: *Physical Review E* 56.5 (1. Nov. 1997), S. 5018–5035. ISSN: 1063-651X, 1095-3787. DOI: 10.1103/PhysRevE.56.5018. arXiv: cond-mat/9707325. URL: <http://arxiv.org/abs/cond-mat/9707325> (besucht am 03.09.2023).
- [50] *Jetson Nano Developer Kit*. NVIDIA Developer. URL: <https://developer.nvidia.com/embedded/jetson-nano-developer-kit> (besucht am 15.10.2023).
- [51] *Juce*. JUCE. URL: <https://juce.com/> (besucht am 11.06.2023).
- [52] Mark Katz. *Capturing sound: how technology has changed music*. Rev. ed. OCLC: ocn610019531. Berkeley: University of California Press, 2010. 320 S. ISBN: 978-0-520-26105-1.
- [53] Chris Dongjoo Kim, Byeongchang Kim, Hyunmin Lee, Gunhee Kim. „AudioCaps: Generating Captions for Audios in The Wild“. In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. NAACL-HLT 2019. Minneapolis, Minnesota: Association for Computational Linguistics, Juni 2019, S. 119–132. DOI: 10.18653/v1/N19-1011. URL: <https://aclanthology.org/N19-1011> (besucht am 12.08.2023).
- [54] Diederik P. Kingma, Max Welling. *Auto-Encoding Variational Bayes*. 10. Dez. 2022. arXiv: 1312.6114[cs, stat]. URL: <http://arxiv.org/abs/1312.6114> (besucht am 14.08.2023).

- [55] A. Sophia Koepke, Andreea-Maria Oncescu, João F. Henriques, Zeynep Akata, Samuel Albanie. „Audio Retrieval with Natural Language Queries: A Benchmark Study“. In: *IEEE Transactions on Multimedia* 25 (2023), S. 2675–2685. issn: 1520-9210, 1941-0077. doi: 10.1109/TMM.2022.3149712. arXiv: 2112.09418[cs, eess]. URL: <http://arxiv.org/abs/2112.09418> (besucht am 08.10.2023).
- [56] John F. Kolen, Stefan C. Kremer, Hrsg. *A field guide to dynamical recurrent networks*. New York: IEEE Press, 2001. 421 S. isbn: 978-0-7803-5369-5 978-0-7908-5369-7.
- [57] Jungil Kong, Jaehyeon Kim, Jaekyoung Bae. *HiFi-GAN: Generative Adversarial Networks for Efficient and High Fidelity Speech Synthesis*. 23. Okt. 2020. arXiv: 2010.05646[cs, eess]. URL: <http://arxiv.org/abs/2010.05646> (besucht am 14.08.2023).
- [58] Qiuqiang Kong, Yin Cao, Turab Iqbal, Yuxuan Wang, Wenwu Wang, Mark D. Plumbley. *PANNs: Large-Scale Pretrained Audio Neural Networks for Audio Pattern Recognition*. 23. Aug. 2020. arXiv: 1912.10211[cs, eess]. URL: <http://arxiv.org/abs/1912.10211> (besucht am 19.08.2023).
- [59] Felix Kreuk, Gabriel Synnaeve, Adam Polyak, Uriel Singer, Alexandre Défossez, Jade Copet, Devi Parikh, Yaniv Taigman, Yossi Adi. *AudioGen: Textually Guided Audio Generation*. 5. März 2023. arXiv: 2209.15352[cs, eess]. URL: <http://arxiv.org/abs/2209.15352> (besucht am 04.08.2023).
- [60] Kundan Kumar, Rithesh Kumar, Thibault de Boissiere, Lucas Gestin, Wei Zhen Teoh, Jose Sotelo, Alexandre de Brebisson, Yoshua Bengio, Aaron Courville. *MelGAN: Generative Adversarial Networks for Conditional Waveform Synthesis*. 8. Dez. 2019. arXiv: 1910.06711[cs, eess]. URL: <http://arxiv.org/abs/1910.06711> (besucht am 12.08.2023).
- [61] Edmund Lai. *Practical digital signal processing for engineers and technicians*. OCLC: ocm51527224. London ; Burlington, MA: Newnes, 2004. 289 S. isbn: 978-0-7506-5798-3.
- [62] *librosa.mel\_frequencies — librosa 0.10.1 documentation*. URL: [https://librosa.org/doc/main/generated/librosa.mel\\_frequencies.html](https://librosa.org/doc/main/generated/librosa.mel_frequencies.html) (besucht am 09.10.2023).
- [63] Haohe Liu, Zehua Chen, Yi Yuan, Xinhao Mei, Xubo Liu, Danilo Mandic, Wenwu Wang, Mark D. Plumbley. *AudioLDM: Text-to-Audio Generation with Latent Diffusion Models*. 16. Feb. 2023. arXiv: 2301.12503[cs, eess]. URL: <http://arxiv.org/abs/2301.12503> (besucht am 19.03.2023).
- [64] Haohe Liu, Qiao Tian, Yi Yuan, Xubo Liu, Xinhao Mei, Qiuqiang Kong, Yuping Wang, Wenwu Wang, Yuxuan Wang, Mark D. Plumbley. *AudioLDM2: Learning Holistic Audio Generation with Self-supervised Pretraining*. 10. Aug. 2023. arXiv: 2308.05734[cs, eess]. URL: <http://arxiv.org/abs/2308.05734> (besucht am 05.09.2023).
- [65] Xubo Liu, Turab Iqbal, Jinzheng Zhao, Qiushi Huang, Mark D. Plumbley, Wenwu Wang. *Conditional Sound Generation Using Neural Discrete Time-Frequency Representation Learning*. 6. Okt. 2021. arXiv: 2107.09998[cs, eess]. URL: <http://arxiv.org/abs/2107.09998> (besucht am 10.08.2023).
- [66] Xubo Liu, Haohe Liu, Qiuqiang Kong, Xinhao Mei, Mark D. Plumbley, Wenwu Wang. *Simple Pooling Front-ends For Efficient Audio Classification*. 6. Mai 2023. arXiv: 2210.00943[cs, eess]. URL: <http://arxiv.org/abs/2210.00943> (besucht am 06.10.2023).

