

**INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DA PARAÍBA
CAMPUS DE CAJAZEIRAS
CURSO SUPERIOR DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE
SISTEMAS**

ULISSES ALBUQUERQUE PEREIRA

Sonhin: UMA DIGITAL AUDIO WORKSTATION COM MACHINE LEARNING

CAJAZEIRAS
2024

Ulisses Albuquerque Pereira

Sonhin: UMA DIGITAL AUDIO WORKSTATION COM MACHINE LEARNING

Trabalho de Conclusão de Curso submetido ao Instituto Federal de Educação, Ciência e Tecnologia da Paraíba, Campus Cajazeiras, como requisito parcial para a obtenção do grau de Tecnólogo em Análise e Desenvolvimento de Sistemas.

Orientador: Prof. Dr. Fabio Gomes de Andrade

Cajazeiras
2024

IFPB / Campus Cajazeiras
Coordenação de Biblioteca
Biblioteca Prof. Ribamar da Silva
Catalogação na fonte: Cícero Luciano Félix CRB-15/750

P436s	<p>Pereira, Ulisses Albuquerque. Sonhin : uma digital audio workstation com machine learning / Ulisses Albuquerque Pereira. – 2024.</p> <p>64f. : il.</p> <p>Trabalho de Conclusão de Curso (Tecnólogo em Análise e Desenvolvimento de Sistemas) - Instituto Federal de Educação, Ciência e Tecnologia da Paraíba, Cajazeiras, 2024.</p> <p>Orientador(a): Prof. Dr. Fabio Gomes de Andrade.</p> <p>1. Desenvolvimento de sistemas. 2. <i>Machine Learning</i>. 3. Produção musical. 4. Digital Audio workstation. I. Instituto Federal de Educação, Ciência e Tecnologia da Paraíba. II. Título.</p>
IFPB/CZ	CDU: 004.4:37(043.2)



MINISTÉRIO DA EDUCAÇÃO
SECRETARIA DE EDUCAÇÃO PROFISSIONAL E TECNOLÓGICA
INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DA PARAÍBA

ULISSES ALBUQUERQUE PEREIRA

Sonhin: UMA DIGITAL AUDIO WORKSTATION COM MACHINE LEARNING

Trabalho de Conclusão de Curso apresentado junto ao Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas do Instituto Federal de Educação, Ciência e Tecnologia da Paraíba - Campus Cajazeiras, como requisito à obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas.

Orientador

Prof. Dr. Fabio Gomes de Andrade

Aprovada em: **21 de Outubro de 2024.**

Prof. Dr. Fabio Gomes de Andrade - Orientador

Prof. Me. Diogo Dantas Moreira - Avaliador
IFPB - Campus Cajazeiras

Prof. Me. Fábio Abrantes Diniz
IFPB - Campus Cajazeiras

Documento assinado eletronicamente por:

- **Fabio Gomes de Andrade**, PROFESSOR ENS BASICO TECN TECNOLOGICO, em 22/10/2024 17:29:52.
- **Fabio Abrantes Diniz**, PROFESSOR ENS BASICO TECN TECNOLOGICO, em 22/10/2024 17:57:09.
- **Diogo Dantas Moreira**, PROFESSOR ENS BASICO TECN TECNOLOGICO, em 23/10/2024 12:10:13.

Este documento foi emitido pelo SUAP em 21/10/2024. Para comprovar sua autenticidade, faça a leitura do QRCode ao lado ou acesse <https://suap.ifpb.edu.br/autenticar-documento/> e forneça os dados abaixo:

Código 623093
Verificador: abd211144c
Código de Autenticação:



Rua José Antônio da Silva, 300, Jardim Oásis, CAJAZEIRAS / PB, CEP 58.900-000
<http://ifpb.edu.br> - (83) 3532-4100

RESUMO

Alguns avanços na área de *machine learning*, que possibilitam desde carros autônomos até sistemas capazes de pesquisar fármacos, estão sendo agora incluídos no ramo de produção musical. Contudo, a maioria dos produtos desse segmento gera conteúdo sem que o usuário possa alterar diretamente suas partes, limitando o potencial criativo e o reaproveitamento de elementos desejáveis dos resultados. Visando resolver esse problema, este trabalho desenvolveu uma *Digital Audio Workstation* intitulado Sonhin, em que conteúdo e sugestões de alteração gerados com *machine learning* são integrados diretamente ao fluxo de trabalho do usuário. Embora o sistema não tenha se mostrado capaz de gerar músicas inteiras com qualidade, devido à escassez de dados compatíveis para o treinamento, a geração de instrumentos virtuais com algoritmos genéticos funcionou muito bem e talvez seja aproveitada em versões futuras.

Palavras-chave: *Machine Learning*, Produção musical, *Digital Audio Workstation*

ABSTRACT

Some advances in the field of machine learning, enabling technologies from autonomous cars to systems capable of drug research, are now being applied to the music production industry. However, most products in this segment generate content without allowing users to directly modify its components, limiting the creative potential and the reuse of desirable elements from the results. To address this issue, this work developed a Digital Audio Workstation (DAW) called Sonhin, in which content and modification suggestions generated by machine learning are integrated directly into the user's workflow. Although the system was not capable of generating entire songs with high quality due to the lack of compatible training data, the generation of virtual instruments using genetic algorithms worked very well and might be utilized in future versions.

Keywords: Machine Learning, Music Production, Digital Audio Workstation

LISTA DE FIGURAS

Figura 1 - Página de Músicas.....	23
Figura 2 - Página de Padrão.....	24
Figura 3 - Knobs em uma aplicação de áudio (Vital).....	25
Figura 4 - Página de Progressão.....	26
Figura 5 - Página de Arranjo.....	27
Figura 6 - Página de Instrumento.....	28
Figura 7 - Página de Mixagem.....	29
Figura 8 - Página de Grafo.....	30
Figura 9 - Página de <i>Preset</i>	31
Figura 10 - Prévia de música.....	31
Figura 11 - Página de Conta de Visitante.....	32
Figura 12 - Página de Conta Cadastrada.....	33
Figura 13 - Diagrama entidade-relacionamento do banco de dados.....	35
Figura 14 - diagrama de entidade e relacionamento.....	37
Figura 15 - Arquitetura do <i>front-end</i>	39
Figura 16 - Arquitetura do servidor da aplicação.....	41
Figura 17 - Arquitetura do treino do avaliador.....	43
Figura 18 - Arquitetura do <i>seeder</i> de dados.....	44

LISTA DE QUADROS

Quadro 1 – Atividades para a disciplina de TCC 1.....	15
Quadro 2 – Atividades para a disciplina de TCC 2.....	15
Quadro 3 – Comparação entre ferramentas de produção musical.....	21

LISTA DE CÓDIGOS

Código 1 – Classe derivada com Tailwind CSS.....	46
--	----

LISTA DE ABREVIATURAS E SIGLAS

AIVA	<i>Artificial Intelligence Virtual Artist</i>
CNN	<i>Convolutional Neural Network</i>
DAW	<i>Digital Audio Workstation</i>
EDM	<i>Electronic Dance Music</i>
FNN	<i>Feedforward Neural Network</i>
GAN	<i>Generative Adversarial Network</i>
GPU	<i>Graphics Processing Unit</i>
IA	Inteligência Artificial
JSON	<i>JavaScript Object Notation</i>
JSX	<i>JavaScript XML</i>
LFO	<i>Low Frequency Oscillation</i>
LLM	<i>Large Language Model</i>
LSTM	<i>Long Short Term Memory</i>
MAI	<i>Musical Artificial Intelligence</i>
MVP	<i>Minimum Viable Product</i>
VST	<i>Virtual Studio Technology</i>

SUMÁRIO

1 INTRODUÇÃO.....	11
1.1 MOTIVAÇÃO.....	11
1.2 OBJETIVOS.....	13
1.2.1 Objetivo geral.....	13
1.2.2 Objetivos específicos.....	13
1.3 METODOLOGIA.....	14
2 FUNDAMENTAÇÃO TEÓRICA.....	15
2.1 CONCEITOS BÁSICOS DE MÚSICA.....	15
2.2 CONCEITOS BÁSICOS DE APRENDIZADO DE MÁQUINA.....	17
3 TRABALHOS RELACIONADOS.....	19
4 SOLUÇÃO PROPOSTA.....	21
4.1 REQUISITOS FUNCIONAIS.....	22
4.2 INTERFACE DO USUÁRIO.....	22
4.2.1 Página de projetos.....	23
4.2.2 Página de padrão.....	23
4.2.3 Página de progressão harmônica.....	25
4.2.4 Página de arranjo.....	26
4.2.5 Página de instrumento.....	27
4.2.6 Página de mixagem.....	28
4.2.7 Páginas de grafos.....	29
4.2.8 Páginas de presets.....	30
4.2.9 Página de conta.....	32
4.3 MODELAGEM DOS DADOS.....	33
4.4 ARQUITETURA DO SISTEMA.....	38
4.5 TECNOLOGIAS UTILIZADAS.....	44
4.5.1 Front-end.....	45
4.5.2 Interface Gráfica.....	45
4.5.3 Back-end.....	46
4.5.4 Motor de áudio.....	47
4.5.5 Aprimorador.....	48
5 RESULTADOS.....	49
6 CONCLUSÃO.....	49
7 TRABALHOS FUTUROS.....	50
REFERÊNCIAS.....	52
APÊNDICE A - Motivos do nome Sonhin.....	59
APÊNDICE B - Iterações anteriores do projeto.....	60
Mysic.....	60
MAI.....	60
MAI 1.....	60
MAI 2.....	61

MAI 3.....	61
MAI 4.....	61
MAI 5.....	61
MAI 6.....	62
Sonhin 1.....	62
Sonhin 2.....	63
Sonhin 3.....	63

1 INTRODUÇÃO

Digital Audio Workstations (DAWs) são programas destinados a produção musical que normalmente contam com várias funções, como gravar áudio, fazer arranjos e adicionar efeitos (YANG, 2024). Tornaram-se um componente central na produção musical (FAJAR; SUKMAYADI, 2021), permitiram que basicamente qualquer indivíduo com um computador pessoal possa criar músicas, e moldaram gêneros musicais inteiros (REUTER, 2021).

Contudo, criar músicas usando uma DAW continua sendo um processo complexo que demanda tempo e dedicação (JILLINGS; STABLES, 2017). Pois, os resultados obtidos dependem primariamente do conhecimento e habilidade do usuário (REGELSKI, 2007).

Concomitantemente, o campo de Inteligência Artificial vem apresentando grandes avanços na geração de conteúdo. Já é possível gerar textos, imagens, vídeos e áudios para os mais diversos segmentos (FEUERRIEGEL et al., 2024), incluindo produção musical (STURM et al., 2024).

Porém, as ferramentas atuais de geração de música apresentam graves limitações no controle pelo usuário, em geral se baseando em apenas alguns parâmetros ou *prompt* (ZHU et al., 2023), impossibilitando a cooperação direta com produtores musicais humanos (BRIOT; PACHET, 2020). Como consequência as músicas geradas tendem a ser genéricas e sem expressividade (POSTOLACHE, 2024).

Esse trabalho se propõe a solucionar tanto a demora de produzir músicas em DAWs como a falta de controle de geradores de música por meio da criação de uma DAW intitulada Sonhin (a explicação do nome encontra-se no Apêndice A) que possui funções de geração de conteúdo e de sugestões de alterações por Inteligência Artificial integrada ao fluxo de trabalho do usuário.

1.1 MOTIVAÇÃO

Produzir músicas costuma ser um processo muito manual (JILLINGS; STABLES, 2017). Em sua maior parte consiste em fazer alterações (por exemplo, de notas e efeitos), avaliar se as alterações foram boas, mantê-las ou descartá-las, e repetir até obter um resultado bom o bastante. O processo como um todo costuma demorar de algumas horas a dias para cada música.

Um sistema capaz de criar músicas inteiras pode futuramente poupar muito tempo de compositores humanos. Estes poderiam passar menos tempo aprendendo controles técnicos (por exemplo automação de filtro passa-baixo em ruído branco), e mais tempo focando em características de mais alto nível, como histórias e sentimentos (POSTOLACHE, 2024).

