PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL
ESCOLA POLITÉCNICA
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

MODELO TRANSFORMER CODIFICADOR-DECODIFICADOR PARA GERAÇÃO DE ACOMPANHAMENTO MUSICAL COM BASE EM UMA MELODIA DE ENTRADA

RENAN FRANTZ

Trabalho de Conclusão II apresentado como requisito parcial à obtenção do grau de Bacharel em Ciência da Computação na Pontifícia Universidade Católica do Rio Grande do Sul.

Orientador: Prof. Rafael Scopel Silva

Porto Alegre
2024

“What makes the difference between man and all the rest of the animal creation? Every beast that strays beside me has the same corporal necessities with myself: he is hungry, and crops the grass; he is thirsty, and drinks the stream; his thirst and hunger are appeased; he is satisfied, and sleeps; he rises again, and is hungry; he is again fed, and is at rest. I am hungry and thirsty, like him, but when thirst and hunger cease, I am not at rest.”
(Samuel Johnson)

DEDICATÓRIA

Dedico este trabalho aos meus pais, Marta e Rene, pelo amor incondicional e apoio constante; aos meus amigos, que sempre estiveram ao meu lado nos momentos de alegria e de dificuldade; ao meu orientador, pela orientação, paciência e sabedoria essenciais para a realização deste trabalho; e à música, minha fonte de inspiração e motivação, que me acompanhou em cada passo desta jornada.

MODELO TRANSFORMER CODIFICADOR-DECODIFICADOR PARA GERAÇÃO DE ACOMPANHAMENTO MUSICAL COM BASE EM UMA MELODIA DE ENTRADA

RESUMO

Neste artigo, exploramos a criação de um modelo generativo baseado em transformers para criar acompanhamentos com base em uma melodia. Primeiro, analisamos a motivação para criação deste, então, exploramos a bibliografia existente e trabalhos nos quais nos baseamos. A partir disto, vemos como o desenvolvimento deste modelo será conduzido e quais recursos serão necessários para isto. Por fim, analisamos os resultados e fazemos comparações entre os diferentes modelos treinados. Os modelos finais faltaram com performance pela limitação dos dados, mas funcionam em situações especificas.

Palavras-Chave: inteligência artificial, aprendizado de máquina, transformers, música, melodia, acompanhamentos, modelos generativos, aprendizado profundo.

ENCODER-DECODER TRANSFORMER MODEL FOR ACCOMPANIMENT GENERATION WITH A BASE MELODY AS INPUT

ABSTRACT

In this article, we explore the creation of a generative model based on transformers for creating accompaniments based on a base melody. First, we analyse the motivation for creating this tool, then, we explore the bibliography surrounding the subject and the existing work we will be basing ourselves in. With that, we will see how the development of this model will be conducted, and what resources are necessary for it. Finally, we analyze the results and make comparisons between the different trained models. The final models lack performance due to a low quantity of data, but work in specific situations

Keywords: artificial intelligence, machine learning, transformers, music, melody, accompaniments, generative models, deep learning.

LISTA DE FIGURAS

2.1 Arquitetura geral do modelo. Retirada de [Eng23] 18
2.2 Arquitetura do SingSong. Retirada de [Eng23] 19
2.3 Arquitetura geral do EnCodec. Retirada de [TVJ22] 20
4.1 Diagrama do pre-processamento 24
4.2 Diagrama da extração de tokens 25
4.3 Diagrama do treino 27
4.4 Diagrama da inferência 27
5.1 Step x Acurácia de treino para treino de bateria - tudo. 32
5.2 Step x Acurácia de validação para treino de bateria - tudo. 32
5.3 Step x Acurácia de treino para treino de bateria - tudo sem vocal. 33
5.4 Step x Acurácia de validação para treino de bateria - tudo sem vocal. 33
5.5 Step x Acurácia de treino para treino de bateria - baixo. 34
5.6 Step x Acurácia de validação para treino de bateria - baixo. 34
5.7 Step x Acurácia de treino para treino de baixo - bateria. 35
5.8 Step x Acurácia de validação para treino de baixo - bateria. 35
5.9 Step x Acurácia de treino para treino de baixo - tudo sem vocal. 36
5.10 Step x Acurácia de validação para treino de baixo - tudo sem vocal. 36
5.11 Step x Acurácia de treino para treino de baixo - tudo. 37
5.12 Step x Acurácia de validação para treino de baixo - tudo. 37

LISTA DE TABELAS

2.1 Bitrate, discriminabilidade (quanto menor melhor) e qualidade (quanto maior melhor) dos tokens semânticos e acústicos. Retirada de [BMV+23] 18
5.1 Resultados de Acurácia da Validação e Número de Acompanhamentos Válidos 30
5.2 Links para Áudio 31

LISTA DE SIGLAS

SAM – Segment Anything Model

RNNS – Recurrent Neural Networks

LSTM – Long Short-Term Memory

NNS – Neural Networks

BPTT – Backpropagation Through Time

FAD – Fréchet Audio Distance

SUMÁRIO

1 INTRODUÇÃO **11**

2 REVISÃO BIBLIOGRÁFICA **12**

2.1 REDES NEURAIS 12

2.2 REDES NEURAIS RECORRENTES 12

2.2.1 RETROPROPAGAÇÃO ATRAVÉS DO TEMPO 13

2.2.2 LONG SHORT-TERM MEMORY 13

2.3 TRANSFORMERS 14

2.3.1 MECANISMO DE ATENÇÃO 15

2.3.2 CODIFICADOR 16

2.3.3 DECODIFICADOR 16

2.4 MODELOS GENERATIVOS DE ÁUDIO 17

2.4.1 AUDIOLM 17

2.4.2 SINGSONG 18

2.4.3 ENCODEC 19

2.4.4 W2V-BERT 20

2.4.5 T5X 21

3 DEFINIÇÃO DO OBJETIVO **22**

4 METODOLOGIA **23**

4.1 COLETA E PREPARAÇÃO DOS DADOS 23

4.1.1 PRÉ-PROCESSAMENTO DOS DADOS 23

4.2 DEFINIÇÃO DO MODELO 25

4.2.1 TOKENS SEMÂNTICOS (W2V-BERT) 26

4.2.2 TOKENS ACOUSTICOS (ENCODEC) 26

4.3 TREINAMENTO DO MODELO 26

4.4 AVALIAÇÃO DO MODELO 27

4.4.1 MÉTRICA DE ACURÁCIA 28

5 RESULTADOS **30**

5.1 BATERIA COMO ENTRADA 30

5.1.1 BATERIA -> TODOS INSTRUMENTOS 31

5.1.2 BATERIA -> TUDO SEM VOCAL 31
5.1.3 BATERIA -> APENAS BAIXO 33
5.2 BAIXO COMO ENTRADA 34
5.2.1 BAIXO -> APENAS BATERIA 34
5.2.2 BAIXO -> TUDO SEM VOCAL 35
5.2.3 BAIXO -> TODOS INSTRUMENTOS 36

6 DISCUSSÃO DOS RESULTADOS 38

7 CONCLUSÃO 40

REFERÊNCIAS 41

1. INTRODUÇÃO

As tecnologias de inteligência artificial e aprendizado de máquina vêm crescendo exponencialmente nos últimos anos, como exemplos temos o surgimento do ChatGPT [cha], o DALL-E [RPG+21] e o SAM [KX23]. Estas tecnologias trazem tanto uma evolução tecnológica, como desafios para empregos vulneráveis a essa automatização, desafios que são responsáveis tanto pela extinção de empregos como mudanças drásticas que requerem re-treino dos funcionários [Zip23].

Dito isso, a inteligência artificial também permite a evolução de indústrias e o crescimento da economia através do crescimento da produtividade [Zip23]. Além disso, a IA também democratiza o acesso a serviços pessoais, como a geração de arte, música, texto ou diversos outros conteúdos que, normalmente, exigiriam uma verba que raramente está disponível para pequenos criadores [Tec23].

Observando estes fatos, gera-se a motivação de criar ferramentas de IA, como uma ajuda para os criadores humanos. Percebe-se, também, que a área de geração de áudio e música em um geral é muito menos explorada que a área de geração de texto e imagem, hoje não temos uma ferramenta equivalente em qualidade ao ChatGPT ou o DALL-E para música, o mais próximo disso seria a ferramenta AudioLM [BMV+23]. Dito isso, modelos fundacionais, treinados usando predios inteiros de servidores e datasets com terabytes de dados, estão fora do escopo deste artigo, aqui, exploramos a criação de um modelo que permita auxiliar o músico a criar uma composição, ou improvisar com acompanhamento sem precisar de uma banda.

Nossos resultados são promissores, mas deixam a desejar, conseguindo produzir acompanhamentos com boa qualidade mas com pouca frequência. O grupo conclui que isso se deve principalmente a falta de dados, que gera uma falta de generalização no modelo.

Primeiro revisaremos a bibliografia existente quanto ao tema, apos isso, definimos o objetivo. Com base na bibliografia e no objetivo definimos a metodologia, que utilizamos para produzir resultados, analiza-los, e realizar conclusões sobre o trabalho.

2. REVISÃO BIBLIOGRÁFICA

Vamos a uma explicação dos fundamentos relacionados a geração de música com modelos generativos, primeiro iremos entender a base de todas redes neurais, subsequentemente sobre Redes Neurais Recorrentes (RNNs) e um subgrupo das mesmas, as redes Long Short-Term Memory (LSTM), finalmente, vamos a explicação dos transformers, a arquitetura utilizada por diversos grandes geradores de texto e áudio, como o AudioLM[BMV+23], o ChatGPT[cha], BERT[DCLT19], Bing AI[bin] etc.

2.1 Redes Neurais

Redes Neurais são compostas por camadas de unidades de processamento, chamadas neurônios, onde cada neurônio em uma camada está conectado a todos os neurônios na camada seguinte. As camadas são categorizadas em entrada, oculta(s) e saída. Nas saídas de cada neurônio, são aplicadas funções de ativação, introduzindo não-linearidades que permitem que a rede aprenda padrões complexos. [MP43]

O treinamento envolve o ajuste dos pesos das conexões através de um processo chamado retropropagação, que minimiza o erro entre a saída prevista e a saída real ao ajustar os pesos na direção que reduz o erro.

A retropropagação consiste em duas etapas, a propagação para frente, onde os dados de entrada são passados através da rede, camada por camada, até a camada de saída. Cada neurônio calcula uma saída baseada em suas entradas, pesos e uma função de ativação. Após essa etapa, vamos para a segunda etapa, a propagação para trás.

O gradiente do erro em relação a cada peso é calculado usando a regra da cadeia do cálculo, propagando o erro de volta através da rede, da camada de saída para a camada de entrada. Isso envolve o cálculo de derivadas parciais do erro com relação a cada peso, que são então usadas para atualizar os pesos na direção que minimiza o erro. A partir dos novos pesos, atualizamos os pesos do modelo.

2.2 Redes Neurais Recorrentes

RNNs têm conexões recorrentes que permitem que a informação seja passada de uma etapa de tempo para a próxima. Isso permite que elas processem sequências de dados de forma eficaz. [She20]

Semelhante às Redes Neurais (NNs), RNNs são treinadas através de retropropagação, mas utilizam uma variação chamada retropropagação através do tempo (BPTT) devido à sua natureza temporal.

RNNs podem sofrer do problema do desaparecimento ou explosão do gradiente, tornando difícil o aprendizado de dependências de longo prazo.