- [67] Xubo Liu, Haohe Liu, Qiuqiang Kong, Xinhao Mei, Jinzheng Zhao, Qiushi Huang, Mark D. Plumbley, Wenwu Wang. *Separate What You Describe: Language-Queried Audio Source Separation*. 28. März 2022. arXiv: 2203.15147[cs, eess]. URL: <http://arxiv.org/abs/2203.15147> (besucht am 05.09.2023).
- [68] Yinhai Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, Veselin Stoyanov. *RoBERTa: A Robustly Optimized BERT Pretraining Approach*. 26. Juli 2019. arXiv: 1907.11692[cs]. URL: <http://arxiv.org/abs/1907.11692> (besucht am 14.08.2023).
- [69] *Logic Pro for Mac*. Apple. URL: <https://www.apple.com/logic-pro/> (besucht am 14.08.2023).
- [70] Machine Learning at Berkeley. *Diffusion Models*. Berkeley, 3. Dez. 2022. URL: <https://www.youtube.com/watch?v=687zEGODmHA> (besucht am 03.09.2023).
- [71] Saeed Masoudnia, Reza Ebrahimpour. „Mixture of experts: a literature survey“. In: *Artificial Intelligence Review* 42.2 (Aug. 2014), S. 275–293. ISSN: 0269-2821, 1573-7462. doi: 10.1007/s10462-012-9338-y. URL: <http://link.springer.com/10.1007/s10462-012-9338-y> (besucht am 06.10.2023).
- [72] M. V. Mathews. „The Digital Computer as a Musical Instrument“. In: *Science* 142.3592 (1963). Publisher: American Association for the Advancement of Science, S. 553–557. ISSN: 00368075, 10959203. URL: <http://www.jstor.org/stable/1712380> (besucht am 06.06.2023).
- [73] Max Mathews, John Chowning, Curtis Roads. „Music Meets the Computer“. Computer History Museum, 14. Dez. 2004. URL: <https://www.youtube.com/watch?v=Hloic1oBfug> (besucht am 06.06.2023).
- [74] MIDI Association. *MIDI*. MIDI. URL: <https://www.midi.org/> (besucht am 28.07.2023).
- [75] Radford M. Neal. *Annealed Importance Sampling*. 4. Sep. 1998. arXiv: physics/9803008. URL: <http://arxiv.org/abs/physics/9803008> (besucht am 03.09.2023).
- [76] Alex Nichol, Prafulla Dhariwal. *Improved Denoising Diffusion Probabilistic Models*. 18. Feb. 2021. arXiv: 2102.09672[cs, stat]. URL: <http://arxiv.org/abs/2102.09672> (besucht am 04.09.2023).
- [77] Oli Larkin. *Machine Learning Audio Plug-ins with iPlug2 and ONNX Runtime*. 12. Apr. 2023. URL: [https://www.youtube.com/watch?v=t662qg12f\\_Y](https://www.youtube.com/watch?v=t662qg12f_Y) (besucht am 16.07.2023).
- [78] ONNX. ONNX. URL: <https://onnx.ai/> (besucht am 16.07.2023).
- [79] ONNX-Runtime. ONNX Runtime. URL: <https://onnxruntime.ai/> (besucht am 16.07.2023).
- [80] Aaron van den Oord, Oriol Vinyals, Koray Kavukcuoglu. *Neural Discrete Representation Learning*. 30. Mai 2018. arXiv: 1711.00937[cs]. URL: <http://arxiv.org/abs/1711.00937> (besucht am 10.08.2023).
- [81] OpenAI. *GPT-4 Technical Report*. 27. März 2023. arXiv: 2303.08774[cs]. URL: <http://arxiv.org/abs/2303.08774> (besucht am 08.09.2023).
- [82] Barry R. Parker. *Good vibrations: the physics of music*. OCLC: ocn320194527. Baltimore: Johns Hopkins University Press, 2009. 274 S. ISBN: 978-0-8018-9264-6.