Considerando o longo termo, é provável que *Machine Learning* venha a ser uma parte integral do ramo musical, pois, computadores podem lidar com quantidades exorbitantes de *feedback*, a custos cada vez menores e durante tempo indefinido (SAGIROGLU; SINANC, 2013). As futuras gerações talvez até vejam as músicas criadas até agora como muito amadoras.

O uso de Inteligência Artificial para música também pode, futuramente, proporcionar experiências mais imersivas, como trilhas sonoras criadas sob demanda em videogames (HUTCHINGS; MCCORMACK, 2020), ou músicas feitas levando em consideração o humor do usuário (BARREAU, 2018).

Além desses fatores, o projeto foi escolhido devido a algumas vantagens econômicas de se produzir música com Inteligência Artificial, descritas a seguir.

Geração de conteúdo com Inteligência Artificial possui efeitos de rede. Cada usuário a mais na plataforma possibilita mais correção e personalização do gerador. O que facilita a aquisição e retenção de usuários, em uma espiral positiva. Essa dinâmica favorece o monopólio, que é um dos objetivos comerciais mais visados (THIEL; MASTERS, 2014).

O engajamento dos usuários pode ser facilmente medido e usado para treinamento. Por exemplo, plataformas de *streaming* de música registram cada vez que uma música é consumida, por quem, a que hora, quantas vezes foi pulada ou compartilhada, etc (MAASØ; HANGEN, 2020). Futuramente, podem ser implementados até mesmo testes A/B de uma mesma música com sutis variações para um mesmo usuário, já que humanos tendem a escutar músicas de que gostam mais uma vez.

De forma geral, pode-se estabelecer uma métrica simples como lucro de anúncios e assinaturas, e então otimizar o modelo inteiro de geração com base nela (STURM et al., 2024). Ao passo que outros problemas, como tratamento de pacientes, são especialmente demorados. E outros, como xadrez ou go, têm *feedback* muito discreto (apenas vitória, derrota ou empate) (BURMEISTER; WILES, 2002).

Muitos dos problemas de Inteligência Artificial precisam de dados reais como entrada para serem úteis. Para ilustrar, consideremos um sistema de análise de crédito. Seu objetivo é calcular a probabilidade de um cliente específico pagar um empréstimo caso aprovado (SADOK; SAKKA; EL MAKNOUZI, 2022). Por definição, o sistema simplesmente não pode emitir o resultado antes de ter acesso aos dados do cliente. Por outro lado, um sistema de geração de músicas não precisa de dados do mundo real para gerar novas músicas. Uma vez funcional, ele pode gerar um número virtualmente ilimitado de músicas para serem publicadas ou postas à venda para criadores de conteúdo usando como entrada simples valores pseudo aleatórios (STURM et al., 2024).

1.2 OBJETIVOS

Esta seção descreve o objetivo geral e os objetivos específicos a serem alcançados por meio do desenvolvimento deste TCC.

1.2.1 Objetivo geral

O objetivo geral deste trabalho é desenvolver uma DAW, que reduza a quantidade de trabalho manual necessário para a criação de músicas por meio de conteúdo gerado e alterações sugeridas por Inteligência Artificial.

1.2.2 Objetivos específicos

Para guiar o desenvolvimento do projeto, os objetivos específicos para o sistema são:

- Criar músicas semelhantes às humanas, tanto em sonoridade, como em estrutura, e técnicas;

- Contar com variabilidade, ou seja produzir resultados diferentes o bastante uns dos outros – versões anteriores do projeto já falharam devido ao *overfitting*, em que os resultados eram sempre iguais;
- Permitir treinamento através de feedback humano;
- Não depender de muitas músicas prontas, uma vez que em geral são protegidas por direitos autorais.

1.3 METODOLOGIA

O Sonhin é um projeto comercial desenvolvido sob a metodologia de *Lean Startup*, que é marcada por ciclos de desenvolvimento de produto como forma de testar hipóteses, “e ver quais partes são brilhantes e quais são absurdas” (RIES, 2011). As onze iterações anteriores do projeto estão descritas no Apêndice B.

Para essa iteração, o projeto pode ser dividido nas seguintes atividades:

- Desenvolvimento de um motor de áudio (A1) responsável por renderizar áudio a partir de notas e instrumentos. Deve visar por desacoplamento da semântica da música, uma vez que renderizar áudio em tempo real demanda otimização de baixo nível, o que dificulta manter estruturas complexas de dados.
- Desenvolvimento de uma interface gráfica para coletar *feedback* (A2). Esta deve priorizar a facilidade de uso e possibilidade de salvar as alterações feitas pelo usuário de forma reproduzível.
- Desenvolvimento de um modelo de Inteligência Artificial (A3), que deve ser capaz de alterar os dados de uma música da mesma forma que um usuário, possibilitando treiná-la com esse mesmo tipo de interações.
- Integração dos sistemas (A4). Trata-se de uma etapa fundamental para tornar o produto usável, e que costuma demandar muito tempo devido a erros inesperados e difíceis de replicar.

Cabe citar que algumas etapas, como levantamento de requisitos e modelagem de dados já foram realizadas continuamente em iterações anteriores.

2 FUNDAMENTAÇÃO TEÓRICA

Esta seção expõe explicações tanto de música como Inteligência Artificial para os conceitos utilizados neste trabalho.

2.1 CONCEITOS BÁSICOS DE MÚSICA

Música tende a ser mais fácil de reconhecer do que de definir (GODT, 2005). Algumas definições como “conjunto organizado de sons”, falham frente a contra-exemplos como fala humana (KANIA, 2024). Outras definições, que focam em características agradáveis da música, como harmonia, desconsideram o papel do gosto pessoal (GODT, 2005), cultura e música experimental (ELDRITCH, 2013) (por exemplo em “Dystopia” de KSVL Noh).

Mas para os propósitos deste trabalho basta a aproximação de música como “algo percebido como música”. Para esse critério existem várias afirmações que não são absolutas, mas funcionam a maior parte das vezes. Por exemplo, pode-se dizer que uma música é composta por notas. Nota é uma abstração de um som (que pode ser a soma de outros sons) que possui altura, um instante de início e duração (NATTIEZ, 1990).

A duração de uma nota normalmente é representada em divisões do tempo (BENWARD; SAKER, 2020), já que uma música pode ser tocada mais rápida ou mais devagar sem perder a semântica. Mas nesse projeto, assim como em vários softwares, a duração é dada em segundos. Isso facilita representar efeitos como *strum* (um atraso ordenado de notas de um mesmo acorde, muito usado em pianos e violões).

A altura se refere a frequência percebida de um som. Uma nota com frequência alta é chamada de “aguda”, e uma com frequência baixa é “grave” (PLACK; OXENHAM; FAY, 2005). Algumas vezes o termo “nota” se refere à altura de uma nota (NATTIEZ, 1990). Um som normalmente contém várias frequências simultaneamente. A frequência mais audível é chamada de “fundamental”. É ela que marca a altura da nota. Já as outras frequências são as “harmônicas” (BENWARD; SAKER, 2020).

Notas normalmente são organizadas em conjuntos chamados padrões que costumam durar 2, 4 ou 8 barras. E se repetem ao longo da música. Também é comum que padrões tenham variações, em que se altera a sonoridade ou algumas notas. Espera-se que cada música tenha tanto repetição como variação de notas (MCADAMS; MATZKIN, 2001). Músicas com pouca variação normalmente são percebidas como maçantes, ao passo que músicas com variação demais são percebidas como caóticas.

Os padrões, por sua vez, são organizados em seções, que possuem funções específicas (FINDEISEN, 2015). Por exemplo:

- Intro: apresentar ao ouvinte o tema da música.
- Outro: produzir uma mudança gradual ou um fechamento ao final da música.
- Verso: despertar interesse através de conteúdo menos repetitivo.
- Refrão: tornar a música memorável através de repetição.
- Ponte: tornar a música interessante através de elementos de alguma forma diferentes do restante da música.

Alguns *softwares* modelam a disposição de padrões ao longo do tempo a partir de cliques. Nesse caso, o clique representa um padrão sendo usado na música com instantes inicial e final.

O equipamento ou *software* que transforma notas (abstrações) em sons (físicos ou em áudio) é chamado instrumento. Os instrumentos podem gerar sons de formas variadas. Instrumentos acústicos geralmente usam cordas (violão, piano), membranas (tambores), sopro (flauta), etc. Enquanto que instrumentos virtuais replicam esses sons por meio de gravações, ou sintetizam sons próprios, a partir de algoritmos. Dentre os principais tipos de síntese estão:

- Síntese aditiva, em que são somadas ondas de frequências diversas.
- Síntese subtrativa, em que são removidas harmônicas. Por exemplo, uma aproximação razoável de um baixo é uma onda dente de serra com um filtro passa-baixo que comece com uma frequência que decaia com o tempo.
- Síntese de amplitude modulada, que consiste em usar um oscilador para modular a amplitude (diferença entre picos e vales do áudio).
- Síntese de frequência modulada, que consiste em usar um oscilador para modular a frequência.

- Síntese granular, em que uma gravação é dividida em pequenas extensões (grânulos) que podem ser tocados em uma ordem diferente da original.

Uma possível abordagem para se obter sons com qualquer um desses tipos de síntese é através de sintetizadores modulares. Tratam-se de *softwares* ou equipamentos compostos por partes menores (módulos) que realizam funções específicas (HUANG, 2020), como remover frequências ou modular a amplitude.

Existem ferramentas chamadas efeitos que possibilitam fazer alterações em notas e áudios. Elas geralmente contam com controles, que o usuário pode usar para ajustá-los. Pode-se citar como exemplos de efeitos de áudio reverberação, saturação e distorção (ZÖLZER, 2011), e como efeitos de notas *strum*, *ghost notes*, e *staccato*.

2.2 CONCEITOS BÁSICOS DE APRENDIZADO DE MÁQUINA

Aprendizado de Máquina, ou *Machine Learning* (ML), é um subcampo de Inteligência Artificial (IA) que busca criar sistemas capazes de resolver problemas através de treinamento com dados, em vez de regras escritas manualmente (como geralmente ocorre na programação) (MAHESH, 2020). Pode ser subdividido em categorias com base em qual a natureza dos dados usados para o treinamento (AYODELE, 2010):

Aprendizado supervisionado usa dados anotados com a resposta desejada (CUNNINGHAM; CORD; DELANY, 2008). Por exemplo, em sistemas de reconhecimento de objetos a partir de imagens, as imagens usadas no treinamento são normalmente anotadas por humanos, e espera-se que o sistema consiga prever quais anotações seriam dadas a imagens novas.

Aprendizado não supervisionado usa dados não anotados (NAEEM et al., 2023). Por exemplo, em algoritmos de recomendação, é comum separar usuários em grupos, que podem ser deduzidos automaticamente, agrupando-se os dados por semelhança. Depois disso, os grupos podem ou não ser anotados para melhor legibilidade.

Aprendizado por reforço usa um ambiente com o qual um agente pode interagir para gerar os dados. O agente é otimizado considerando que o para emitir

ações que maximizem recompensas (FRANÇOIS-LAVET et al, 2018). É muito usado em inteligências artificiais para jogos, como xadrez e go.

Em todas essas categorias, o sistema precisa de uma métrica para otimizar. O valor de quão ruim foi o sistema é chamado de *loss* (perda) (CHNG, 2022). O objetivo de um sistema de ML é diminuir o *loss*. Dentre as formas de se reduzir o *loss* destacam-se algoritmos genéticos e retropropagação de erro.

Algoritmos genéticos seguem um processo análogo à seleção natural biológica (e por isso possuem termos semelhantes). Neles, existe um conjunto (população) de possíveis soluções (indivíduos) que são escolhidos (seleção) a partir de quão bem resolvem o problema (*fitness*)(MIRJALILI, 2019). Estes então passam por diferentes etapas (operadores genéticos) como criação de cópias (reprodução), troca de partes das soluções umas com as outras (*crossover*) e adição de valores aleatórios (mutação) (MIRJALILI, 2019). Esse tipo de otimização é muito usado quando o espaço de soluções possíveis é muito amplo (KUMAR et al., 2010).

Já a **retropropagação de erro** ajusta as variáveis do modelo com base no peso estimado que cada uma delas teve sobre o *loss* (BASHEER; HAJMEER, 2000). Costuma gerar resultados mais fácil e rapidamente, mas demanda que todas as operações, ou seja, todas as funções matemáticas usadas no modelo, possuam derivada.