2.2.1 Retropropagação Através do Tempo

A retropropagação através do tempo se diferencia da retropropagação normal pois utiliza passos de tempo. Primeiro, fazemos a propagação para frente, onde para cada passo de tempo na sequência, computamos a saída da rede com base nos dados de entrada e nos pesos atuais, mantendo a sequência de estados ocultos e saídas para cada passo de tempo.

Após ter feito isso, computamos a função de custo acumulada sobre todos os passos de tempo. Esta função de custo é uma medida do erro entre as saídas previstas pela rede e as saídas desejadas.

Tendo feito isso, iniciamos a retropropagação do último passo de tempo para o primeiro. Em cada passo de tempo, calculamos os gradientes da função de custo com relação aos pesos e aos estados ocultos. Isto é feito aplicando a regra da cadeia para obter as derivadas parciais, finalmente, acumulamos os gradientes através dos passos de tempo.

2.2.2 Long Short-Term Memory

LSTM é uma variante especial de RNN projetada para lidar com o problema do desaparecimento do gradiente e aprender dependências de longo prazo em dados sequenciais. A arquitetura LSTM é composta por unidades de memória que incluem três portas: a porta de entrada, a porta de esquecimento e a porta de saída. [She20]

A porta de entrada decide quanta informação nova será armazenada na célula de memória utilizando uma função sigmóide para determinar a quantidade de informação a ser armazenada e uma função tangente hiperbólica para criar um vetor de novos valores candidatos que poderiam ser adicionados à célula de memória.

A porta de esquecimento decide quanta informação da célula de memória anterior será mantida ou descartada utilizando uma função sigmóide para determinar a quantidade de informação a ser retida.

A porta de saída decide qual informação será enviada para fora como a saída da unidade utilizando uma função sigmóide para determinar a quantidade de informação a ser

enviada para fora e uma função tangente hiperbólica para criar uma versão regulada dos valores na célula de memória, que é então multiplicada pelo resultado da função sigmóide para produzir a saída da unidade.

A célula de memória é atualizada combinando a informação da porta de entrada e da porta de esquecimento, junto com a célula de memória anterior.

A arquitetura LSTM permite que redes neurais recorrentes aprendam e mantenham informações por longos períodos de tempo, tornando-as muito eficazes para tarefas envolvendo sequências temporais longas, como tradução automática, reconhecimento de fala e análise de série temporal.

2.3 Transformers

Os Transformers são uma arquitetura de aprendizado profundo muito mais recente que as anteriores explicadas aqui. Esta arquitetura foi introduzida em 2017 no artigo "Attention is All You Need"[VSP+17]. Os autores abordam muitas limitações das RNNs e LSTMs, especialmente no que diz respeito ao processamento de sequências longas.

De forma geral, os transformers são utilizados para modelos de linguagem, estes modelos são treinados em grandes quantidades de texto de forma não supervisionada, então não há a necessidade de "labeling". Este tipo de modelo aprende a linguagem na qual foi treinado de uma forma geral, e portanto, pode não ser tão útil para tarefas específicas. Para isso, os modelos base recebem um "fine-tuning"para a área onde irão atuar.

O processo de fine-tuning consiste em treinar um modelo em uma certa área a partir de um modelo já treinado. Isso se faz utilizando um banco de dados específico, geralmente consideravelmente menor do que o usado para treinar o modelo fundacional, com o objetivo de especializar o modelo base. Podemos observar um exemplo desta técnica na criação de modelos do ChatGPT com personalidades específicas, como um modelo com um certo padrão de linguagem, certos traços de personalidade específicos ou mesmo com conhecimento especializado em uma área específica. [ope21]

A principal inovação dos Transformers é o mecanismo de atenção, que permite que cada posição em uma sequência atenda a todas as posições em uma sequência de entrada e de saída de forma paralela, ao contrário das RNNs e LSTMs que processam sequências de forma sequencial. O mecanismo de atenção pondera diferentes partes da entrada de forma diferente, dando mais "atenção"às partes mais relevantes para a tarefa em questão, este mecanismo é explorado com mais profundidade na subseção a seguir.

A arquitetura é dividida em uma parte de codificador, que processa a entrada, e uma parte de decodificador, que gera a saída. Ambas as partes são compostas por

várias camadas de mecanismos de atenção e redes neurais feed-forward, permitindo que aprendam representações complexas e hierárquicas dos dados.

Como os Transformers, devido à sua arquitetura de autoatenção, não têm uma noção inerente de ordem ou posição, informações de posicionamento são adicionadas às entradas para que o modelo possa fazer uso da ordem das sequências. As informações de posicionamento são de extrema importância, como exemplo, podemos observar motivos dessa importância no contexto de linguagem natural:

1. **Contextualização da Sequência**: Em linguagem natural, a posição das palavras em uma frase pode alterar significativamente o significado. Por exemplo, “gato morde cachorro” tem um significado completamente diferente de “cachorro morde gato”. As informações de posição permitem que o modelo entenda essas nuances, melhorando sua capacidade de interpretar e gerar texto corretamente.

2. **Relações entre Palavras**: A posição das palavras ajuda o modelo a entender relações e dependências gramaticais. Por exemplo, em muitas línguas, adjetivos seguem substantivos específicos, e verbos podem concordar em número e pessoa com seus sujeitos. As informações de posicionamento ajudam o modelo a capturar essas regras gramaticais.

3. **Modelagem de Coerência e Coesão**: Em textos mais longos, a posição das frases e parágrafos pode ser crucial para entender a narrativa ou o argumento de um texto. A informação posicional ajuda o modelo a manter a coerência e a coesão ao longo de textos extensos.

2.3.1 Mecanismo de Atenção

As camadas de mecanismo de atenção dizem para o modelo para prestar atenção em certas palavras específicas dentre as que foram passadas quando estiver trabalhando com a representação de cada palavra

Como exemplo, considere a tarefa de traduzir textos do inglês para o português. Dado a entrada "You like this paper", um modelo de tradução precisará também prestar atenção na palavra adjacente "You"para obter a tradução correta para a palavra "like", porque em português o verbo "gostar"é conjugado de maneira diferente dependendo do sujeito. O restante da frase, no entanto, não é útil para a tradução dessa palavra. Da mesma forma, ao traduzir "this", o modelo também precisará prestar atenção na palavra "paper", porque "este"se traduz de maneira diferente dependendo se o substantivo associado é masculino ou feminino. Novamente, as outras palavras na frase não importaram para a tradução de "this". Com frases mais complexas (e regras gramaticais mais complexas), o modelo pre-

cisaria prestar atenção especial às palavras que podem aparecer mais distantes na frase para traduzir corretamente cada palavra.

Este conceito se aplica a qualquer tarefa de linguagem natural, uma palavra por si só deixa muita informação de fora, pois seu contexto a afeta fortemente, tanto em quesito de gramática como de significado.

2.3.2 Codificador

O codificador recebe uma entrada e cria uma representação desta entrada, esta representação geralmente codifica as características relevantes da entrada para o modelo. O modelo será otimizado para compreender a entrada.

Podemos criar modelos de tipo apenas codificador. Estes modelos são úteis para classificar/entender a entrada, como por exemplo classificação de frases e reconhecimento de objetos.

Em modelos tipo apenas codificador, em cada estágio, as camadas de atenção podem acessar todas as palavras na frase inicial. Esses modelos são geralmente caracterizados como tendo atenção bi-direcional, e podem ser chamados de modelos auto-codificadores.

O treino destes modelos geralmente envolve corromper uma parte de uma frase e dar a tarefa de reconstruir a frase para o modelo.

2.3.3 Decodificador

O decodificador usa a representação do codificador junto com outras entradas para gerar o resultado desejado. Este tipo de modelo é otimizado para gerar saídas.

Assim como com modelos tipo apenas codificadores, também podemos criar modelos tipo apenas decodificadores. Estes modelos são úteis para geração de texto/conteúdo.

Em modelos tipo apenas decodificadores, em cada estágio, para cada palavra o modelo pode apenas acessar as palavras antes desta na frase. Estes modelos são chamados de auto-regressivos. O treino destes modelos geralmente envolve predizer a próxima palavra em uma frase.

2.4 Modelos Generativos de Áudio

Os avanços recentes em modelos generativos têm visto uma expansão para além do processamento de texto, entrando na fronteira do áudio e da música. Nesta seção, introduzimos dois modelos representativos dessa nova onda: o AudioLM e o SingSong.

2.4.1 AudioLM

O AudioLM [BMV+23] é um modelo de Transformer que emprega pré-treinamento generativo para capturar uma ampla gama de características de áudio. O AudioLM é uma abordagem para geração de áudio que visa alcançar alta qualidade com consistência a longo prazo. O método funciona mapeando o áudio de entrada em uma sequência de tokens discretos e aborda a geração de áudio como uma tarefa de modelagem de linguagem nesse espaço de representação.

Um grande problema com modelos de aprendizado de máquina que precisam processar áudio vem da grande quantidade de dados incluídas em arquivos de áudio, logicamente, queremos poder gerar áudio de alta qualidade, porém isso gera uma limitação para o modelo e o custo computacional, porém se usarmos detalhes de menos, teremos um resultado de baixa qualidade. Para resolver estes interesses conflitantes, o AudioLM usa uma abordagem híbrida de tokens semânticos (baixo detalhe) e acústicos (alto detalhe), com este esquema, os tokens semânticos permitem a coerência estrutural (como o mesmo gênero e ritmo ao longo de uma música) e os tokens acústicos permitem a geração de áudio de alta qualidade.

Para a geração dos tokens acústicos, o AudioLM utiliza o codec baseado em rede neural SoundStream[Z+21], o SoundStream utiliza um codificador convolucional para mapear o áudio de entrada para uma representação com um sampling rate significativamente menor que o original. Feito isso, o decodificador mapeia a representação discreta para áudio real, isso permite que o modelo tenha um custo computacional muito menor, ainda assim gerando áudio de alta qualidade.

Junto ao SoundStream , utiliza-se o w2v-BERT, o w2v-BERT é uma rede neural que gera uma representação de uma faixa de áudio (como por exemplo áudio para texto), para o AudioLM, utiliza-se uma camada intermediária onde transforma-se o áudio em tokens semânticos.

Ambas as representações são avaliadas quanto a qualidade da reconstrução de áudio a partir delas utilizando o decoder do SoundStream e comparando a saída do mesmo com o áudio original. A comparação é feita utilizando a pontuação ViSQOL [CLS+20], um proxy computacional para calcular a similaridade entre um áudio referência e sua reconstru-

ção. A discriminabilidade fonética é calculada através do erro ABX, uma métrica baseada em distância que considera um grupo de fonemas trigramas que só são diferentes no seu fonema central (por exemplo "casa"x "cena"). O erro ABX calcula quão frequentemente uma instância de X "casa"está próxima de outra instância B "cena"do que de outra instância do mesmo trigrama A "casa". Podemos observar os valores associados com estas representações na tabela 2.1

Tokenization	Bitrate	Phonetic discriminability within/across (↓)	Reconstruction quality (↑)
Semantic (w2v-BERT)	250 bps 6000 bps	6.7 / 7.6 5.6 / 6.2	1.1 1.4
Acoustic (SoundStream)	2000 bps 6000 bps	22.4 / 28.7 17.8 / 26.6	3.3 3.9

Tabela 2.1 – Bitrate, discriminabilidade (quanto menor melhor) e qualidade (quanto maior melhor) dos tokens semânticos e acústicos. Retirada de [BMV+23]

Podemos observar que utilizando ambas as representações podemos garantir consistência ao longo prazo com os tokens semânticos e detalhes acústicos e qualidade de áudio através dos tokens acústicos. O modelo AudioLM utiliza uma abordagem hierárquica, primeiro modelando os tokens semânticos para toda a sequência, e então utilizando estes como condicionamento para inferir os tokens acústicos. Isso resulta em uma arquitetura de três etapas como ilustrado na figura 2.1. Em todos os estágios um Transformer apenas decodificador é treinado para inferir o próximo token dados todos os tokens anteriores do estágio atual.