- 
- [83] Pierre-Louis Suckrow. *Text-zu-spielbarem-Klang: Synthesizer basierend auf Latent-Diffusion-Technologie*. URL: <https://suckrowpierre.github.io/TtPS.github.io/> (besucht am 07. 09. 2023).
  - [84] William C. Pirkle. *Designing software synthesizer plug-ins in C++ with audio DSP*. 2nd edition. New York: Routledge, 2021. ISBN: 978-0-367-51048-0 978-0-367-51046-6.
  - [85] Patrick von Platen, Suraj Patil, Anton Lozhkov, Pedro Cuenca, Nathan Lambert, Kashif Rasul, Mishig Davaadorj, Thomas Wolf. *Diffusers: State-of-the-art diffusion models*. Version 0.12.1. original-date: 2022-05-30T16:04:02Z. 18. Okt. 2023. URL: <https://github.com/huggingface/diffusers> (besucht am 18. 10. 2023).
  - [86] Manos Plitsis, Theodoros Kouzelis, Georgios Paraskevopoulos, Vassilis Katsouros, Yannis Panagakis. *Investigating Personalization Methods in Text to Music Generation*. 20. Sep. 2023. doi: 10.48550/arXiv.2309.11140. arXiv: 2309.11140[cs,eess]. URL: <http://arxiv.org/abs/2309.11140> (besucht am 14. 10. 2023).
  - [87] *PyInstaller Manual — PyInstaller 6.0.0 documentation*. URL: <https://pyinstaller.org/en/stable/> (besucht am 11. 10. 2023).
  - [88] Qosmo. *Neutone by Qosmo*. Neutone by Qosmo. URL: <https://neutone.space/> (besucht am 28. 07. 2023).
  - [89] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, Ilya Sutskever. *Learning Transferable Visual Models From Natural Language Supervision*. 26. Feb. 2021. arXiv: 2103.00020[cs]. URL: <http://arxiv.org/abs/2103.00020> (besucht am 10. 08. 2023).
  - [90] Hannes Raffaseder. *Audiodesign*. 2., aktualisierte und erweiterte Auflage. München: Hanser, 2010. 313 S. ISBN: 978-3-446-41762-5.
  - [91] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, Peter J. Liu. *Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer*. 28. Juli 2020. arXiv: 1910.10683[cs,stat]. URL: <http://arxiv.org/abs/1910.10683> (besucht am 19. 08. 2023).
  - [92] Adam Roberts, Hyung Won Chung, Anselm Levskaya, Gaurav Mishra, James Bradbury, Daniel Andor, Sharan Narang, Brian Lester, Colin Gaffney, Afroz Mohiuddin, Curtis Hawthorne, Aitor Lewkowycz, Alex Salcianu, Marc van Zee, Jacob Austin, Sebastian Goodman, Livio Baldini Soares, Haitang Hu, Sasha Tsvyashchenko, Aakanksha Chowdhery, Jasmin Bastings, Jannis Bulian, Xavier Garcia, Jianmo Ni, Andrew Chen, Kathleen Kenealy, Jonathan H. Clark, Stephan Lee, Dan Garrette, James Lee-Thorp, Colin Raffel, Noam Shazeer, Marvin Ritter, Maarten Bosma, Alexandre Passos, Jeremy Maitin-Shepard, Noah Fiedel, Mark Omernick, Brennan Saeta, Ryan Sepassi, Alexander Spiridonov, Joshua Newlan, Andrea Gesmundo. *Scaling Up Models and Data with \$\\texttt{t5x}\\\$ and \$\\texttt{seqio}\\\$*. 31. März 2022. arXiv: 2203.17189[cs]. URL: <http://arxiv.org/abs/2203.17189> (besucht am 07. 10. 2023).
  - [93] Martin Robinson. *Getting started with JUCE: leverage the power of the JUCE framework to start developing applications*. OCLC: 862386437. Birmingham: Packt Publishing, 2013. ISBN: 978-1-4619-4968-8.