Essa propriedade é a razão de várias complexidades adicionais. Por exemplo, modelos que funcionam a partir de dados textuais geralmente precisam converter palavras em representações numéricas multidimensionais (*tokens*) (GREFENSTETTE, 1999).

Outra dificuldade inerente das operações precisarem ser diferenciáveis é lidar com dados de tamanhos variados. As abordagens mais comuns incluem:

- **Padding** que consiste em determinar um tamanho máximo e concatenar valores nulos para os dados menores que ele (HASHEMI, 2019).
- **Usar recorrência**, em que o sistema processa os valores em uma posição e retorna o estado a ser usado na próxima posição (SCHMIDT, 2019).
- **Mecanismos de atenção** que funcionam como bancos de dados (usados em várias aplicações tradicionais), mas com lógica difusa (VASWANI et al., 2017).

Mecanismos de atenção vêm ganhando destaque graças a *large language models* (LLMs) como o GPT-4 usado no ChatGPT (MO et al., 2024). Suas vantagens incluem maior velocidade durante treinamento quando comparados com redes neurais recorrentes (RNN), e possibilidade de visualizar quais partes dos dados são mais importantes no cálculo do resultado (VASWANI et al., 2017).

3 TRABALHOS RELACIONADOS

Essa seção compara outras ferramentas no nicho de produção musical para tornar as diferenças de Sonhin claras.

Embora o método mais comum de se produzir música atualmente seja por meio de DAWs que não possuem integração com *machine learning*, uma vez que tendem a contar com os mesmos recursos principais (FAJAR; SUKMAYADI, 2021), apenas uma delas é apresentada nessa seção.

Fruity Loops Studio (FL Studio) é um DAW sem funcionalidades de ML, repleta de funcionalidades como gravar áudio, inserir instrumentos virtuais de terceiros, edição de padrões e arranjos, automação dos parâmetros de efeitos e monitoração dos níveis para mixagem (“FL Studio Online Manual”, [s. d.]). Nessa aplicação o usuário tem total liberdade de como ajustar notas, instrumentos e efeitos, mas não tem auxílio de *machine learning* para tomar essas decisões e poupar tempo.

SOUNDRAW é uma ferramenta automatizada de geração de músicas para criadores de conteúdo. Permite a criação de músicas com base em tema, sentimento e gênero (PUGACHEV, 2023). Além disso, conta com um editor para ajustar a intensidade da música ao conteúdo. Funciona com trechos pré-gravados que pertencem à empresa (SOUNDRAW, 2023), o que possibilita sonoridade acústica, detalhes intrincados e grande variedade. Porém, pelo mesmo motivo, não suporta edição de notas individuais e geração de novas melodias.

AIVA (Artificial Intelligence Virtual Artist) é uma ferramenta de criação de músicas para conteúdo, com várias opções de estilo. Seu algoritmo funciona com *deep learning* usando uma grande coleção de músicas clássicas (Mozart, Beethoven, Bach, etc) para treinamento (HAIMING, 2021). Além disso, pode usar músicas enviadas pelo usuário em formato MIDI como base para geração de outras

músicas. Embora conte com um editor integrado para alterar notas, articulações e instrumentos, esse não conta com grande variedade de efeitos e funcionalidades de edição.

Suno é uma Inteligência Artificial de criação de músicas a partir de descrições em texto (FREYBERG, 2024). É composto por três modelos de aprendizado profundo com funções específicas: criação de letras, síntese de voz com melodia, e geração de instrumental para acompanhamento (YU et al., 2024). Embora seja capaz de entregar resultados de qualidade, apresenta problemas em compor músicas longas coerentes e de cumprir descrições mais objetivas (como o andamento em BPM) (YU et al., 2024). Além disso, por trabalhar diretamente com áudio, apresenta resultados genéricos e falta de controle sobre os elementos individuais (POSTOLACHE, 2024). O Quadro 3 resume os principais pontos da comparação desta seção.

Quadro 3 – Comparação entre ferramentas de produção musical

	Flexibilidade de edição	Possui geração automática?	Renderiza a partir de
FL Studio	Alta	Não	Plugins e <i>samples</i>
SOUNDRAW	Baixa	Sim	Samples
AIVA	Média	Sim	Instrumentos baseados em <i>samples</i>
Suno	Nula	Sim	Sistema com aprendizado profundo
Sonhin	Alta	Sim	Sintetizadores modulares

Fonte: Autor

Considerando as ferramentas expostas e outras semelhantes é possível ver com clareza o diferencial visado pela para a ferramenta **Sonhin**: poder criar músicas com o aumento de produtividade oferecido por Inteligência Artificial, com total controle dos elementos musicais.

4 SOLUÇÃO PROPOSTA

Depois de muita pesquisa e experimentação nas iterações anteriores, chegou-se a algumas dificuldades de implementação que precisam ser apresentadas como contexto para a solução proposta.

Não existem grandes bases de dados abertas de músicas devido a direitos autorais, exceto por alguns gêneros específicos, como música clássica e música folclórica. Ainda que houvesse acesso a bases de dados, provavelmente seria em formato MIDI (como usado por AIVA), que não contém dados dos sons produzidos, ou em formato de áudio (como usado no Suno) cujo processamento é muito caro e demorado (TOKUI, 2020).

Diferente do caso no artigo “*Deep reinforcement learning from human preferences*”, comparação entre trechos curtos não foram suficientes como dados de treinamento, principalmente porque a música tem dependência temporal longa. Ou seja, mesmo uma música composta de trechos individuais bons é considerada ruim se as partes não forem semelhantes o bastante entre si.

Espera-se que músicas tenham elementos novos. Isso dificulta o uso direto de redes neurais, porque com poucos dados disponíveis para treinamento uma vez que tendem a ignorar os valores aleatórios de entrada e retornar saídas muito próximas umas das outras.

Além disso, como a música é geralmente produzida por humanos, faltam softwares capazes de renderizar e ajustar áudio a partir de código. Duas das melhores alternativas encontradas foram Fluidsynth (que não consegue lidar com efeitos adicionais como equalização, flanger, etc) e Ableton Live (que é pago e se mostrou difícil de integrar).

Tendo em vista todos esses pontos, a solução adotada foi a criação de uma DAW inteira com customizações para que funcione com IA. Isso permite que a coleta

de *feedback* utilize todas as alterações feitas por usuários, que os instrumentos virtuais sejam otimizados para semântica e alteração por algoritmos genéticos.

4.1 REQUISITOS FUNCIONAIS

Os requisitos funcionais essenciais, de o que o usuário pode fazer, são:

- Gerar uma música completa com apenas um ou dois cliques.
- Escutar o áudio da música renderizado em tempo real.
- Baixar uma música.
- Manter (ou seja, criar, ler, atualizar e apagar) os diferentes elementos que compõem uma música: notas, padrões, seções, camadas, sons, instrumentos e efeitos.
- Gerar alterações sugeridas por IA em qualquer um desses elementos.
- Manter *presets* de instrumentos, padrões e arranjos.
- Gerar *presets* automaticamente.
- Desfazer e refazer alterações.
- Mudar a ordem de efeitos.
- Copiar e colar efeitos.
- Selecionar um trecho, seção ou padrão para ser escutado em repetição.
- Desabilitar a saída de áudio de um instrumento (mutar).
- Desabilitar a saída de áudio de todos os outros instrumentos (solo).

Os requisitos de geração de conteúdo e alterações são os diferenciais propostos da DAW Sonhin. Os demais foram levantados através de observação direta de usuários em vídeos explicativos de como produzir música, na plataforma YouTube.

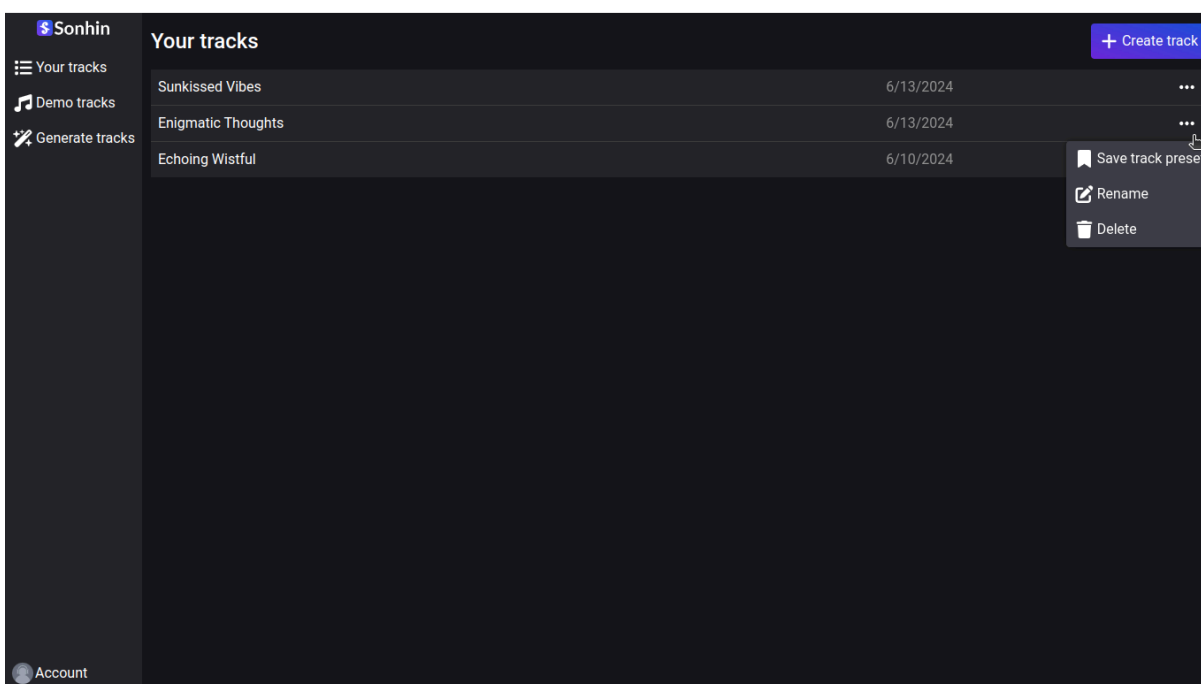
4.2 INTERFACE DO USUÁRIO

Essa seção detalha cada uma das páginas que compõem a interface de usuário.

4.2.1 Página de projetos

A página de projetos (ilustrada na Figura 1) permite ao usuário ver uma lista dos projetos que lhe pertencem. Clicar em um dos itens da lista leva à página de edição de padrões (que é uma das páginas do editor de música). Ainda nesta página é possível apagar, renomear e salvar uma música como *preset*.

Figura 1 - Página de Músicas

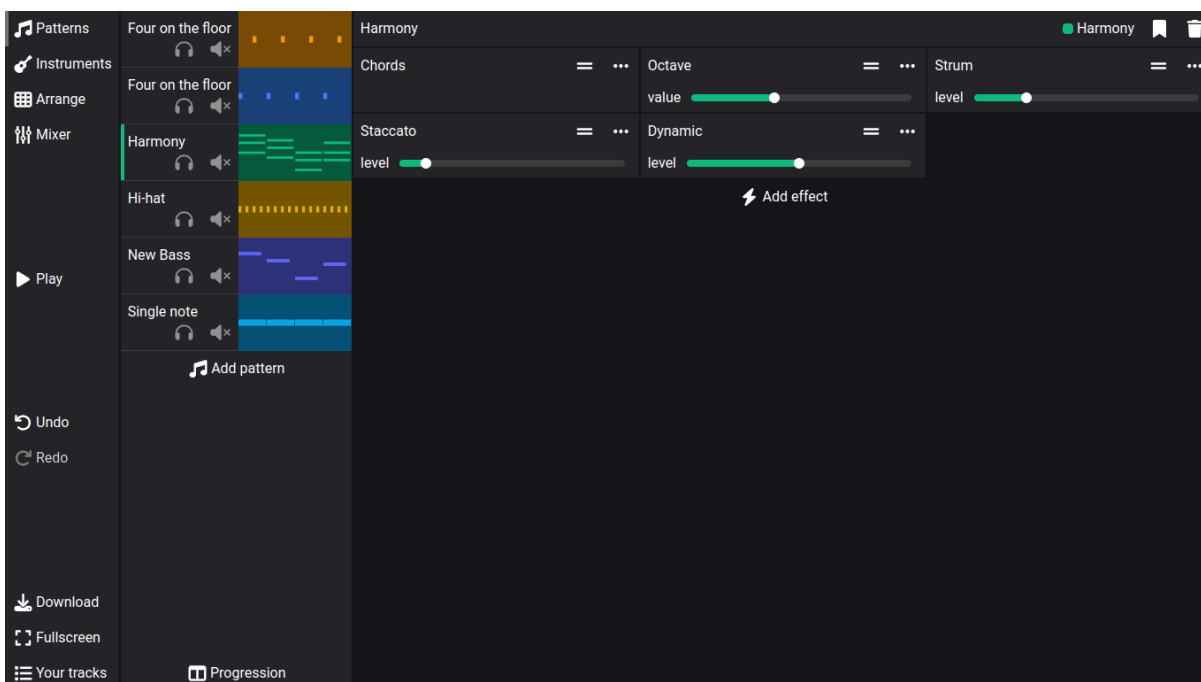


Fonte: Autor

4.2.2 Página de padrão

Quando um padrão é selecionado, a página correspondente (ilustrada na Figura 2) mostra a lista de efeitos, um botão para remover, outro para salvar como *preset*, um menu suspenso para escolher o canal, e um campo de texto para renomear.