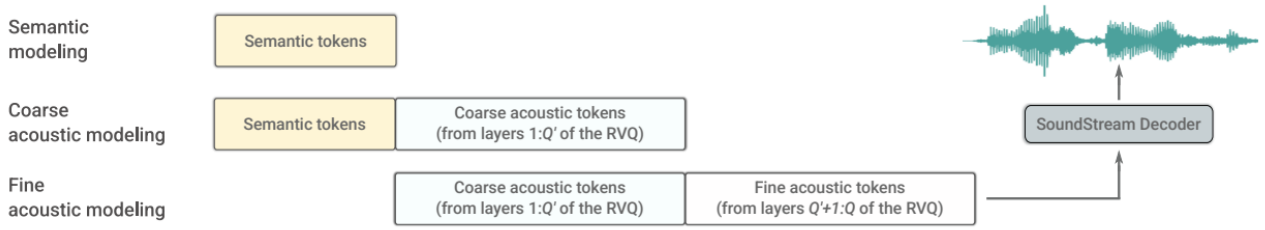

Figura 2.1 – Arquitetura geral do modelo. Retirada de [Eng23]

2.4.2 SingSong

O SingSong [Eng23] adapta o AudioLM para criar acompanhamentos musicais a partir de uma única faixa de áudio vocal. Este processo enfrenta o desafio de necessitar que os componentes da música—como vocais, guitarra, baixo e bateria—estejam separados em faixas individuais, algo típico apenas de ambientes de estúdio. Para superar isso, o SingSong emprega tecnologias avançadas de separação de áudio, como o MDX-Net [CLS+20], um desenvolvimento recente no campo da inteligência artificial.

Além disso, também existe a necessidade de modificar o AudioLM para que ao invés de gerar áudio subsequente, gere áudio acompanhando a entrada original, para isso, primeiro o SingSong codifica os vocais em tokens semânticos com o w2v-BERT (assim como o AudioLM), estes tokens semânticos são então utilizados como a entrada para o modelo codificador-decodificador, que devolve os tokens semânticos e acústicos do acompanhamento. Estes tokens semânticos e acústicos são então a saída do modelo, que tem o seu loss (diferença do gabarito) calculada em relação ao acompanhamento original da música, que foi codificado em tokens semânticos (com o w2v-BERT) e acústicos (com o SoundStream). Para gerar o output final, a saída do modelo deve ser passada pelo decodificador do SoundStream, assim como é feito com o AudioLM. Podemos observar está arquitetura na figura 2.2

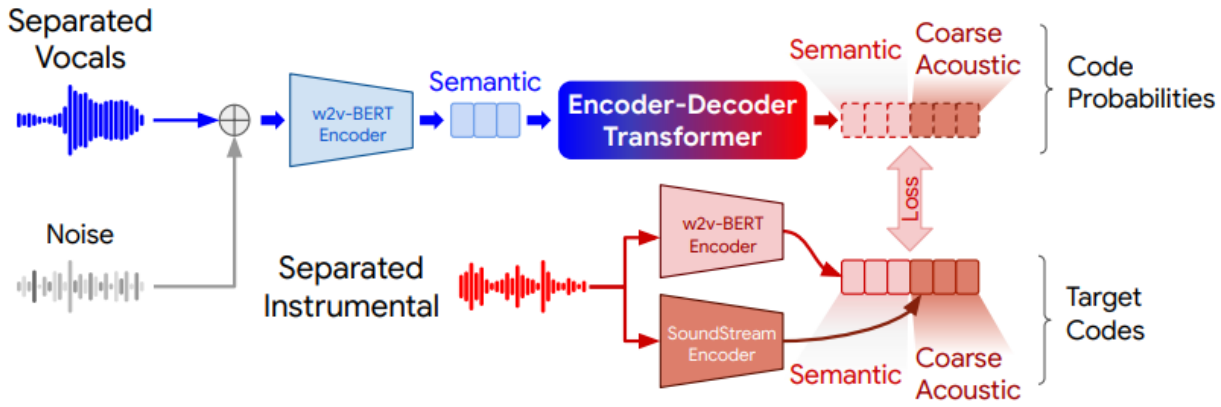

Figura 2.2 – Arquitetura do SingSong. Retirada de [Eng23]

2.4.3 EnCodec

Como dito antes, a arquitetura original do SingSong utiliza o Soundstream, dito isso, para nossa implementação optamos pelo EnCodec [TVJ22] por dois principais fatores: 1. Implementação oficial em código aberto e modelos pre-treinados e 2. Implementação e arquitetura mais recentes.[Res23] Além de maior facilidade na implementação e reprodutibilidade do projeto, temos uma qualidade superior no quesito codec.

O EnCodec é um codec de áudio neural de alta fidelidade projetado para operação em tempo real. Ele se baseia em uma arquitetura de codificador-decodificador que processa áudio de forma contínua, utilizando um espaço latente quantizado, treinado de maneira end-to-end. O modelo foi desenvolvido para simplificar e acelerar o treinamento através de um único adversário espectral multi-escala. Essa abordagem permite que o EnCodec entregue amostras de áudio de alta qualidade em diferentes taxas de amostragem e larguras de banda. Além disso, o modelo utiliza uma combinação de perdas objetivas e perceptivas

para otimizar tanto a precisão técnica quanto a qualidade perceptiva do áudio comprimido, fazendo uso de discriminadores no processo de treinamento para aprimorar a fidelidade do áudio gerado. Podemos observar visualmente a arquitetura do EnCodec na figura 2.3

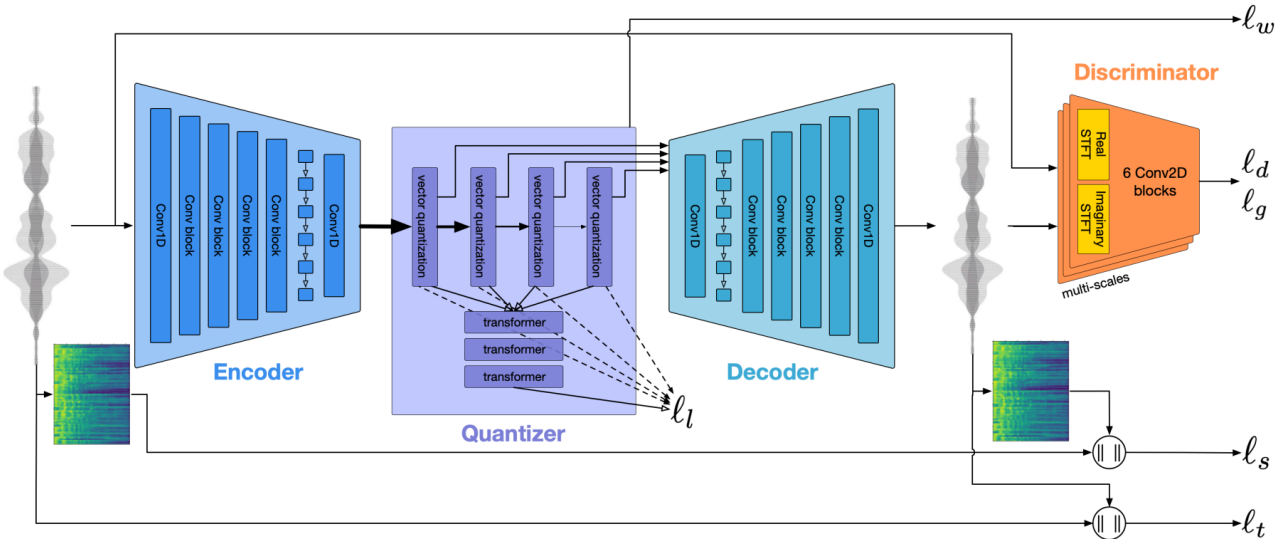

Figura 2.3 – Arquitetura geral do EnCodec. Retirada de [TVJ22]

2.4.4 w2v-Bert

O w2v-BERT [CZG21] inclui um codificador de características que utiliza camadas convolucionais para realizar subsample e transformar a entrada acústica em representações de áudio. Estas representações são então processadas por um módulo contrastivo que as discretiza em unidades de fala representativas através de um mecanismo de quantização. Este módulo gera vetores de contexto a partir das posições mascaradas e vetores quantizados das posições não mascaradas. Estes vetores são utilizados juntos para resolver tarefas contrastivas, visando otimizar a capacidade do modelo de distinguir entre diferentes unidades de áudio.

O módulo de previsão mascarada subsequente utiliza esses vetores de contexto para gerar representações de fala contextualizadas de alto nível, que são usadas para prever os identificadores de tokens correspondentes através de uma camada softmax. Esse processo ajuda o modelo a aprender representações significativas de áudio sem supervisão direta, apenas a partir de dados de fala não rotulados.

Originalmente, o modelo é feito para adquirir representações de fala para modelos de linguagem, mas através do uso de uma de suas camadas intermediárias se torna pos-

sível utilizá-lo para reconhecer a estrutura semântica da música. A estrutura semântica se refere às regras gerais da música, como o ritmo, a chave, o "sentimento"etc.

2.4.5 T5x

O T5 [RSR+20] é baseado na arquitetura do Transformer, que é amplamente utilizada devido à sua eficiência em aprender dependências de longo alcance e sua capacidade de ser paralelizada. O modelo utiliza blocos de codificadores e decodificadores: os codificadores processam a entrada de texto e os decodificadores geram a saída de texto. Esta arquitetura é semelhante à usada em modelos como BERT (codificador apenas) e GPT (decodificador apenas), mas o T5 incorpora ambas as partes, permitindo-lhe realizar uma gama mais ampla de tarefas.

No nosso caso, utilizamos o T5 com tokens diretos, ou seja um vocabulário "pass-through", para a previsão de tokens semânticos e acústicos do acompanhamento com base nos tokens da entrada. Utilizamos a implementação T5x [RCL+22], uma implementação em JAX e FLAX atualizada e direcionada ao uso com TPUs [Goo24b], o que permite uso prático da arquitetura sem muita dificuldade.

3. DEFINIÇÃO DO OBJETIVO

Este capítulo delineia o objetivo principal desta pesquisa. Tendo em vista os fundamentos teóricos e as tecnologias discutidas nos capítulos anteriores, o objetivo principal é desenvolver e implementar um sistema baseado em modelos de aprendizado profundo, especificamente usando redes neurais, para a geração automática de acompanhamentos musicais.

Objetivo Principal: Desenvolver uma rede neural que seja capaz de gerar automaticamente acompanhamentos musicais para uma melodia principal fornecida como entrada.

Sub-objetivos:

1. **Treinamento da Arquitetura SingSong:** Modificar e treinar a arquitetura do SingSong para aceitar diversas entradas instrumentais, tais como bateria e baixo, expandindo sua capacidade de geração musical além das faixas vocais.