- [94] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, Björn Ommer. *High-Resolution Image Synthesis with Latent Diffusion Models*. 13. Apr. 2022. arXiv: 2112.10752[cs]. URL: <http://arxiv.org/abs/2112.10752> (besucht am 14.08.2023).
- [95] Olaf Ronneberger, Philipp Fischer, Thomas Brox. *U-Net: Convolutional Networks for Biomedical Image Segmentation*. 18. Mai 2015. arXiv: 1505.04597[cs]. URL: <http://arxiv.org/abs/1505.04597> (besucht am 06.04.2023).
- [96] Royalty-Free Sounds, FX, Presets & More | Splice. URL: <https://splice.com/> (besucht am 14.10.2023).
- [97] André Ruschkowski. *Elektronische Klänge und musikalische Entdeckungen*. 3., ergänzte Auflage 2019. Reclams Universal-Bibliothek 19613. Ditzingen: Reclam, 2019. ISBN: 978-3-15-019613-7.
- [98] Martin Russ. *Sound synthesis and sampling*. 3. ed. Amsterdam: Elsevier, Focal Press, 2009. 559 S. ISBN: 978-0-240-52105-3.
- [99] Swapnil Sayan Saha, Sandeep Singh Sandha, Mani Srivastava. „Machine Learning for Microcontroller-Class Hardware: A Review“. In: *IEEE Sensors Journal* 22.22 (15. Nov. 2022), S. 21362–21390. ISSN: 1530-437X, 1558-1748, 2379-9153. DOI: 10.1109/JSEN.2022.3210773. arXiv: 2205.14550[cs]. URL: <http://arxiv.org/abs/2205.14550> (besucht am 15.10.2023).
- [100] sampler Juce. JUCE Docs. URL: [https://docs.juce.com/master/group\\_\\_juce\\_\\_audio\\_\\_formats-sampler.html](https://docs.juce.com/master/group__juce__audio__formats-sampler.html) (besucht am 19.08.2023).
- [101] C.E. Shannon. „Communication in the Presence of Noise“. In: *Proceedings of the IRE* 37.1 (Jan. 1949), S. 10–21. ISSN: 0096-8390. DOI: 10.1109/JRPROC.1949.232969. URL: <http://ieeexplore.ieee.org/document/1697831/> (besucht am 02.07.2023).
- [102] Jascha Sohl-Dickstein, Eric A. Weiss, Niru Maheswaranathan, Surya Ganguli. *Deep Unsupervised Learning using Nonequilibrium Thermodynamics*. 18. Nov. 2015. arXiv: 1503.03585[cond-mat,q-bio,stat]. URL: <http://arxiv.org/abs/1503.03585> (besucht am 11.08.2023).
- [103] Bjarne Stroustrup. *The C++ programming language*. 3rd ed. Reading, Mass: Addison-Wesley, 1997. 910 S. ISBN: 978-0-201-88954-3.
- [104] Xu Tan, Tao Qin, Jiang Bian, Tie-Yan Liu, Yoshua Bengio. *Regeneration Learning: A Learning Paradigm for Data Generation*. 20. Jan. 2023. arXiv: 2301.08846[cs,eess]. URL: <http://arxiv.org/abs/2301.08846> (besucht am 06.09.2023).
- [105] *TensorFlow Lite API Reference*. URL: [https://www.tensorflow.org/lite/api\\_docs](https://www.tensorflow.org/lite/api_docs) (besucht am 15.10.2023).
- [106] Daniel M. Thompson. *Understanding audio: getting the most out of your project or professional recording studio*. Recording: Audio. OCLC: ocm58450656. Boston, Mass: Berklee Press, 2005. 357 S. ISBN: 978-0-634-00959-4.
- [107] K. S. Thyagarajan. *Introduction to Digital Signal Processing Using MATLAB with Application to Digital Communications*. 1st ed. 2019. Cham: Springer International Publishing : Imprint: Springer, 2019. 1 S. ISBN: 978-3-319-76029-2. DOI: 10.1007/978-3-319-76029-2.
- [108] *TorchScript — PyTorch 2.0 documentation*. URL: <https://pytorch.org/docs/stable/jit.html> (besucht am 16.07.2023).