Figura 2 - Página de Padrão

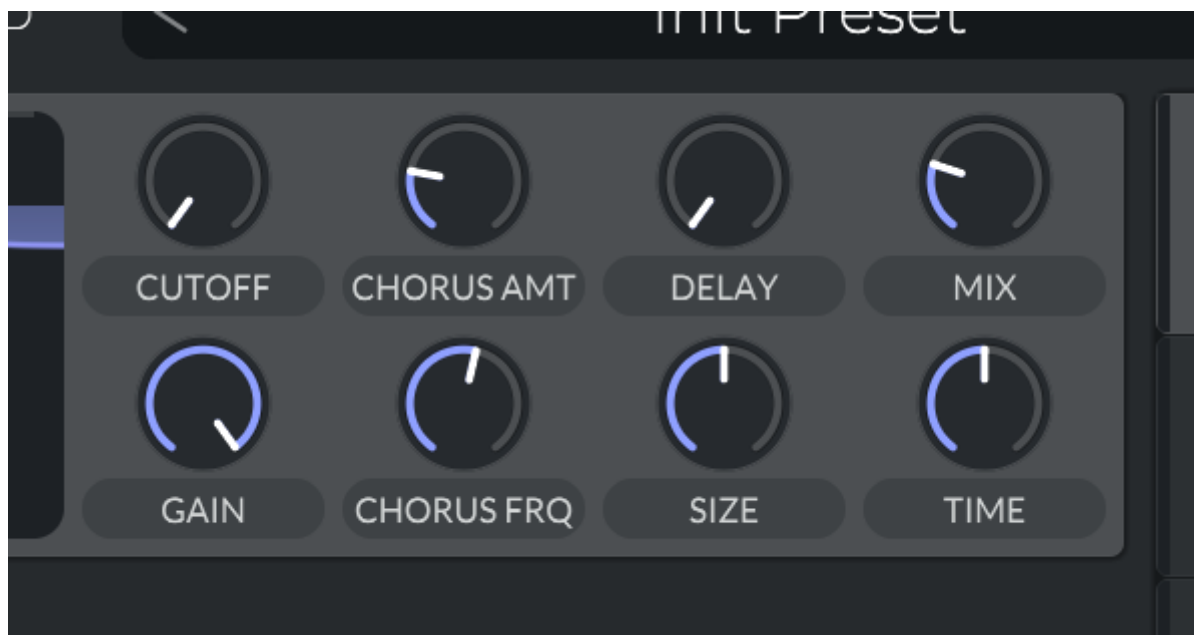


Fonte: Autor

Os parâmetros de cada efeito podem ser modificados usando controles deslizantes. Estes controles possuem semântica pré-programada para cada efeito, permitindo ajustes de valores como inteiros ou de ponto flutuante, de forma linear ou exponencial, e com valores máximos e mínimos específicos.

Diferente de grande parte das aplicações de áudio não foram utilizados controles conhecidos como *knobs* (ilustrados na Figura 3), pois não são intuitivos (LAUBHEIMER, 2017) (como confirmado por testes de usabilidade em versões anteriores). Além disso, esse tipo de controle se mostra difícil de montar em um *layout* sempre coerente e apresenta área reduzida para o texto de legenda.

Figura 3 - Knobs em uma aplicação de áudio (Vital)

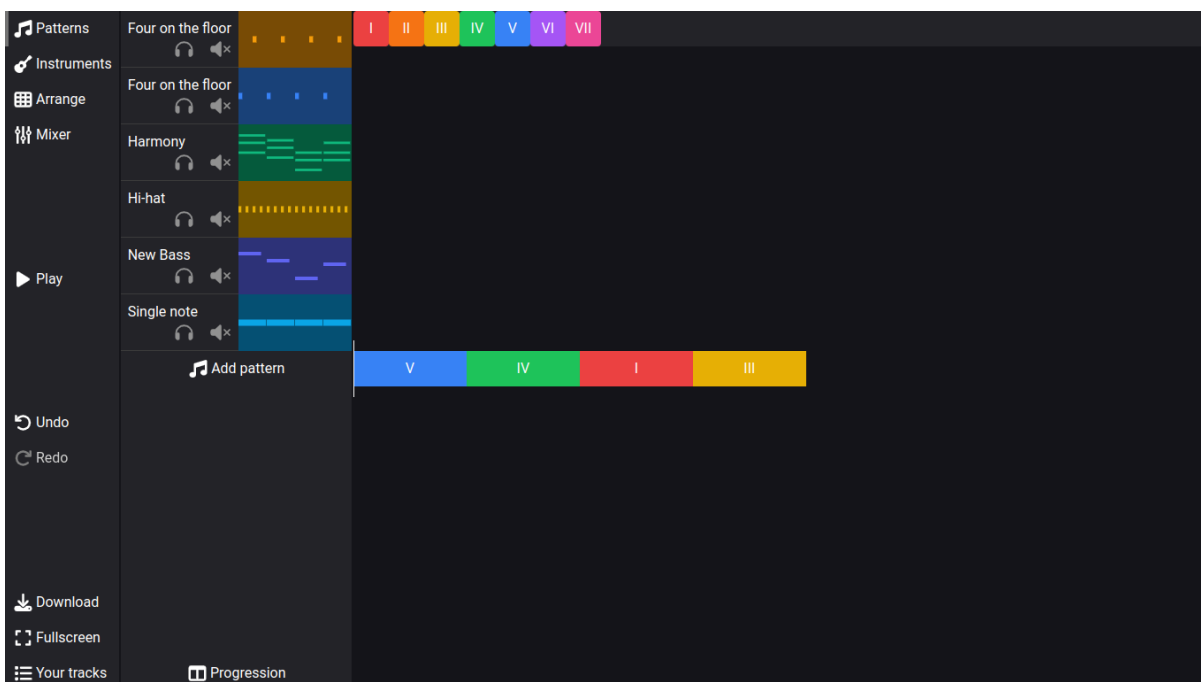


Fonte: Autor

4.2.3 Página de progressão harmônica

A página de progressão harmônica (ilustrada na Figura 4) permite editar a progressão harmônica usada na música. Atualmente o sistema só funciona com uma progressão harmônica para a música inteira, mas isso deve ser resolvido em versões futuras.

Figura 4 - Página de Progressão



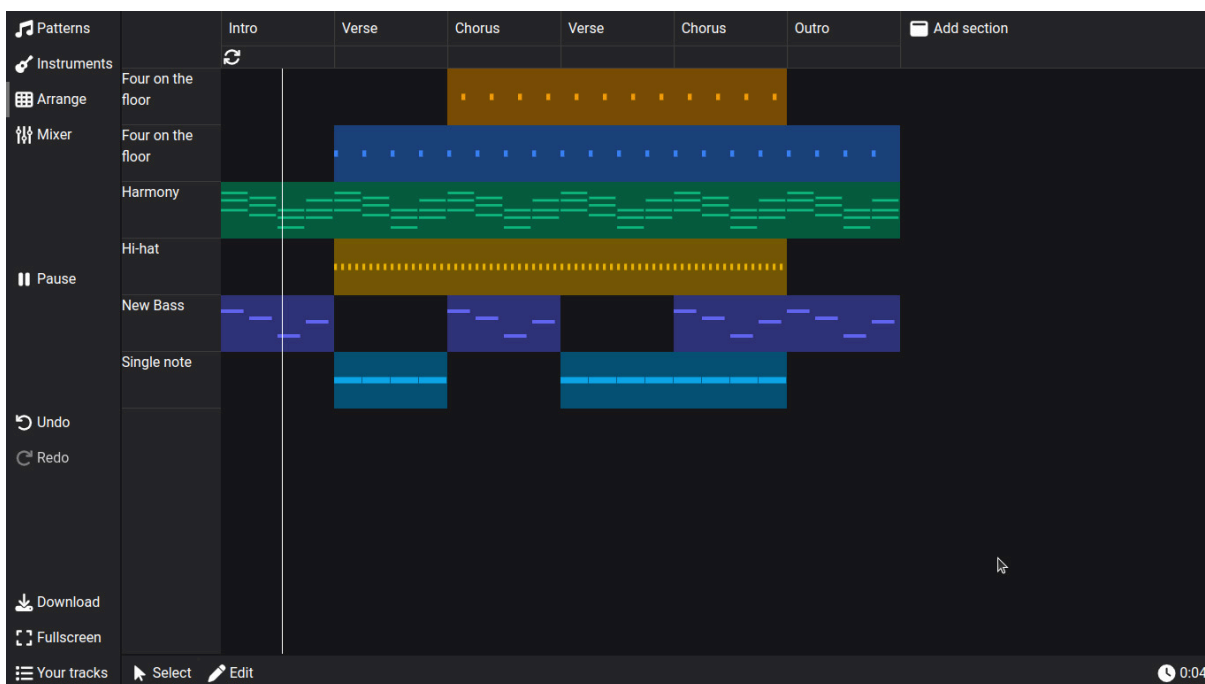
Fonte: Autor

4.2.4 Página de arranjo

Na página de arranjo (ilustrada na Figura 5) é possível ajustar a estrutura da música como um todo. Na parte superior fica a lista de seções e um seletor de repetição, na parte lateral a lista de padrões, e no centro os cliques dispostos verticalmente por padrão e horizontalmente por instante de início.

Na parte inferior encontra-se um indicador do tempo atual na música, e um seletor de modos. No modo de seleção, clicar em um clipe o seleciona e clicar em um espaço vazio altera o tempo atual na música. Já no modo de edição, clicar em um clipe o apaga e clicar em um espaço vazio cria um clipe com início e padrão correspondentes a posição do clique.