2. **Geração de Acompanhamento Instrumental:** Implementar o sistema para que ele possa gerar acompanhamentos sem vocais, focando na coerência e qualidade musical em harmonia com a melodia principal.

3. **Análise de Desempenho:** Avaliar a qualidade dos acompanhamentos gerados, utilizando métricas objetivas e feedback de ouvintes humanos.

No próximo capítulo, detalharemos a metodologia que será aplicada para alcançar estes objetivos, incluindo as técnicas de pré-processamento de dados, a configuração do treinamento da rede neural e os critérios de avaliação dos resultados.

4. METODOLOGIA

Neste capítulo, será explicada a metodologia utilizada para realizar o desenvolvimento do modelo.

4.1 Coleta e Preparação dos Dados

Para treinar o modelo planejado, são necessários dados de áudio diversificados, incluindo diferentes instrumentos como guitarra, bateria, baixo, entre outros. Esses dados devem cobrir uma ampla gama de gêneros musicais para garantir a versatilidade do modelo.

Por estes motivos o banco de dados a ser utilizado será o MedleyDB [BST+14], o qual está disponível para acesso quando requisitado. O MedleyDB possui diversas faixas de músicas com as faixas de áudio de cada instrumento já devidamente separadas e anotadas, providenciando uma imensa facilidade no seu uso.

A estrutura de diretórios do MedleyDB é organizada da seguinte forma: no nível superior, encontramos a pasta "MedleyDB", subdividida em "Metadata"e "V1". Dentro de "V1", cada pasta leva o nome de uma música e contém um subdiretório chamado "Nome_da_musica_STEMS", onde cada faixa instrumental está separada em arquivos .wav individuais. A pasta metadata foi retirada do repositório do dataset e incluída na pasta do mesmo [B+24].

4.1.1 Pré-processamento dos dados

Inicialmente, todos os dados do MedleyDB são submetidos a um processo de compressão e normalização do sample rate para 16000 Hz. Este passo é fundamental para manter a consistência do input para os modelos de aprendizado subsequente, garantindo que todas as amostras de áudio estejam no mesmo formato e qualidade.

Para cada música, existe um arquivo metadata em formato .yaml correspondente na pasta metadata. Este arquivo possui as informações da organização das STEMS (ramos em inglês), da música. Cada STEM representa um instrumento, e através da informação de qual STEM representa qual instrumento podemos realizar a separação destes para o treino.

O processo de leitura inicia com a carga dos metadados utilizando a função load_metadata, que abre e lê o arquivo YAML correspondente. A partir destes metadados, podemos identificar especificamente quais arquivos de áudio são necessários para a análise ou processamento posterior. Com os metadados carregados, utilizamos a função find_input_stem para localizar as faixas específicas baseadas no nome do instrumento desejado, retornando o

nome do arquivo correspondente. A função mix_audio_stems é empregada para combinar diversas faixas de áudio numa única faixa. Esta funcionalidade é necessária para criar o label desejado. Este processo esta representado na figura 4.1.

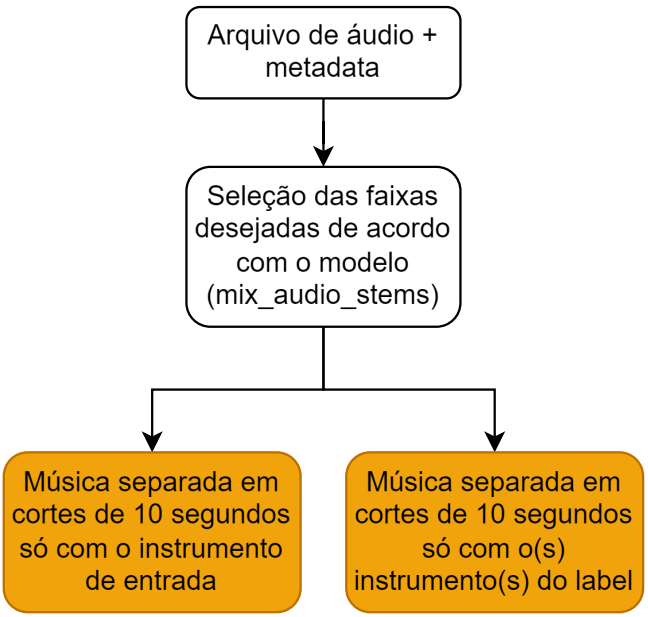

Figura 4.1 – Diagrama do pre-processamento

Tendo a entrada e a saída, calculamos os tokens acústicos com o EnCodec com 4 dimensões para a saída e os tokens semânticos com o w2v-BERT para ambas, e os guardamos em arquivos texto, onde primeiro inserimos os tokens semânticos e então os acústicos "flattened". Por fim, devido a limitação de dados, separamos o dataset em uma distribuição restrita para testes 90-5-5 (90% para treino, 5% para validação e 5% para teste). Este processo esta representado na figura 4.2.

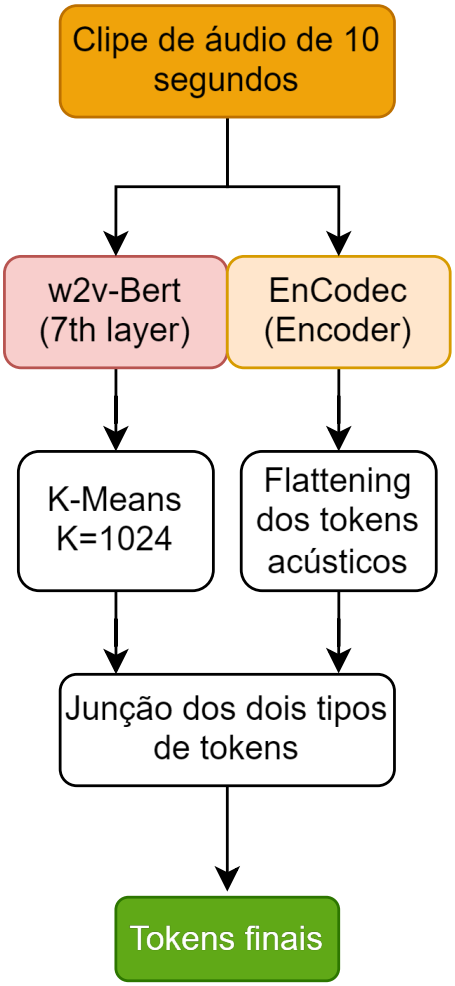

Figura 4.2 – Diagrama da extração de tokens

4.2 Definição do Modelo

O modelo será baseado na arquitetura do SingSong, portanto, a entrada será o instrumento desejado. Esta entrada será codificada pelo w2v-BERT em tokens semânticos, os quais serão utilizados como entrada para o modelo transformer T5. A saída deste modelo serão os tokens semânticos e acústicos do acompanhamento, os quais serão comparados com os tokens semânticos a acústicos do instrumental original.

Para podermos ouvir a saída, os tokens acústicos serão decodificados em música através do EnCodec. O codificador originalmente utilizado pelo SingSong é o Soundstream, porém, devido ao Soundstream não possuir implementação oficial, nem pesos pré-treinados, o grupo optou por utilizar o EnCodec como alternativa, uma arquitetura mais recente e pré-treinada de codec neural.

Por fim, modificamos a arquitetura padrão T5 para forçar a estrutura da saída, para tal, modificamos a etapa de decodificação, mais especificamente o sampling do modelo, para que os tokens escolhidos sempre sejam válidos para sua posição.

Por restrições de hardware, o modelo será treinado em clips de 10 segundos de áudio. Um diagrama do processo simplificado pode ser observado na figura 4.2.

4.2.1 Tokens Semânticos (w2v-BERT)

Para extrair tokens semânticos usando o w2v-BERT 2.0, utilizamos a função extract_layer_embeddings, que processa a onda sonora e converte-a em embeddings representativos de um determinado nível da arquitetura do modelo. Os embeddings podem ser extraídos de uma camada específica do modelo, conforme especificado pelo usuário. A camada escolhida foi a camada 7, de acordo com o que foi testado e feito pelos autores do SingSong.

Após termos os embeddings, aplicamos o K-means com K=1024 nestes embeddings (todos desse instrumento), os índices dos clusters são utilizados como tokens semânticos. Esta etapa é essencial para a redução de dimensionalidade, pois seria inviável utilizar as embeddings que possuem dimensionalidade de 1024. temos uma frequência de 50 tokens por segundo, sendo assim o total de tokens semânticos para as entradas de 500. Um diagrama do processo simplificado pode ser observado na figura 4.2.

4.2.2 Tokens Acousticos (EnCodec)

Utilizamos o EnCodec com a configuração de 3kbps, assim gerando 4 dimensões para cada token, com uma frequência de 75 tokens por segundo, tendo assim um total de 300 valores por segundo. A escolha de 3kbps se faz essencial para que o modelo possa ser treinado no hardware disponível.

4.3 Treinamento do Modelo

Utilizamos um tamanho de vocabulário total de 5120: 1024 tokens semânticos, 1024 tokens acústicos por dimensão (4096 no total).

Tendo os tokens semânticos e acústicos, podemos treinar o modelo. As configurações iniciais de treino serão as padrões do modelo T5x 1.1 [Goo24c], com a modificação de adicionar dropout de 0.01. O treinamento de modelos T5x é feito através de Python, TensorFlow e arquivos gin (arquivos utilizados para configuração de modelos, hiperparâmetros e de rotinas de treinamento com Flax e Jax), definimos um gin para o modelo em si modificado para o nosso vocabulário, um arquivo gin para o treinamento em si onde definimos o nú-

mero de passos (45000), o dropout (0.01), o batch_size (32) e o feature_length da entrada e saída (512 para entrada (tokens semânticos) e 3512 para saída (tokens acústicos)).

O T5x é implementado para ser executado em TPUs (Tensor Processing Units), por isso estes serão utilizados com o acesso gratuito dado a pesquisadores através do TPU Research Cloud [Goo24a]. Além disso, o feature_length necessário para a saída dos tokens acústicos exige uma quantidade de memória extremamente elevada.

Definimos uma máquina TPU on-demand V4-8 (4 chips V4, 8 tensorcores). Podemos observar um diagrama do processo de treino na figura 4.3, e um diagrama de inferência na figura 4.4.

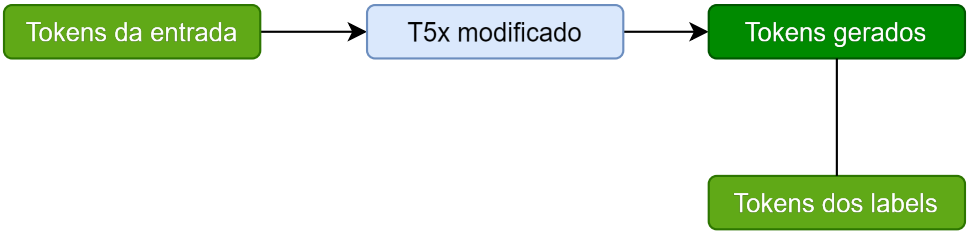

Figura 4.3 – Diagrama do treino

Figura 4.4 – Diagrama da inferência

4.4 Avaliação do Modelo

Definir objetivamente algo como certo ou errado na musica não é algo simples, e depende muito do contexto e do estilo desejado, exemplos disto podem ser encontrados facilmente em diversas músicas, por exemplo, são muito comuns no jazz, onde os músicos sempre buscam subverter as regras clássicas deliberadamente utilizados notas "erradas"e escrevendo por cima delas [Dav88]. Portanto, as métricas que utilizamos são de distância do acompanhamento original do banco de dados e avaliação qualitativa do grupo.