- [109] Kinko Tsuji, Stefan C. Müller. *Physics and music: essential connections and illuminating excursions*. Cham, Switzerland: Springer Nature, 2021. ISBN: 978-3-030-68675-8.
- [110] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, Illia Polosukhin. *Attention Is All You Need*. 5. Dez. 2017. arXiv: 1706.03762[cs]. URL: <http://arxiv.org/abs/1706.03762> (besucht am 19.03.2023).
- [111] Daniel Walz. *Plugin Gui Magic*. URL: <https://foleysfinest.com/developer/pluginguimagic/> (besucht am 15.07.2023).
- [112] Harvey Elliott White, Donald H. White. *Physics and music: the science of musical sound*. OCLC: 878661301. Mineola, New York: Dover Publications, Inc., 2014. ISBN: 978-0-486-79400-6.
- [113] Yusong Wu, Ke Chen, Tianyu Zhang, Yuchen Hui, Taylor Berg-Kirkpatrick, Shlomo Dubnov. *Large-scale Contrastive Language-Audio Pretraining with Feature Fusion and Keyword-to-Caption Augmentation*. 7. Apr. 2023. arXiv: 2211.06687[cs, eess]. URL: <http://arxiv.org/abs/2211.06687> (besucht am 18.07.2023).
- [114] Xuenan Xu, Heinrich Dinkel, Mengyue Wu, Kai Yu. „A CRNN-GRU Based Reinforcement Learning Approach to Audio Captioning.“ In: *DCASE*. 2020, S. 225–229. URL: <https://myw19.github.io/AboutPageAssets/papers/xnx98-xu-dcase2020.pdf> (besucht am 08.10.2023).
- [115] Dongchao Yang, Jianwei Yu, Helin Wang, Wen Wang, Chao Weng, Yuexian Zou, Dong Yu. *Diffsound: Discrete Diffusion Model for Text-to-sound Generation*. 28. Apr. 2023. arXiv: 2207.09983[cs, eess]. URL: <http://arxiv.org/abs/2207.09983> (besucht am 04.08.2023).
- [116] Young Gale, Ford Clifford. *Hugh Le Caine*. The Canadian Encyclopedia. 16. Dez. 2013. URL: <https://www.thecanadianencyclopedia.ca/en/article/hugh-le-caine-emc> (besucht am 25.07.2023).
- [117] Sergey Zagoruyko, Nikos Komodakis. *Wide Residual Networks*. 14. Juni 2017. arXiv: 1605.07146[cs]. URL: <http://arxiv.org/abs/1605.07146> (besucht am 04.09.2023).
- [118] Neil Zeghidour, Alejandro Luebs, Ahmed Omran, Jan Skoglund, Marco Tagliasacchi. *SoundStream: An End-to-End Neural Audio Codec*. 7. Juli 2021. arXiv: 2107.03312[cs, eess]. URL: <http://arxiv.org/abs/2107.03312> (besucht am 07.08.2023).

All links were last followed on 19. Oktober 2023.



## A. Veröffentlichung

Der auf *GitHub* [34] veröffentlichte Code befindet sich im Repository <https://github.com/suckrowPierre/WaveGenSynth>. Die für diese Arbeit relevante Version ist im Branch „thesis“, <https://github.com/suckrowPierre/WaveGenSynth/tree/thesis> zu finden. Das Repository umfasst den Quellcode sowohl für das digitale Instrument als auch für den *FastApi*-Server. Zusätzlich sind ein Plugin-Installer für Mac, ausführbare Server-Dateien für die Silicon- und Intel-Architekturen auf Mac sowie eine Zip-Datei vorhanden. Letztere beinhaltet das *Python*-Skript für den Server und die benötigten *Python*-Pakete für *Conda*, welche die Ausführung des Servers ohne eine ausführbare Datei ermöglichen.