Figura 5 - Página de Arranjo

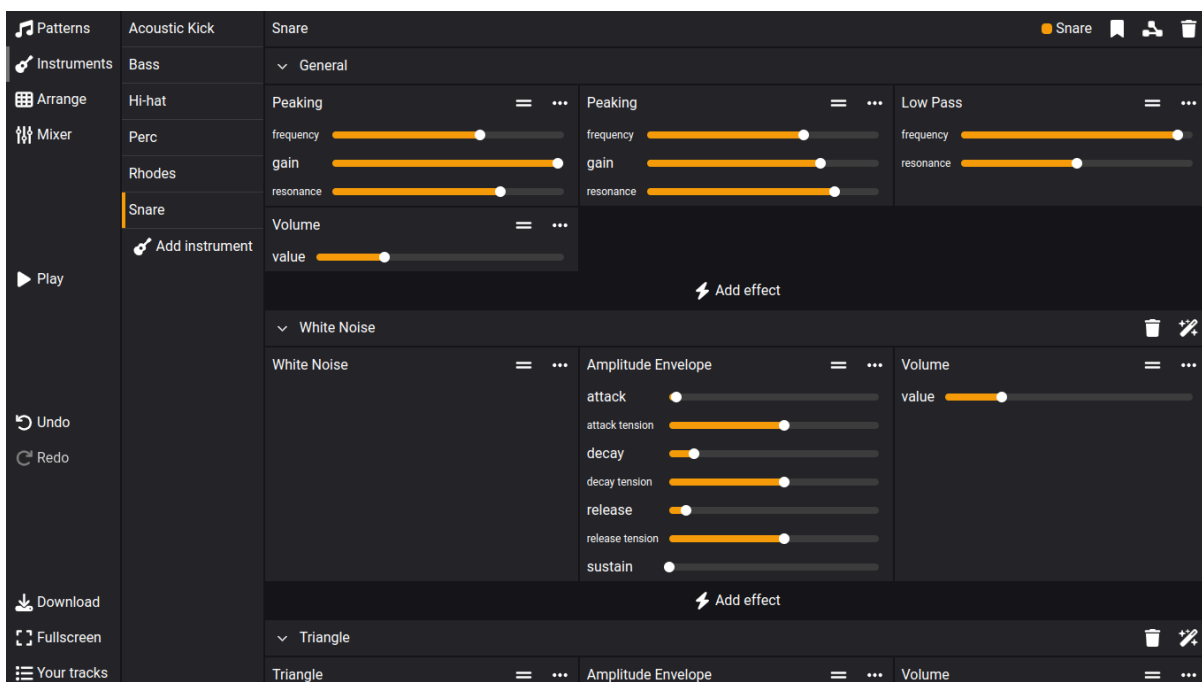


Fonte: Autor

4.2.5 Página de instrumento

A página de instrumento (ilustrada na Figura 6) é semelhante a página de padrões, mas também inclui seções para cada som do instrumento (que possui sua própria lista de efeitos), e um botão para mostrar o grafo do instrumento.

Figura 6 - Página de Instrumento



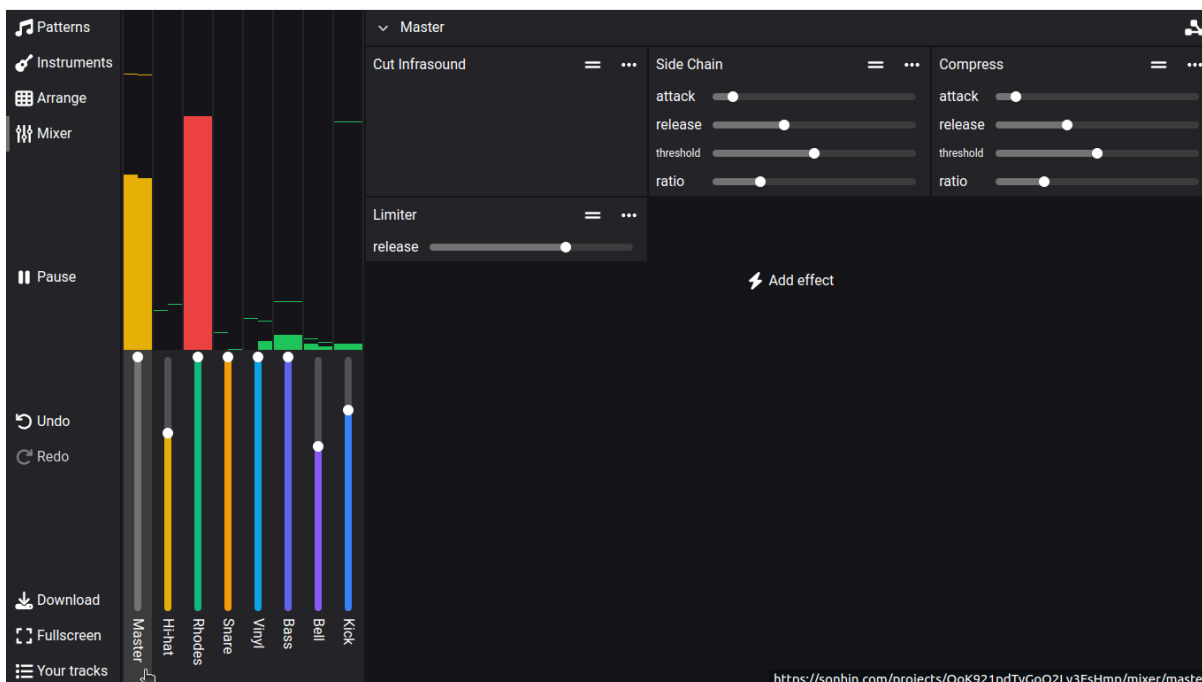
Fonte: Autor

Ao clicar no botão de adicionar efeito, é mostrado um menu suspenso com as opções de efeitos. Alguns efeitos só podem ser inseridos em sons (por dependerem de dados da nota a ser tocada), enquanto outros só podem ser inseridos no instrumento como um todo (devido a limitações de performance).

4.2.6 Página de mixagem

A página de mixagem (ilustrada na Figura 7) contém indicadores e controles de intensidade das saídas de áudio dos instrumentos e da master. Também permite modificar os efeitos e conferir os grafos usados nessas saídas.

Figura 7 - Página de Mixagem



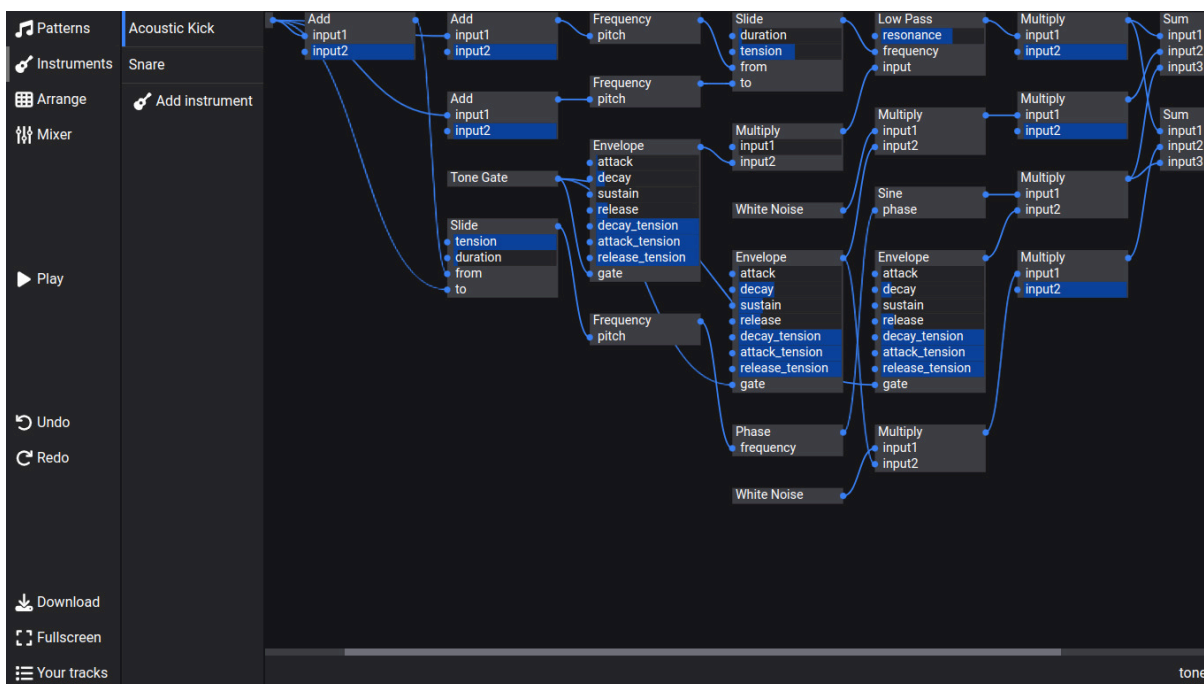
Fonte: Autor

Os indicadores de intensidade consistem em barras verticais coloridas cuja altura equivale à intensidade. No topo dessas barras, linhas horizontais indicam a intensidade máxima dentro de um intervalo de tempo. As barras ficam amarelas quando a intensidade está próxima de passar do limite, e vermelhas quando a intensidade passa do limite.

4.2.7 Páginas de grafos

As páginas de grafos (ilustrada na Figura 8) permitem visualizar os grafos de determinados elementos. Em versões futuras, será possível editar os grafos diretamente, expandindo a gama de possibilidades para os sons e permitindo a criação de novos efeitos sem a necessidade de programação.

Figura 8 - Página de Grafo

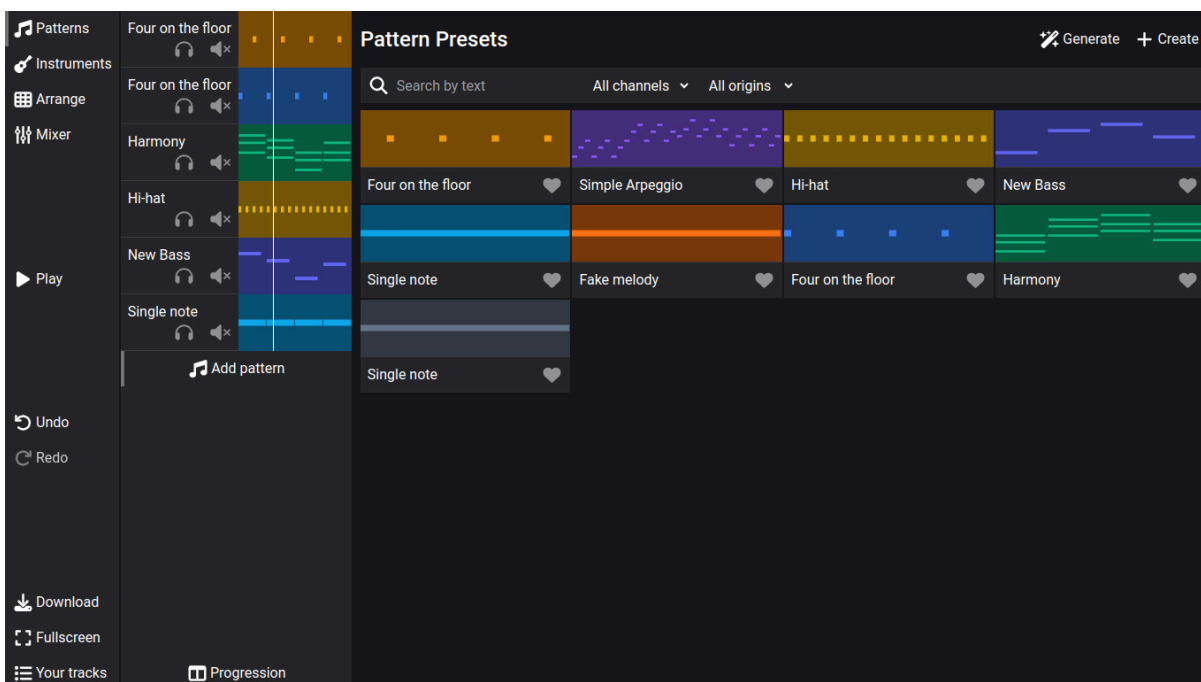


Fonte: Autor

4.2.8 Páginas de *presets*

Para cada tipo de elemento que pode ser salvo como *preset*, há uma página de pesquisa para os *presets* criados por usuários e outra página destinada à geração automática de novos *presets*. A Figura 9 ilustra a página de pesquisa de *presets* de padrões.

Figura 9 - Página de *Preset*

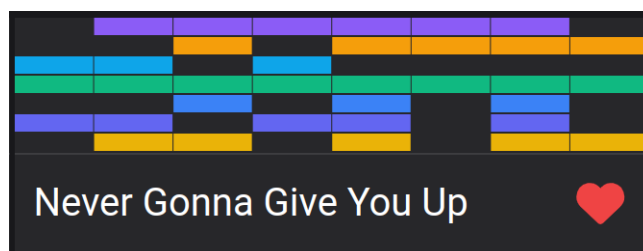


Fonte: Autor

Para cada *preset*, são exibidos o nome, um botão de curtir e uma prévia. A prévia pode ser ouvida colocando-se o cursor sobre sua área. Quando uma prévia para de tocar, o sistema salva os dados da prévia como visualização para servir de *feedback*.

Diferentes elementos podem ter diferentes tipos de prévias. Por exemplo, uma prévia de música (ilustrada na Figura 10) mostra clipes em vez de notas.