Para avaliar o modelo, primeiro separaremos o dataset em partes de treinamento, a qual será usada para treinar o modelo, de validação, a qual será usada para testar a performance do modelo e ajustar seus hiperparâmetros, e de teste, que será utilizada para saber a precisão final do modelo, em uma base não previamente utilizada.

Com esta separação clássica da base de dados, podemos avaliar a performance do modelo durante o treino, podendo assim ajustar seus hiperparâmetros e seu treinamento, e após o treino, podendo assim saber qual sua performance em casos no mundo real.

Na música, não existe uma única resposta para harmonizar uma melodia, mas aqui tentamos criar um modelo que gere uma resposta válida, de acordo com o banco de dados utilizado para o treino. No nosso caso, treinamos em um banco de dados com diversos estilos e gêneros diferentes, não focando em nenhum gênero especifico, com a ideia de formar a base para fine-tuning.

A eficácia do modelo na geração de acompanhamentos musicais é otimizada utilizando a função de perda de entropia cruzada, uma métrica padrão para problemas de classificação e geração de sequências. A entropia cruzada mede a divergência entre as distribuições de probabilidade previstas pelo modelo e as distribuições reais dos dados de treinamento.

Definição da Entropia Cruzada: A função de perda de entropia cruzada é definida como a média negativa do logaritmo das probabilidades preditas para a classe verdadeira. Matematicamente, é expressa pela seguinte equação:

$$L(y, \hat{y}) = -\sum_{i} y_i \log(\hat{y}_i)$$

onde y representa o vetor de etiquetas verdadeiras em forma de one-hot encoding, e $\hat{y}$ são as probabilidades preditas pelo modelo para cada classe.

4.4.1 Métrica de Acurácia

Para quantificar o desempenho do modelo na tarefa de geração de acompanhamentos musicais, utilizamos a acurácia como métrica principal. A acurácia é calculada como a proporção de predições corretas (tanto positivas quanto negativas) em relação ao total de casos avaliados.

Equação da Acurácia: A acurácia é definida pela seguinte equação:

$$\text{Acurácia} = \frac{\text{Número de predições corretas}}{\text{Número total de predições}} \times 100$$

onde o "Número de predições corretas"é a contagem de vezes em que o modelo gerou tokens que correspondem exatamente aos tokens nos arquivos de referência (labels), e o "Número total de predições"é o número total de tokens gerados pelo modelo.

Implementação Computacional: Utilizamos a função `accuracy_score` da biblioteca `sklearn.metrics` para calcular a acurácia, conforme o seguinte código em Python:

```
from sklearn.metrics import accuracy_score
accuracy = 100 * accuracy_score(targets, predictions)
```

Componentes da Métrica:

- *Targets:* São os tokens verdadeiros, extraídos dos arquivos de texto que representam as etiquetas corretas para cada amostra de áudio processada.
- *Predictions:* São os tokens gerados pelo modelo, que devem ser comparados com os tokens de *targets* para determinar a precisão das predições.

Análise Qualitativa Complementar: Além da acurácia, realizamos uma análise qualitativa, classificando manualmente os áudios gerados como 'Música', 'Silêncio' ou 'Barulho'. Essa análise ajuda a entender melhor o contexto das predições do modelo e oferece insights sobre os casos em que o modelo falha em produzir um acompanhamento musical coerente, mesmo quando os tokens numéricos correspondem, ou quando o modelo sucede em produzir um acompanhamento musical coerente, mas diferente do original.

5. RESULTADOS

Treinamos diferentes modelos da arquitetura, com graus de complexidade diferentes, para testar as capacidades do modelo com este banco de dados específico. Utilizamos a bateria e o baixo como base (cada um em modelos diferentes), devido a sua inclusão na grande maioria das músicas do banco de dados (escolher um instrumento pouco usado diminui os dados disponíveis para treino).

Utilizamos 108 arquivos do dataset de teste para averiguar os resultados, a maioria destes trechos de áudio se parecem apenas com barulho, além de rodarmos o algoritmo quantitativo de avaliação, também ouvimos os trechos de cada modelo, e documentamos os que são música, e não barulho ou silêncio.

A escolha do instrumento de entrada se deu principalmente pela quantidade de músicas que contém o mesmo, a bateria sendo o instrumento com mais dados, foi o primeiro instrumento de entrada, seguido deste utilizamos o baixo como segundo instrumento de entrada.

Cada modelo treinado sera explorado em secções neste capitulo, a tabela comparativa 5.1 mostra os resultados numéricos de todos os modelos.

Tabela 5.1 – Resultados de Acurácia da Validação e Número de Acompanhamentos Válidos

Configuração do Modelo	**Acurácia de Validação Final**	**Nº de Acompanhamentos Válidos**
Bateria -> todos instrumentos	0.0487	5
Bateria -> tudo sem vocal	0.0580	17
Bateria -> apenas baixo	0.2301	11
Baixo -> apenas bateria	0.2040	15
Baixo -> tudo sem vocal	0.0598	7
Baixo -> todos instrumentos	0.0512	20

5.1 Bateria como entrada

Como dito anteriormente, o primeiro grupo de modelos foi treinado utilizando a bateria como entrada, aqui, desenvolvemos 3 modelos diferentes, cada um com um grau de complexidade da saída diferente. Primeiro, treinamos o modelo mais complexo, que consiste na bateria como entrada e todos os outros instrumentos como saída, depois, todos outros instrumentos menos a voz, e por fim, apenas o baixo.

5.1.1 Bateria -> todos instrumentos

O primeiro modelo treinado utiliza a bateria como entrada e responde com todos outros instrumentos, este modelo foi o que mais gerou dados com barulho, e teve muita dificuldade em gerar acompanhamentos válidos, isso deve-se principalmente a variabilidade da voz como instrumento, o que gera uma dificuldade para o modelo aprender como reproduzir o mesmo.

Das 108 entradas, 5 tiveram um acompanhamento válido gerado (acompanhamento valido fica aqui definido como um clipe de áudio sonoramente agradável e consistente, com sons claramente definidos de instrumentos e uma melodia regrada), dentre estas 1 foi surpreendente pois conseguiu gerar um acompanhamento vocal coerente e que pode-se discernir as palavras, podemos ouvir estes resultados na linha 1 da tabela 5.2. Os outros acompanhamentos válidos podem ser observados na linha 2 da tabela 5.2, um exemplo do barulho gerado nas saídas inválidas pode ser visto na linha 3 da tabela 5.2. Podemos observar a acurácia do modelo durante o treino para o set de treino no gráfico 5.1 e para o set de validação no gráfico 5.2.

Tabela 5.2 – Links para Áudio

Titulo	**Link para áudio**
Bateria > Todos instrumentos 1	IA, Original, Entrada, Juncão IA, Juncão original
Bateria > Todos instrumentos 2	IA, Original, Entrada, Juncão IA, Juncão original
Exemplo de saída de barulho	IA, Original, Entrada
Bateria > Sem voz melodia silencio	IA, Original, Entrada
Bateria > Sem voz melodia repetida 1	IA, Original, Entrada, Juncão IA, Juncão original
Bateria > Sem voz melodia repetida 2	IA, Original, Entrada, Juncão IA, Juncão original
Bateria > Sem voz melodia única	IA, Original, Entrada, Juncão IA, Juncão original
Bateria > Baixo melodia repetida	IA, Original, Entrada, Juncão IA, Juncão original
Bateria > Baixo melodia única	IA, Original, Entrada, Juncão IA, Juncão original
Baixo > Bateria melodia única 1	IA, Original, Entrada, Juncão IA, Juncão original
Baixo > Bateria melodia única 2	IA, Original, Entrada, Juncão IA, Juncão original
Baixo > Sem voz exemplo de invalida	IA, Original, Entrada, Juncão IA, Juncão original
Baixo > Sem voz exemplo valido	IA, Original, Entrada, Juncão IA, Juncão original
Baixo > Todos instrumentos repetida	IA, Original, Entrada, Juncão IA, Juncão original
Baixo > Todos instrumentos 1	IA, Original, Entrada, Juncão IA, Juncão original
Baixo > Todos instrumentos 2	IA, Original, Entrada, Juncão IA, Juncão original

5.1.2 Bateria -> tudo sem vocal

O segundo modelo utiliza a mesma entrada mas corta o vocal de todas as músicas, este modelo também gerou muito barulho, mas teve mais sucesso que o anterior,

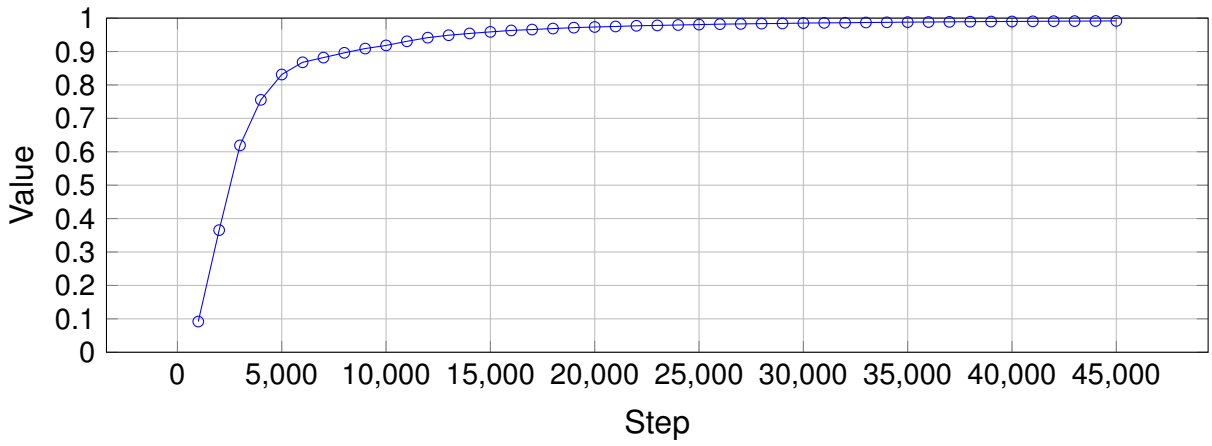

Figura 5.1 – Step x Acurácia de treino para treino de bateria - tudo.