Die README-Datei bietet Anleitungen zur Installation und Ausführung des Codes.

## WaveGenSynth

WaveGenSynth is a unique synthesizer that leverages diffusion techniques to create playable sounds. It can seamlessly integrate into your musical workflow as a VST, AU, standalone application, or run directly from the source code. The digital instrument is the result of my bachelor thesis Text-to-Playable-Sound: synthesizers based on latent diffusion technology.

### Text-to-Playable-Sound: synthesizers based on latent diffusion technology

In this work, the integration and applicability of generative artificial intelligence in the field of music production was analyzed through the introduction of a digital instrument. Using selected diffusion models, users can define sounds through textual descriptions and play and manipulate them with standardized music production tools. The diffusion models used were evaluated for their suitability in the given context and modified for integration into a digital instrument. Using certain frameworks, the digital instrument was created, resulting in a user interface. This allows users to edit model and instrument-specific parameters. The analysis showed that the models used do not always adequately respond to the context of music production, resulting in unexpected sound patterns or abstract artifacts. Currently, available text-to-audio models do not provide high-quality reproduction of familiar sounds but offer opportunities for experimental applications. The implementation of a prototype of the digital instrument allows for such experiments and the exploration of innovative sound synthesis methods. However, functions are currently missing to reproduce selected areas of the generated sounds or to play them indefinitely. Nevertheless, interesting and unusual soundscapes can already be produced, which could find application in musical compositions.

### Table of Contents

- Installation
  - Binary Installations (VST/AU/Standalone)
  - Building from Source
    - \* Mac
    - \* Windows
- Usage
- Model-evaluation
- Credits

### Installation

!!! Installer and Server-Application files are currently unsigned for mac. To open right click and click on open. Double click won't work

## Binary Installations (VST/AU/Standalone)

Download the latest stable release from the releases page, and follow the instructions for your specific platform.

Both Installer and Server are available for Intel and Apple Silicon Macs.

After installation, run the server and connect to it through the plugin. Note that the server executable may exhibit unstable performance on Intel Macs. If you face issues with the server executable or prefer running it as a script, download `server_light.zip` and follow the instructions in step 4 below.

## Building from Source

If you prefer to build from source, follow these steps:

### 1. Clone the Repository:

- Clone the repository along with its submodules using the command:  
`git clone --recurse-submodules <REPOSITORY_URL>`
- If cloned normally, initialize and update the submodules with:  
`git submodule init  
git submodule update`

### 2. Load the Project:

Open the project using the provided CMake file.

### 3. Configure Run/Debug Settings:

Create a Run/Debug Configuration by selecting one of the available executables to run the project.

### 4. Set Up the Server:

- Update the environment: `conda env update --file environment.yml` (located in the FastAPI folder)
- Activate the environment: `conda activate WaveGenSynthAPI_env`
- Start the server: `uvicorn server:app --reload`

**Mac** On Mac, Xcode is required for C++ support.

**Windows** Not yet tested.

## Usage

Operating WaveGenSynth is straightforward: 1. Launch the Server and plugin. 2. Initialize a model. This step may take some time during the first run, causing the plugin to freeze momentarily. Monitor the server console until the download and setup are complete. If a timeout occurs in the plugin, hit refresh after the setup is complete. (Select ‘cuda’ if running on hardware with a Nvidia GPU, ‘mps’ if running on Apple Silicon, and ‘cpu’ as a fallback) 3. Enter a prompt in the designated field. 4. Click the “Generate” button. 5. Select the appropriate

MIDI input device. If using as a standalone click option and then select your MIDI device 6. Enjoy creating and playing sounds!

## Modelevaluation

The effect of different torch-devices, parameters and prompts on the available pre-trained models are displayed here

## Credits

WaveGenSynth was crafted using the following technologies: - **GUI Framework:** JUCE - **GUI Module:** foleys\_gui\_magic - **Diffusion Model:** AudioLDM, integrated via the pipeline and AudioLDM2, integrated via the pipeline from HuggingFace

A special thank you to the developers of these tools that made WaveGenSynth possible.

**Erklärung**

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Berlin, den 19. Oktober 2023



---

Ort, Datum, Unterschrift