Figura 10 - Prévia de música



Fonte: Autor

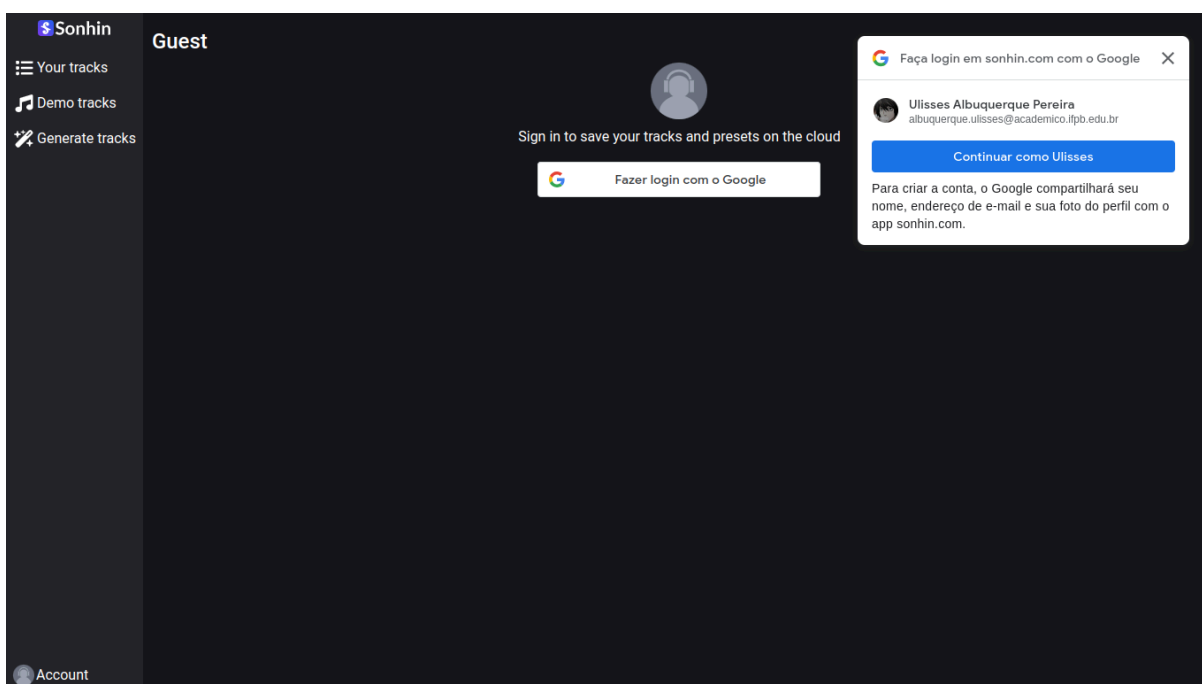
As páginas de gerar novos *presets* são praticamente idênticas às de pesquisa, mas mostram resultados criados automaticamente sob demanda.

4.2.9 Página de conta

A página de conta (ilustrada na Figura 11) mostra o usuário atual, que pode ser do tipo visitante ou cadastrado. Um usuário visitante é criado automaticamente ao abrir o site e pode realizar as mesmas ações de um usuário cadastrado, mas perde sua autenticação caso os dados do navegador sejam apagados.

Para garantir o acesso, basta que o usuário crie uma conta usando o login com o Google. Em comparação com a forma tradicional de se usar email e senha, esse fluxo é mais seguro, dificulta a criação de contas cadastradas por robôs e tem consideravelmente menos atrito. O login pode até ser feito automaticamente pelo Google, sem cliques na página, caso o usuário já tenha feito o cadastro uma primeira vez.

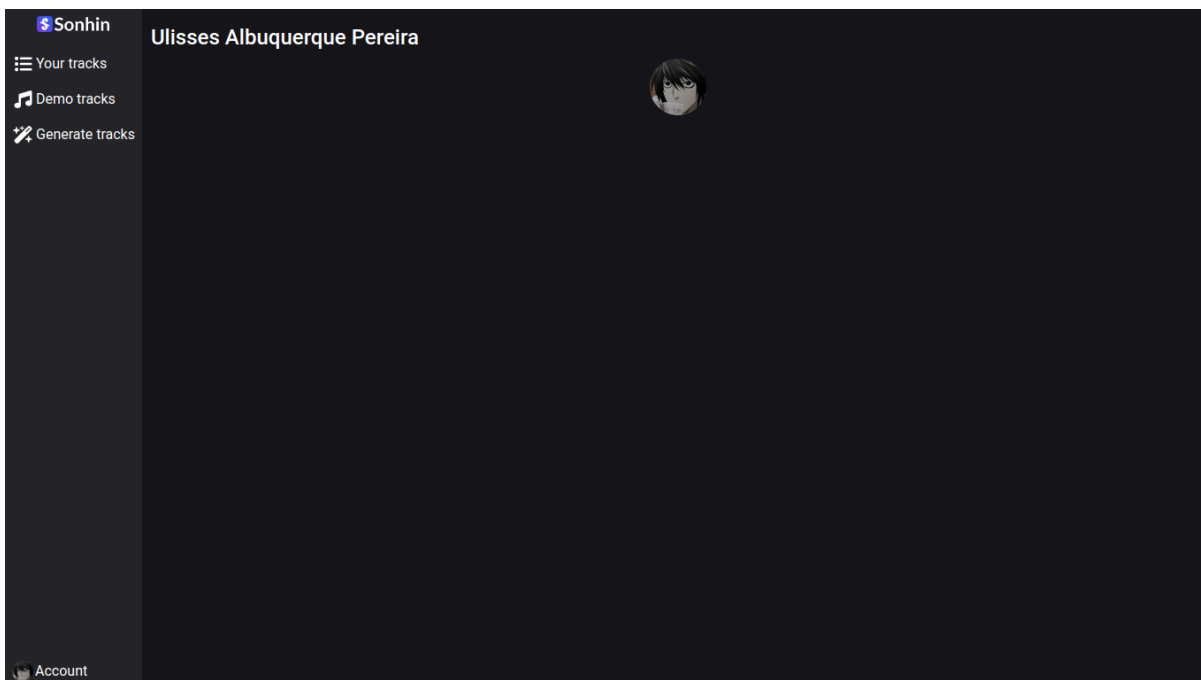
Figura 11 - Página de Conta de Visitante



Fonte: Autor

Uma vez cadastrado, o usuário pode editar sua foto, nome e fazer *signout* na página de conta (Figura 12).

Figura 12 - Página de Conta Cadastrada



Fonte: Autor

4.3 MODELAGEM DOS DADOS

Essa seção detalha os princípios seguidos na modelagem de dados, dados salvos e dados calculados. Os dados salvos são aqueles que são armazenados diretamente no banco de dados. Já os dados calculados são aqueles que são montados sob demanda em tempo de execução, a partir dos dados salvos.

Os dados devem ser auto descritivos, para facilitar a manutenção e a atualização dos dados. Em algumas iterações anteriores do projeto, as músicas eram salvas diretamente como matrizes numéricas, tornando-se excessivamente acopladas ao modelo utilizado para gerá-los.

Deve-se prezar por interfaces simples, mesmo que isso possa resultar em informações incompletas. Por exemplo, espera-se que toda música tenha um andamento, que é a velocidade em que as notas devem ser tocadas. Uma possível abordagem seria adicionar o atributo "andamento" diretamente à música. No entanto, isso complicaria o cálculo das notas, a ordem de execução dos efeitos, a associação de metadados e os algoritmos de geração. Portanto, o andamento é implementado como apenas mais um efeito, embora semanticamente seja um

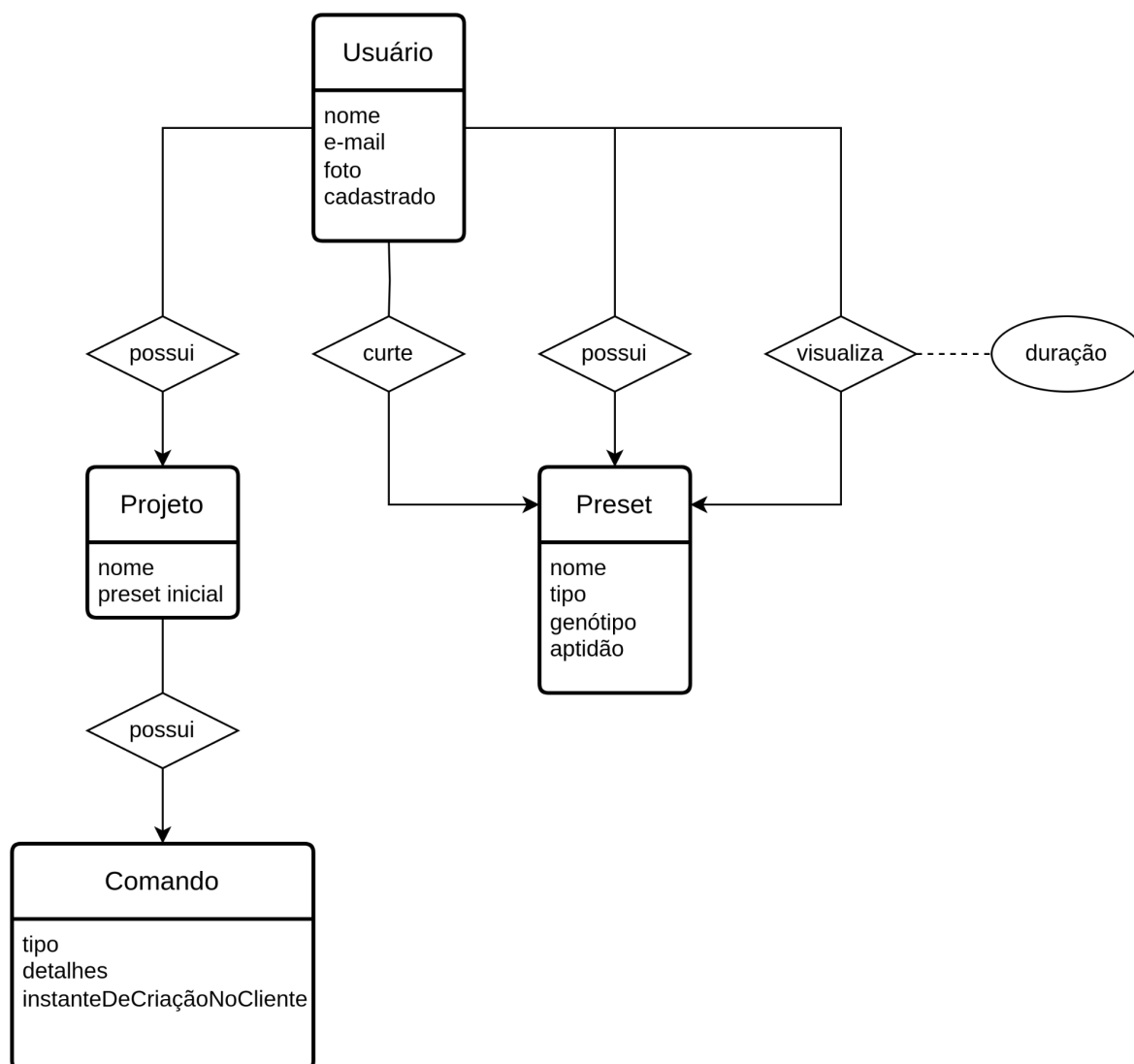
atributo de música. Essa abordagem reduz a certeza de que toda música tenha um andamento definido, mas parece ser um *trade-off* aceitável.

Todas as entidades, tanto salvas como calculadas, devem ter como chave primária um valor gerado aleatoriamente, chamado “id”. Isso permite que tanto o lado do servidor como o do cliente gerem identificadores, removendo a necessidade de aguardar a resposta do servidor para fazer alterações que referenciam essa entidade. A padronização desse atributo também facilita o desenvolvimento, uma vez que não é necessário lembrar de nomes específicos de chave primária ou quais atributos compõem uma chave composta.

Por fim, os dados devem ser salvos de maneira que seja possível calcular o estado de uma música em qualquer momento passado. Essa característica permite aos usuários desfazerem e refazerem alterações e é uma das bases para treinar o sistema de recomendação de alterações. Para tanto, foi utilizada a técnica de *Event Sourcing*, que consiste em salvar uma entidade não como o registro de seu estado atual, mas sim como o conjunto de todas as alterações (embora o estado atual possa ser salvo como *cache* por questões de performance).