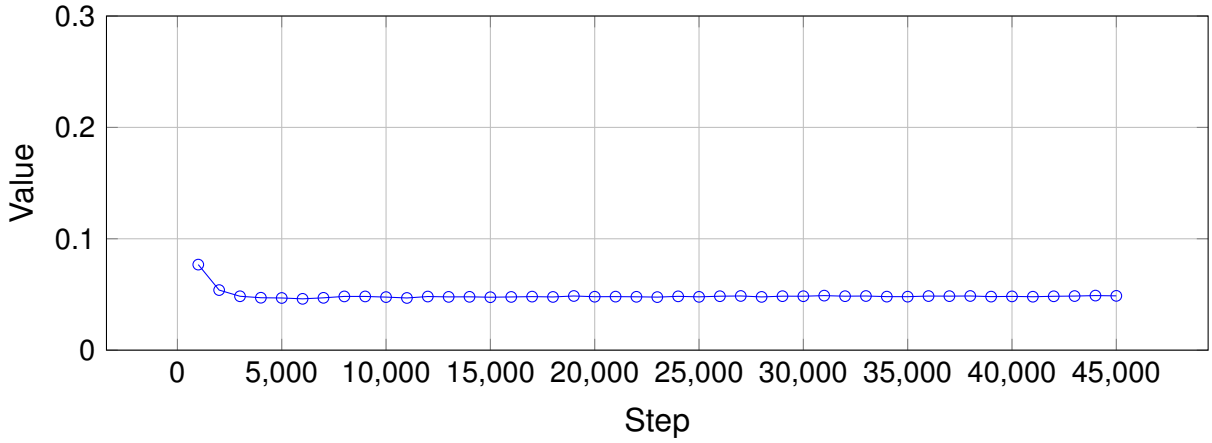

Figura 5.2 – Step x Acurácia de validação para treino de bateria - tudo.

conseguindo gerar 17 acompanhamentos válidos dentre as 108 entradas. Pode-se perceber também que o modelo gerou diversas melodias similares ou iguais com entradas diferentes, tendo 7 saídas com melodias únicas, e outras 2 que se repetiram, uma delas em 3 saídas, e a outra em 7 saídas.

A melodia com maior repetição é, na verdade, uma melodia derivada do silêncio da entrada, temos um exemplo na linha 4 da tabela 5.2. A segunda melodia repetida possui um pouco de variação entre ela, como se fossem partes separadas do mesmo acompanhamento se manifestando em diferentes partes da música, vale ressaltar que todas instancias desta melodia vem de trechos diferentes da mesma música, podemos observar um exemplo da mesma (sua versão mais completa) na linha 5 da tabela 5.2 e outra versão menos completa na linha 6. Também podemos observar um exemplo único na linha 7.

Além disso, podemos observar a acurácia do modelo durante o treino para o set de treino no gráfico 5.3 e para o set de validação no gráfico 5.4.

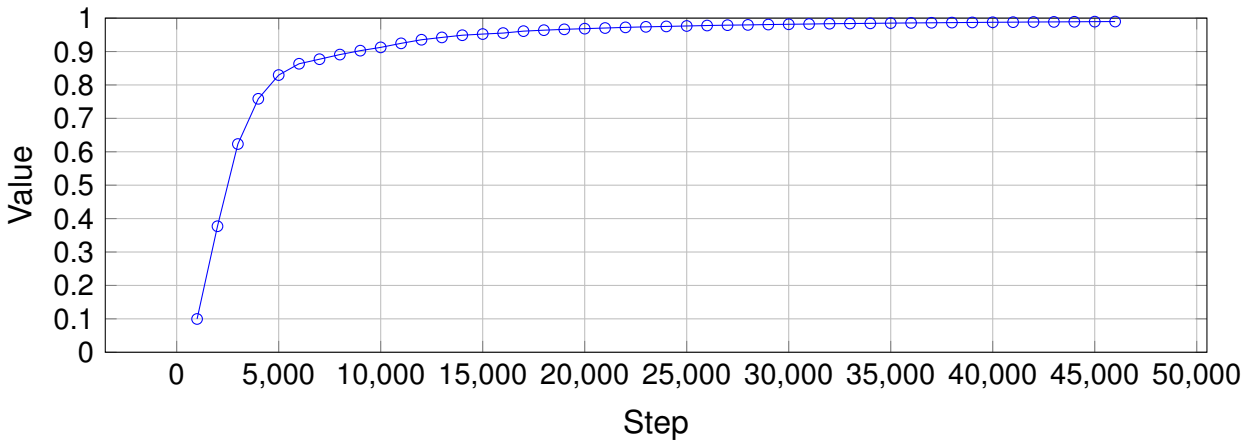

Figura 5.3 – Step x Acurácia de treino para treino de bateria - tudo sem vocal.

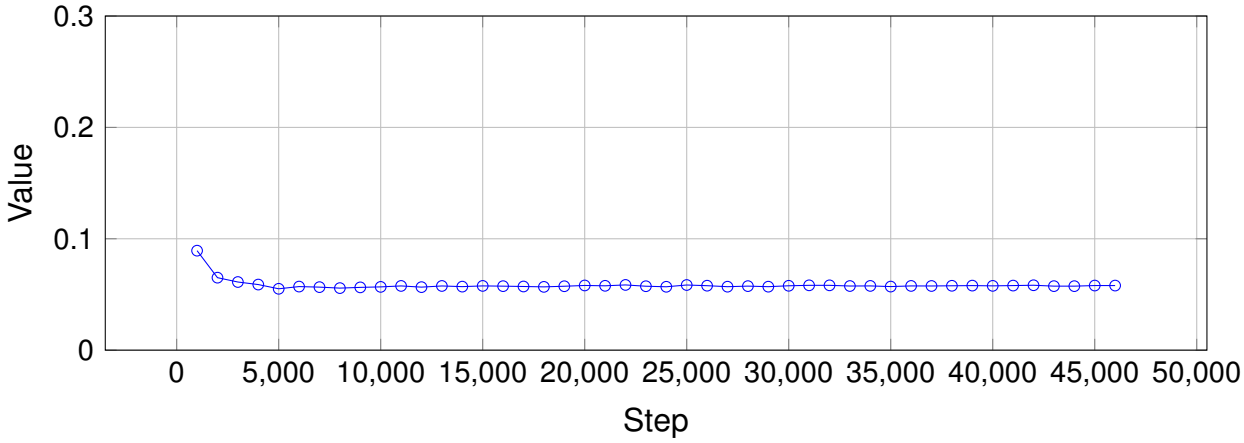

Figura 5.4 – Step x Acurácia de validação para treino de bateria - tudo sem vocal.

5.1.3 Bateria -> apenas baixo

O terceiro modelo utilizando a bateria foi o que menos gerou barulho, as saídas indevidas (as que não teriam chance de serem utilizadas como acompanhamento), ao invés de serem barulho, consistem de silêncio. Dentre as 108 entradas, 11 acompanhamentos válidos foram gerados, dentre estes 7 consistem da mesma melodia. Podemos observar um exemplo da melodia repetida na oitava linha da tabela 5.2 e um exemplo único na nona linha. Podemos observar a acurácia do modelo durante o treino para o set de treino no gráfico 5.5 e para o set de validação no gráfico 5.6.

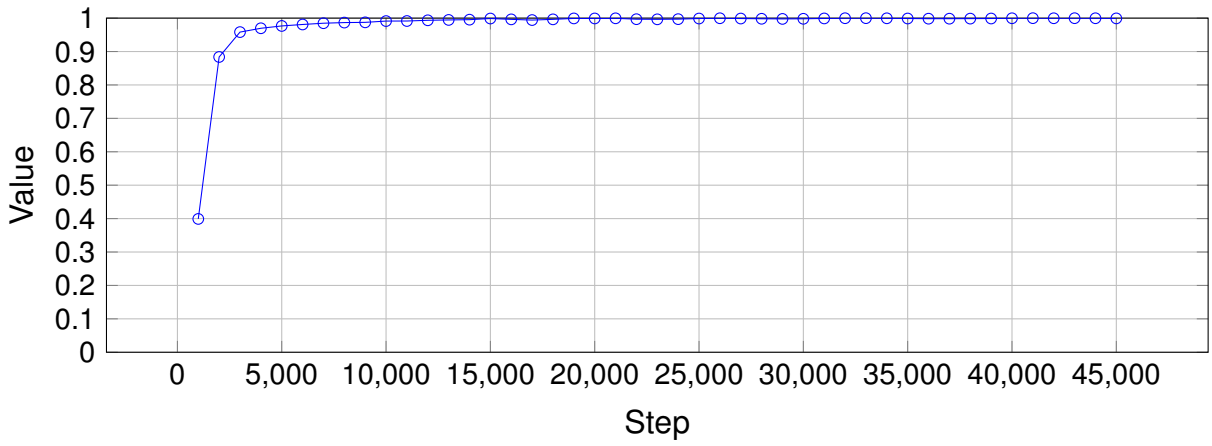

Figura 5.5 – Step x Acurácia de treino para treino de bateria - baixo.

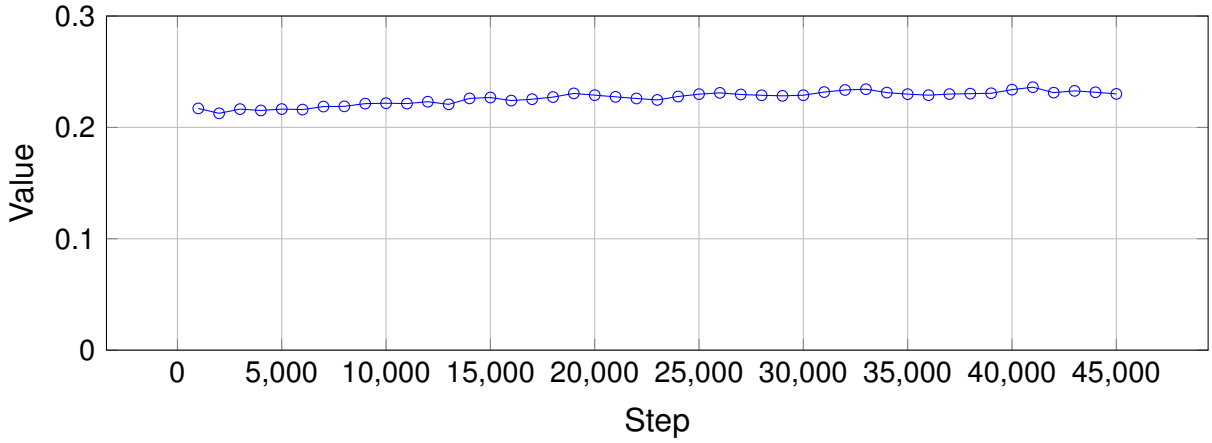

Figura 5.6 – Step x Acurácia de validação para treino de bateria - baixo.

5.2 Baixo como entrada

5.2.1 Baixo -> apenas bateria

Para o quarto modelo, invertemos a entrada e a saída do terceiro modelo, assim podendo testar um caso com mais definição melódica. Neste modelo, observamos um comportamento similar ao modelo Bateria-Baixo, nenhum caso gerou barulho incoerente, mas muitos geraram silêncio, além disso, por volta de 15 saídas possuíam sons de bateria, em todos faltavam um senso rítmico, e na grande maioria, em 13 casos, o acompanhamento começava e parava após alguns segundos.

Podemos ouvir um exemplo de uma saída completa na décima linha da tabela 5.2 e uma que para antes de terminar na décima primeira linha. Podemos observar a acurácia do modelo durante o treino para o set de treino no gráfico 5.7 e para o set de validação no gráfico 5.8.

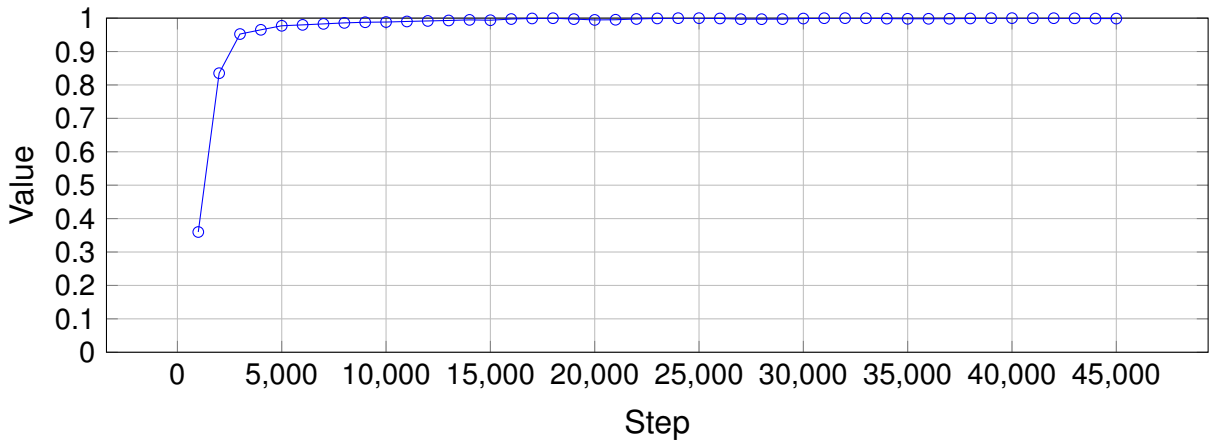

Figura 5.7 – Step x Acurácia de treino para treino de baixo - bateria.

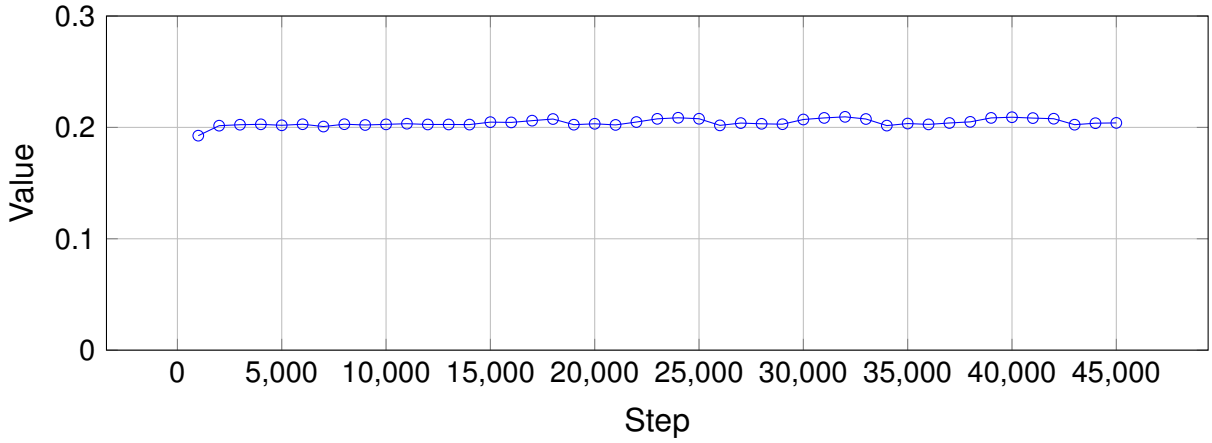

Figura 5.8 – Step x Acurácia de validação para treino de baixo - bateria.

5.2.2 Baixo -> tudo sem vocal

Até aqui, todas as saídas ou eram válidas e se encaixavam como música, ou eram barulho puro, ou silêncio. Porém, no modelo Baixo - Tudo sem vocal, vemos uma mudança de comportamento, tendo poucos casos de barulho ou silêncio, mas muitos casos de sons aleatórios e desorganizados, mas ainda reconhecíveis como instrumentos (por exemplo, como se alguém estivesse batendo aleatoriamente na bateria). Essas entradas também não servem como acompanhamento, mas são muito mais próximas do desejado do que barulho por completo. Podemos observar um exemplo disto na décima segunda linha da tabela 5.2.

Dito isto, este foi um dos piores modelos em gerar acompanhamentos válidos, tendo sucesso em 7 casos, mas sem nenhuma melodia repetida. Dentre tanto as entradas válidas, como as de sons de instrumentos sem estrutura, percebemos uma prevalência em sons de bateria e percussão no geral, sendo estes mais frequentes e altos que outros

instrumentos. Podemos ouvir um exemplo na décima terceira linha da tabela 5.2, podemos observar o gráfico do treino em 5.9 e o da validação em 5.10.

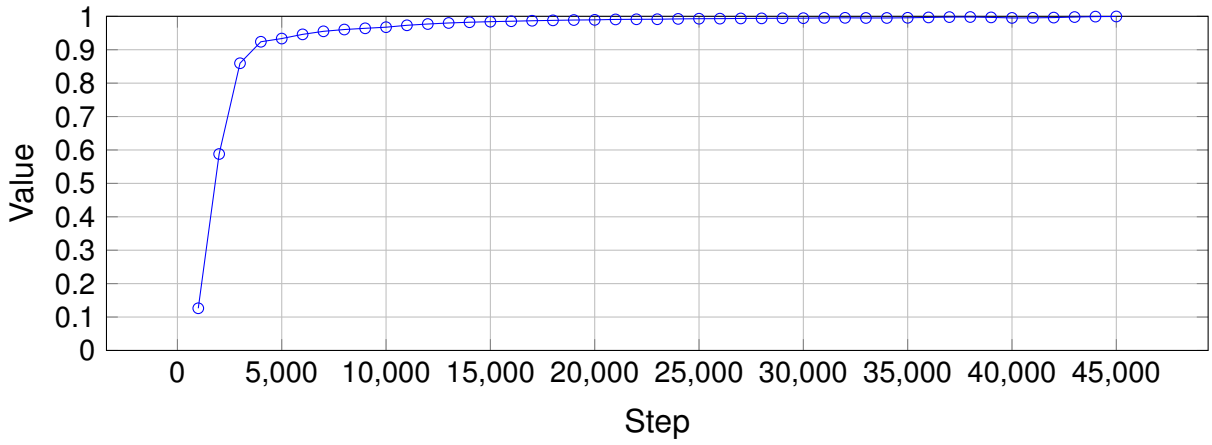

Figura 5.9 – Step x Acurácia de treino para treino de baixo - tudo sem vocal.

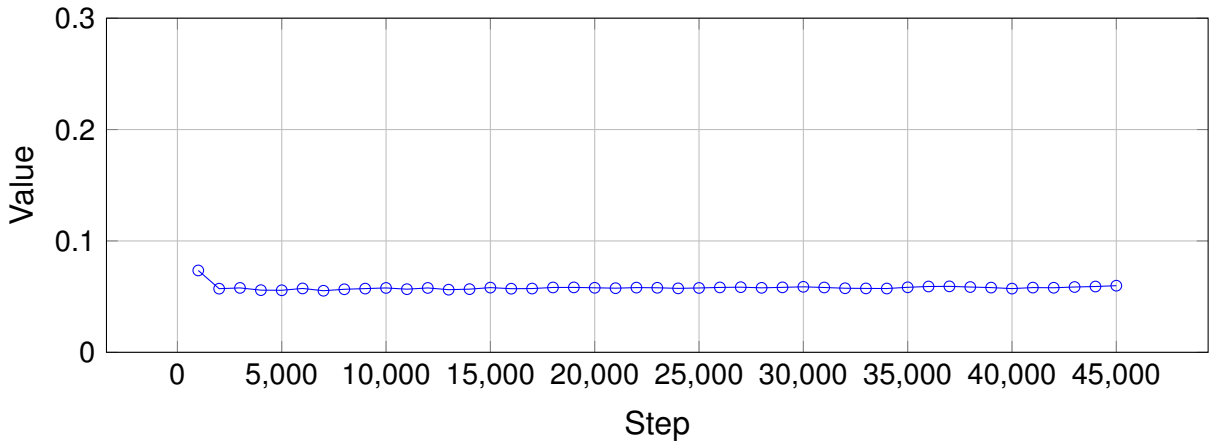

Figura 5.10 – Step x Acurácia de validação para treino de baixo - tudo sem vocal.

5.2.3 Baixo -> todos instrumentos

O último modelo foi treinado utilizando o baixo como entrada e todo o resto como saída, aqui, os resultados foram os melhores em quantidade com 20 resultados válidos, dentre estes, uma melodia repetida foi encontrada, também foram encontradas outras 3 melodias similares entre si, mas não iguais. Podemos observar um exemplo de uma da melodia repetida na décima quarta linha da tabela 5.2, e um exemplo de uma das melodias similares mas não iguais na décima quinta e décima sexta linha.

Além disso, percebe-se um comportamento similar ao modelo anterior, onde os sons de bateria são mais frequentes e intensos, dito isto, os casos inválidos aqui são, dife-

rente do modelo anterior, barulho puro. Podemos observar a acurácia de treino em 5.9 e de validação em 5.10.

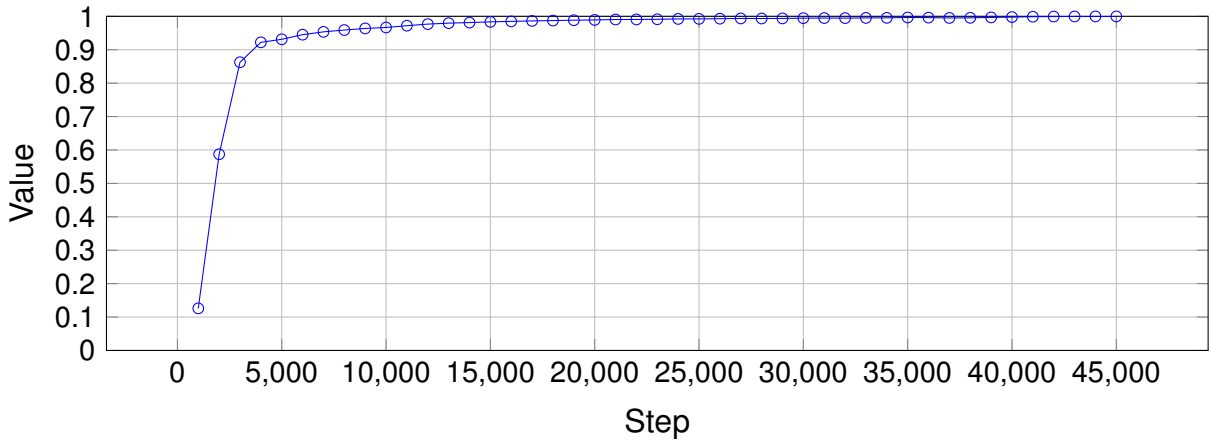

Figura 5.11 – Step x Acurácia de treino para treino de baixo - tudo.

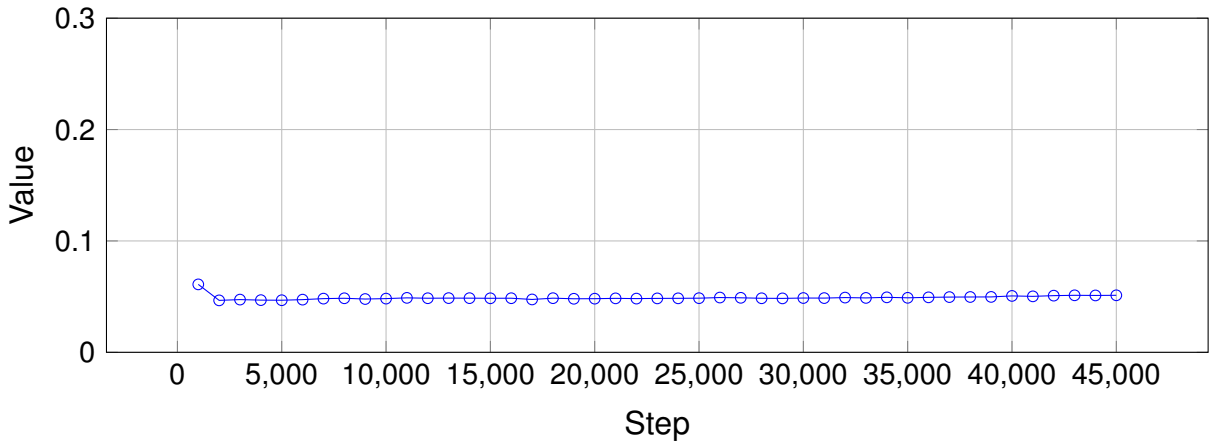

Figura 5.12 – Step x Acurácia de validação para treino de baixo - tudo.