A Figura 13 ilustra o diagrama de entidade e relacionamento dos dados salvos no banco de dados.

Figura 13 - Diagrama entidade-relacionamento do banco de dados



Fonte: Autor

Um comando é o registro de uma alteração. Seu nome advém do padrão de projeto *Command*. Ele possui tipo, detalhes e instante de criação no cliente. O tipo se refere a qual das possíveis alterações, por exemplo “Remover instrumento” ou “Adicionar seção”. Detalhes é um objeto com formato variável, que contém os dados necessários para executar a alteração. O instante de criação do cliente é armazenado para garantir a correta ordem de execução dos comandos mesmo que o envio de dados ao servidor seja feito na ordem errada.

Projeto é a entidade que guarda os dados necessários para se calcular uma música a partir do *preset* inicial e o histórico completo de alterações (comandos que

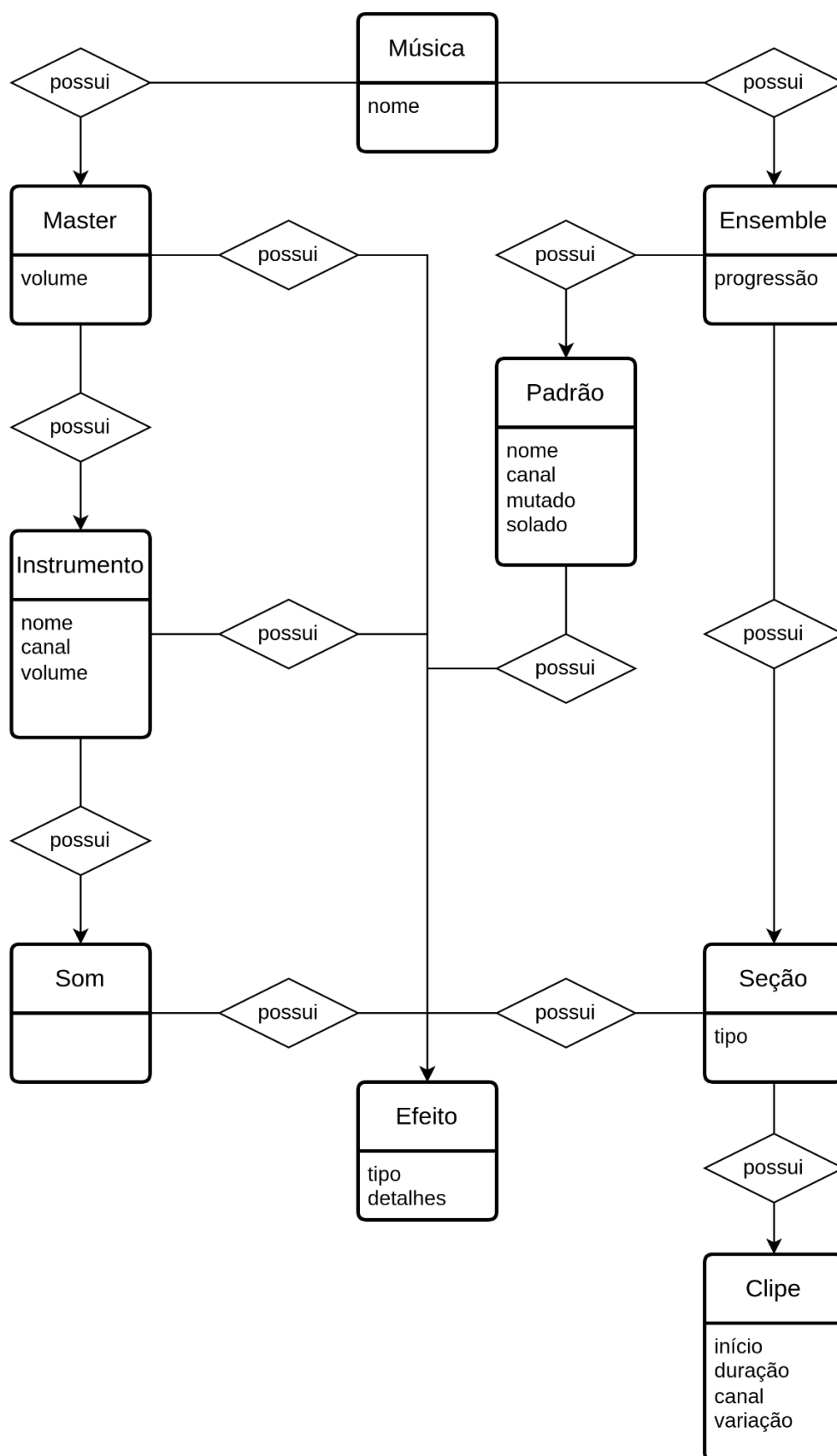
possui). Também contém o valor derivado “nome”, que permite mostrar o nome da música sem calculá-la novamente.

Um *preset* possui nome, tipo, genótipo e aptidão. O tipo refere-se ao tipo de dado guardado, por exemplo “Instrumento” ou “Música”. O genótipo é o conjunto de dados necessários para instanciar o objeto, em um formato próprio para algoritmos genéticos. E aptidão é um valor número que representa o engajamento recebido. A aptidão pode ser calculada sob demanda a partir de visualizações e curtidas, mas é salvo por questões de performance e para desacoplar essas informações do sistema de geração de novos *presets*.

Usuário possui os atributos nome, e-mail, foto e “é cadastrado”. Esse último é um valor booleano que indica se o usuário possui um e-mail. É utilizado principalmente por questões de segurança para não revelar o e-mail do usuário. Além disso, ajuda na expressividade do código, já que é mais natural pensar em usuários como cadastrados ou visitantes, em vez de com ou sem e-mail. Mesmo sendo um valor derivado, é de fácil manutenção, porque é alterado no máximo uma vez, no momento do cadastro do usuário.

A Figura 14 ilustra o diagrama de entidade e relacionamento dos dados calculados, que são explicados a seguir.

Figura 14 - Diagrama de entidade e relacionamento



Fonte: Autor

Um efeito possui tipo e detalhes. Tipo é uma string que representa o tipo do efeito, por exemplo “Reverberação” ou “Saturação”. E detalhes é um objeto com formato variável, que contém os dados necessários para executar o efeito.

Canal é um campo presente tanto em instrumento, padrão e clipe. Trata-se de uma forma de relacionar esses diferentes tipos de objetos de forma mais semântica do que com IDs. O uso desse dado permite associar automaticamente diferentes elementos criados a partir de *presets*, colorir partes da aplicação de forma correspondente e intuitiva, e facilitar o desenvolvimento de algoritmos que combinam diferentes *presets*. Para selecionar diferentes padrões em um mesmo canal se usa o valor numérico “variação”.

Uma música possui nome, uma *master* e um *ensemble*. A *master* e o *ensemble* são abstrações que servem para desacoplar a geração de notas e grafos. A *master* possui instrumentos, efeitos e volume, enquanto que o *ensemble* possui progressão, padrões e seções.

Um instrumento possui sons, efeitos, volume, canal e nome.

Um som possui apenas efeitos.

Um padrão possui efeitos, canal, nome e se é solado e/ou mutado.

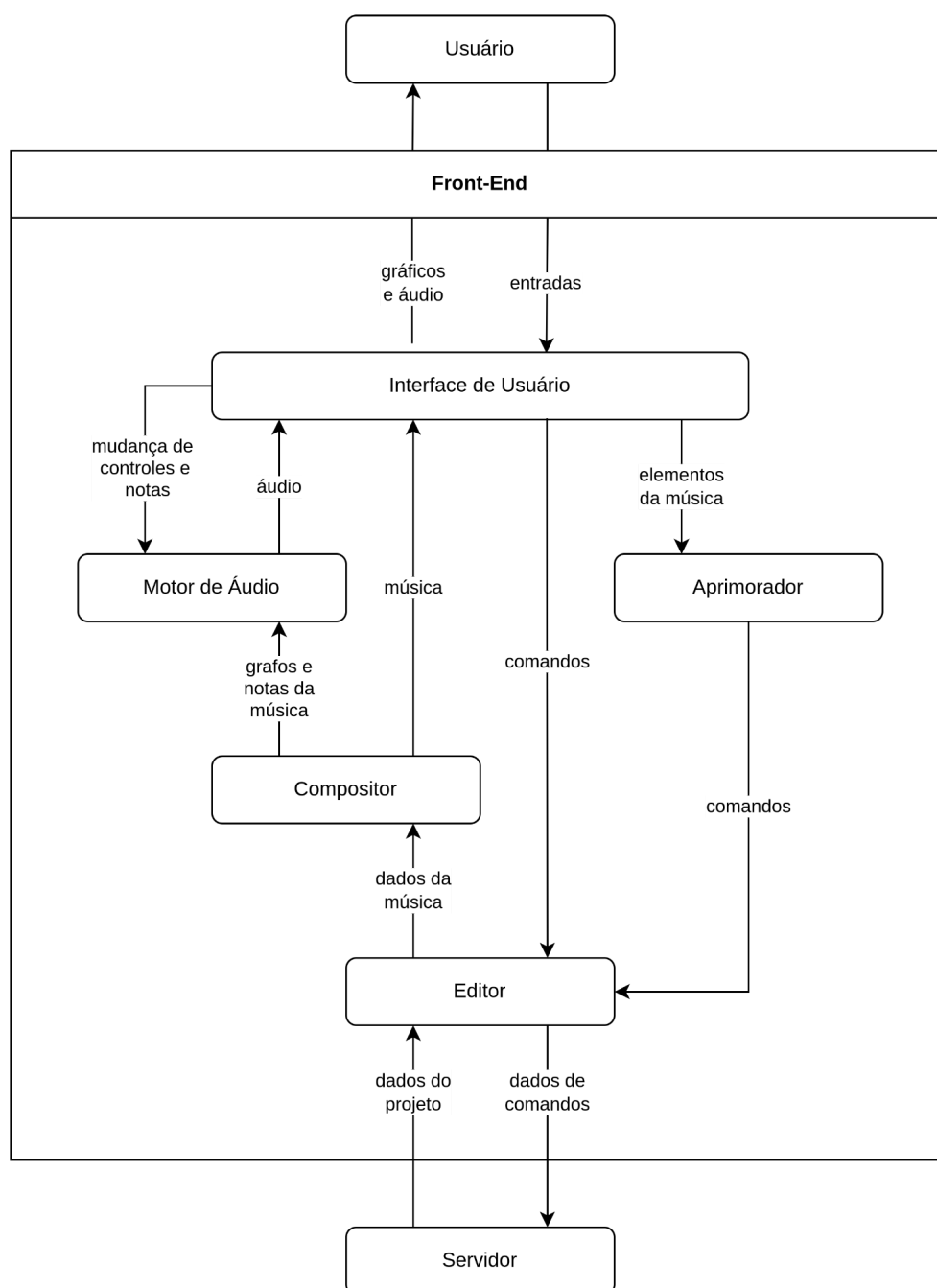
Um clipe possui (instante de) início, duração, canal e variação.

Uma seção possui efeitos, um tipo (como “Verso” ou “Refrão”) e clipes.

4.4 ARQUITETURA DO SISTEMA

A arquitetura do sistema pode ser dividida em *front-end* (executado no lado do cliente) e *back-end* (executado no lado do servidor). O *front-end* é composto por cinco módulos, conforme mostrado na Figura 15.

Figura 15 - Arquitetura do front-end



Fonte: Autor

A interface de usuário permite visualizar e editar os dados da aplicação. É através dela que o usuário pode realizar ações como remover um instrumento ou pedir uma sugestão à IA.

O motor de áudio é responsável por renderizar o áudio da música através de grafos e notas. Também recebe, diretamente da interface de usuário, notas do

teclado musical e ajustes de controles durante a edição de efeitos de áudio e mixagem.

O compositor é o módulo que compõe os vários elementos da música a partir de seus dados planos. Esse processo inclui substituir chaves estrangeiras por cópias independentes dos elementos referenciados. É nesse módulo que notas e grafos são calculados a partir dos dados de efeitos.