6. DISCUSSÃO DOS RESULTADOS

De forma geral, os resultados deixaram a desejar, isso se deve principalmente à limitação dos dados. Podemos observar evidências dessa limitação em diversas situações. Em quase todos os modelos treinados, observamos melodias repetidas com entradas similares. Essas entradas, por mais que similares, não eram iguais, e pode-se perceber que esses padrões repetidos se dão pela existência de entradas similares às que geram as melodias repetidas no dataset de treino, mesmo que em músicas e contextos diferentes. Mesmo assim, o modelo não se limita completamente a só aquilo que ele já viu, conseguindo também gerar melodias únicas.

Durante a fase de treinamento, observou-se que o modelo alcança rapidamente uma alta precisão, aproximadamente 100% no set de treino, após cerca de 15.000 iterações, mesmo utilizando um tamanho de lote relativamente pequeno, de 32. Essa rápida convergência à precisão máxima, apesar do número limitado de iterações e do pequeno batch size, sugere uma significativa capacidade de aprendizado do modelo para casos já vistos. No entanto, essa situação também levanta preocupações sobre a potencial falta de generalização, indicando que o modelo pode estar sofrendo de overfitting. Se vê o overfitting mais claramente quando observamos os gráficos de validação, que demonstram pouca variação ao longo do tempo.

A acurácia como métrica funciona para observar o aprendizado em cima de áudios já antes vistos, mas para áudios inéditos, não funciona bem. Nesses casos, opiniões populares e análises melódicas são o ideal, pois a saída gerada pelo modelo pode ser completamente diferente do label, mas ainda pode seguir uma estrutura musical e/ou soar bem para o ouvinte.

Observamos que, por mais que a entrada inteira não precise ser igual ao que o modelo já viu, ele precisa ter visto e reconhecer os padrões da entrada, e em casos onde ele não reconhece esses padrões, ele gera uma "média"de tudo, gerando barulho nos modelos com bastante som na saída, e silêncio nos modelos com pouco som na saída. É interessante de se perceber que, por mais que o modelo precise ter as informações dos padrões da entrada para conseguir reproduzir um acompanhamento válido, ele mesmo assim consegue gerar saídas completamente únicas e diferentes do que ele viu antes, ao "misturar"o conhecimento de partes da entrada.

Como visto na tabela 5.1, os modelos com o baixo como entrada tiveram uma facilidade maior de gerar acompanhamentos válidos, isso se deve, provavelmente, a maior limitação que o baixo apresenta para acompanhamentos, com regras mais restritas do que soa bem e não soa, diferente da bateria que só define o ritmo, e possui uma maior flexibilidade para a saída. Essa maior definição melódica da saída facilita a geração musical para o modelo, pois introduz menos incerteza e variabilidade para o que está correto.

Também podemos perceber a necessidade de balancear a quantidade de informação e complexidade contida na saída. Quanto mais instrumentos incluímos na saída, mais o modelo tende a ser barulhento e menos "preciso"com seus sons. Quando a saída possui pouca informação, o modelo tende a ser quieto, gerando silêncio ou saídas incompletas. Como mostrado na tabela 5.1, os modelos com menos informação (modelos com apenas um instrumento na saída) demonstram uma acurácia maior na validação final, mas um número igual ou menor aos outros modelos em acompanhamentos válidos, isso se deve a maior quantidade de silêncio nos labels, e a facilidade do modelo de conseguir cumprir com este silêncio.

Com os resultados com potencial gerados neste trabalho, concluímos que a criação de um modelo de geração de acompanhamento é completamente plausível, mas necessitaria de uma quantidade muito maior de dados que a utilizada neste trabalho. Com uma quantidade maior de dados, seria possível treinar um modelo fundacional para geração de acompanhamento musical, abrindo portas para finetuning seja feito em cima deste modelo fundacional com muito menos dados (como a quantidade usada aqui) para treinar o modelo para gêneros, bandas ou artistas específicos.

No entanto, este modelo, atualmente, seria inviável para uma possível aplicação em tempo real para o hardware médio disponível. Além de uma limitação de hardware (inferência de 1 clipe de 10 segundos em uma RTX 4090 leva em torno de 30 segundos), também seria necessário uma reestruturação do modelo para um treino com diferentes tamanhos de trechos de áudio.

7. CONCLUSÃO

Em suma, este estudo explorou a implementação e eficácia de um modelo generativo baseado em transformers para a criação de acompanhamentos musicais a partir de uma melodia base. Os resultados, embora limitados pela quantidade de dados, mostraram que o modelo é capaz de produzir acompanhamentos musicais em contextos específicos, sugerindo que, com um conjunto de dados mais amplo e diversificado, o desempenho do modelo poderia ser significativamente melhorado.

O desafio apresentado pela variabilidade vocal e a necessidade de uma grande quantidade de dados para treinamento eficaz indicam áreas claras para futuras pesquisas e desenvolvimento. Além disso, os modelos que excluíam a voz dos acompanhamentos tenderam a gerar resultados menos ruidosos e mais coerentes, destacando a complexidade de modelar a voz humana de maneira convincente.

A aplicação de modelos como o SingSong adaptado demonstrou potencial não apenas para pesquisa acadêmica, mas também para aplicações práticas, como assistência a músicos e criadores de conteúdo musical. Continuar refinando esses modelos poderia eventualmente levar a ferramentas poderosas para a criação musical automática, democratizando a produção musical de alta qualidade e fomentando ainda mais a criatividade humana através da colaboração entre homem e máquina.

Por fim, este trabalho estabelece uma base sólida para o avanço na área de modelos generativos de música, mas também destaca a necessidade de avanços em coleta de dados, melhorias no design do modelo e técnicas de treinamento que podem aproveitar ao máximo as capacidades dos transformers para processamento de música. A exploração futura dessas áreas promete não apenas avançar nosso entendimento e capacidade em modelagem de música com inteligência artificial, mas também expandir as possibilidades criativas disponíveis para músicos e artistas em todo o mundo.

REFERÊNCIAS BIBLIOGRÁFICAS

[B+24] Bittner, R. M.; et al.. “Medleydb: A dataset of annotations for music information retrieval”. Acessado: 31/05/2024, 2024.

[bin] “Bing ai: Enhancing search with large language models”. Accessed: 2023-11-09.

[BMV+23] Borsos, Z.; Marinier, R.; Vincent, D.; et al.. “Audiolm: A language modeling approach to audio generation”, *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 31–1, 2023, pp. 34–45, accessed: 2023-11-09.

[BST+14] Bittner, R.; Salamon, J.; Tierney, M.; Mauch, M.; Cannam, C.; Bello, J. P. “Medleydb: A dataset of multitrack audio for music research”. https://medleydb.weebly.com/, 2014.

[cha] “Chatgpt: Optimizing language models for dialogue”. Accessed: 2023-11-09.

[CLS+20] Chinen, M.; Lim, F. S. C.; Skoglund, J.; Gureev, N.; O’Gorman, F.; Hines, A. “Visqol v3: An open source production ready objective speech and audio metric”. Accessed: 2023-11-09, 2020.

[CZG21] Chung, Y.-A.; Zhang, Y.; Glass, J. “w2v-bert: Combining contrastive learning and masked language modeling for self-supervised speech pre-training”, *arXiv preprint arXiv:2108.06209*, 2021.

[Dav88] Davis, M. “Rare interview”. Timestamp: 21:27, Capturado em: https://www.youtube.com/watch?v=KllwtKMtYTA, 1988.

[DCLT19] Devlin, J.; Chang, M.-W.; Lee, K.; Toutanova, K. “Bert: Pre-training of deep bidirectional transformers for language understanding”. In: Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics, 2019, accessed: 2023-11-09.

[Eng23] Engel, C. D. A. C. A. R. E. M. P. E. A. A. M. V. I. S. O. P. N. Z. J. “Singsong: Generating musical accompaniments from singing”, *arXiv preprint arXiv:2301.12662*, 2023, https://arxiv.org/pdf/2301.12662.pdf.

[Goo24a] Google. “Tpu research cloud (trc)”. Acessado: 31/05/2024, 2024.

[Goo24b] Google Cloud. “Cloud tpu”. Capturado em: https://cloud.google.com/tpu, 2024.

[Goo24c] Google Research. “Base configuration for t5 1.1 in t5x”. Accessed: 2024-06-14, Capturado em: https://github.com/google-research/t5x/blob/main/t5x/examples/t5/t5_1_1/base.gin, 2024.

[KX23] Kirillov, A.; Xiao, T. “Segment anything”. Accessed: 2023-11-09, 2023.

[MP43] McCulloch, W. S.; Pitts, W. “A logical calculus of the ideas immanent in nervous activity”, *The bulletin of mathematical biophysics*, vol. 5–4, 1943, pp. 115–133.

[ope21] “Preparing your dataset for fine-tuning”. Acesso em: 26 de junho de 2024, 2021.

[RCL+22] Roberts, A.; Chung, H. W.; Levskaya, A.; Mishra, G.; Bradbury, J.; et al.. “T5x: A modular, composable, research-friendly framework for training sequence models at many scales”. Acessado: 31/05/2024, 2022.

[Res23] Research, F. “EnCodec: Open Source Library for Neural Compression”, 2023.

[RPG+21] Ramesh, A.; Pavlov, M.; Goh, G.; et al.. “Zero-shot text-to-image generation”. Accessed: 2023-11-09, 2021.

[RSR+20] Raffel, C.; Shazeer, N.; Roberts, A.; Lee, K.; Narang, S.; Matena, M.; Zhou, Y.; Li, W.; Liu, P. J. “Exploring the limits of transfer learning with a unified text-to-text transformer”, *Journal of Machine Learning Research*, vol. 21–140, 2020, pp. 1–67.

[She20] Sherstinsky, A. “Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network”, *Physica D: Nonlinear Phenomena*, vol. 404, March 2020.

[Tec23] TechCrunch. “Scenario lands $6m for its ai platform that generates game art assets”, *TechCrunch*, 2023.

[TVJ22] Toderici, G.; Vincent, D.; Johnston, N. “End-to-end optimized image compression with latent transformations”, *arXiv preprint arXiv:2210.13438*, 2022.

[VSP+17] Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, ; Polosukhin, I. “Attention is all you need”, *arXiv preprint arXiv:1706.03762*, 2017, https://arxiv.org/pdf/1706.03762.pdf.

[Z+21] Zeghidour, N.; et al.. “Soundstream: An end-to-end neural audio codec”. Accessed: 2023-11-09, 2021.

[Zip23] Zippia. “23+ artificial intelligence and job loss statistics [2023]: How job automation impacts the workforce”, *Zippia*, 2023.