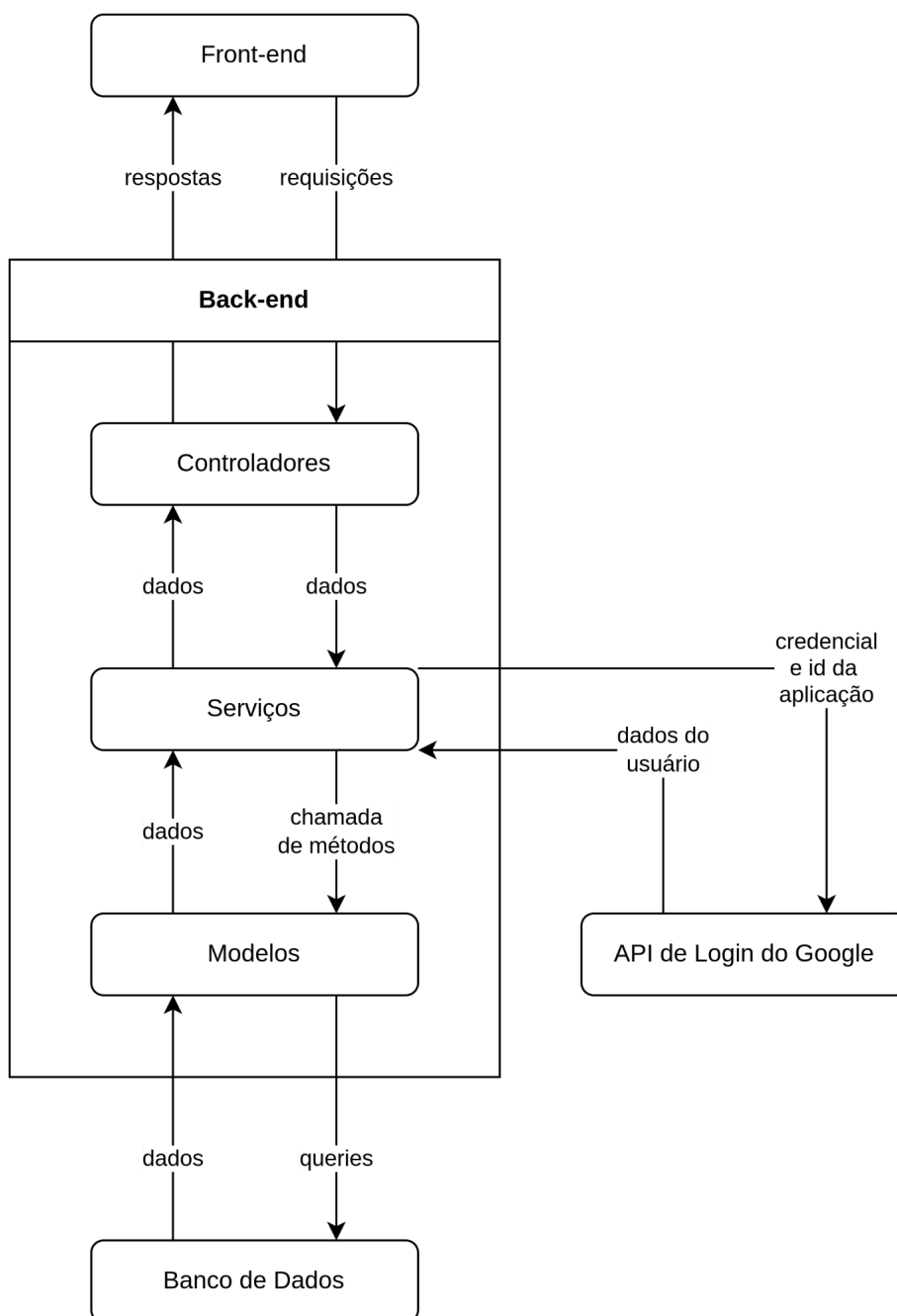
O editor transforma os dados do projeto em dados da música, a partir de todo o histórico de alterações e dos dados do *preset* de música utilizado para a criação do projeto.

Por fim, o aprimorador é o módulo que gera sugestões para a música através de redes neurais. Ele recebe o estado atual da música e um elemento alvo, simula alterações válidas para esse elemento, tenta prever qual delas seria a preferida, e a aplica à música.

Já o *back-end* é composto de três processos que podem ser executados separadamente: o servidor da aplicação, o treino do avaliador de sugestões e o *seeder* de desenvolvimento.

O servidor da aplicação, representado na Figura 16, é responsável pela persistência dos dados, autenticação, autorização e geração de *presets*. É organizado em controladores, serviços e modelos. Os controladores recebem requisições HTTP e extraem as informações necessárias, os serviços executam a lógica de negócio, e os modelos manipulam os dados no banco de dados.

Figura 16 - Arquitetura do servidor da aplicação



Fonte: Autor

Por exemplo, ao apagar um projeto, o controlador extrai o ID do projeto do corpo da requisição e o ID do usuário a partir do cabeçalho de autenticação. Esses dados são então passados para o serviço de apagar um projeto, que permanece desacoplado da origem dos dados fornecidos. O serviço procura pelo projeto, verifica se o ID do usuário corresponde ao do proprietário do projeto e, em seguida, apaga o projeto do banco de dados. Para buscar e apagar o projeto no banco de

dados, o serviço utiliza o modelo de projeto, que é declarado separadamente e não depende de como é utilizado. Em algumas tarefas, o serviço também pode retornar dados que o controlador usa para compor a resposta à requisição.

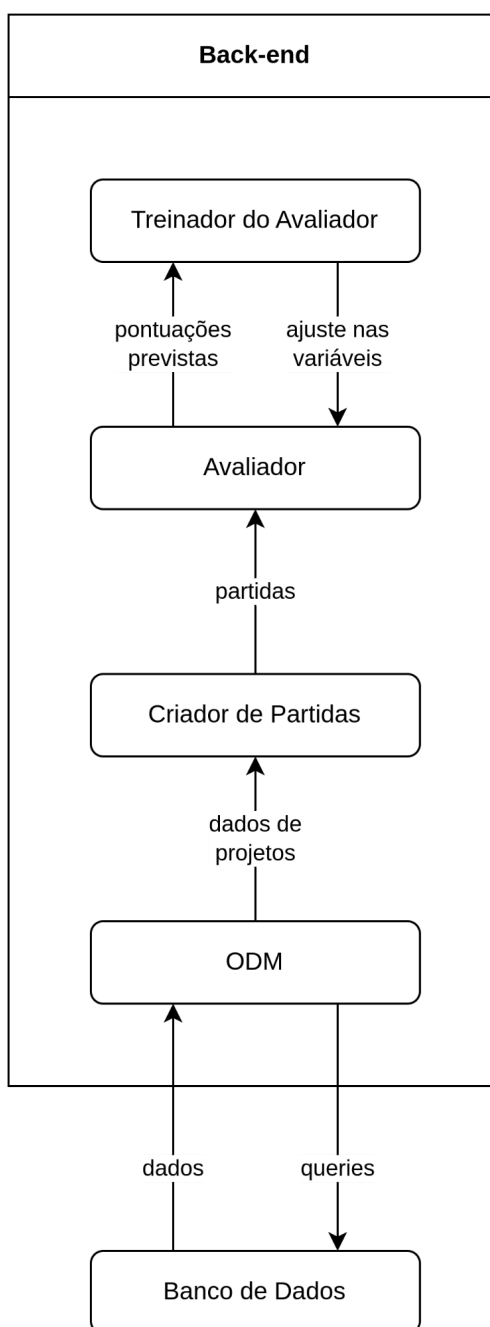
A API de login do Google¹ é utilizada especificamente nos serviços de autenticação e cadastro de usuários. Sua função é autenticar o usuário e fornecer informações básicas como nome, email e foto de perfil. Para isso, uma credencial é obtida no frontend através de um botão "Fazer Login com o Google", repassada ao back-end, e enviada à API (junto ao ID da aplicação). O Google então responde os dados do usuário ou um erro de autenticação.

Outro serviço que cabe ser citado é de gerar *presets*. É consideravelmente mais complexo que outros serviços já que realiza as etapas de algoritmos genéticos. Nele os *presets* são selecionados a partir de uma soma ponderada de curtidas, duração de visualizações, e a quantidade de vezes que o *preset* foi escolhido pelos usuários. As cópias desses *presets* são então recombinados, mudando parte dos efeitos umas com as outras. Depois os valores de efeitos são normalizados conforme a sua semântica pré-programada, recebem mutações (adição de valores aleatórios), e por fim desnormalizados.

O treino do avaliador, ilustrado na Figura 17, é outro processo do backend, responsável por treinar o modelo de rede neural (nomeada avaliador) utilizada no aprimorador no *front-end*. O avaliador tem como entrada os dados de uma música, e tem como saída a um número previsto de quão preferível é essa música para humanos.

Figura 17 – Arquitetura do treino do avaliador

¹ <https://developers.google.com/identity>



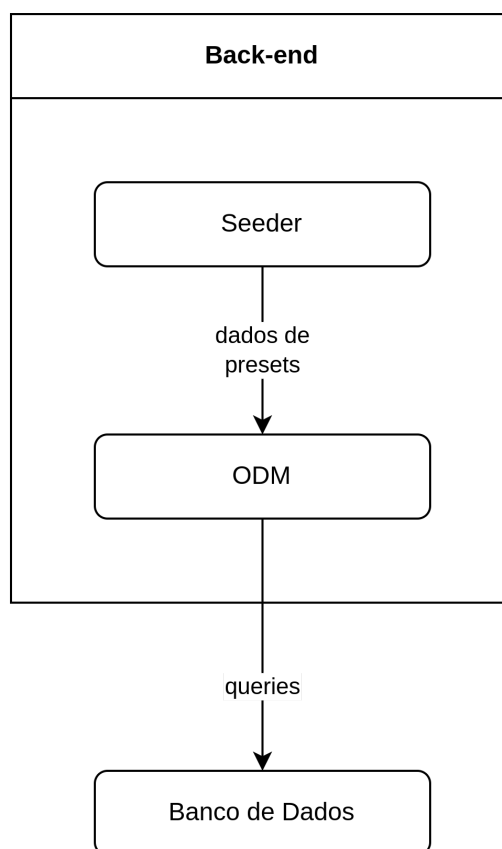
Fonte: Autor

Para treinar o avaliador, esse trabalho assume que um ser humano altera uma música nas páginas de edição conforme suas preferências, mantendo as alterações boas e descartando as alterações ruins. É de se esperar então que a versão de uma música, com uma ou mais alterações não desfeitas, seja preferida às versões anteriores. Essa característica é modelada em um dado chamado partida (como em “partida de esporte”) que é composto por duas músicas e um placar. Esse

placar é uma tupla do tipo (1, 0) caso a primeira música seja preferida e (0, 1) caso a segunda música seja preferida. O treinador do avaliador então utiliza um conjunto de partidas e o registro de todas as operações feitas no avaliador para calcular ajustes nas variáveis do avaliador através de retropropagação de erro.

Por fim, o *back-end* inclui o processo de *seeding* do banco de dados (Figura 18). Esse processo adiciona conteúdo de teste usado durante o desenvolvimento do sistema. Além disso, a forma como o *seeding* foi implementado permite detectar erros e ajuda na criação de migrações de dados. Isso acontece porque os dados do *seeding* estão escritos em linguagem de programação, passível de análise estática.

Figura 18 – Arquitetura do seeder de dados



Fonte: Autor

4.5 TECNOLOGIAS UTILIZADAS

Nessa seção são descritas as tecnologias usadas na implementação do sistema. Em todas as partes exceto no motor de áudio a linguagem de programação utilizada foi TypeScript, devido a sua versatilidade, verificação estática de código, e amplo ecossistema.

4.5.1 *Front-end*

Nessa iteração, o *front-end* foi desenvolvido em Svelte², diferentemente das iterações anteriores, que usavam React³. Essa mudança foi feita principalmente devido a vantagens de legibilidade, gerenciamento de estado global e integração com o *back-end*.

Svelte funciona com um compilador. Isso permite criar componentes reativos de forma mais transparente, com uma sintaxe simples bem próxima de HTML, CSS e JavaScript puros. Enquanto que com React, em geral seria necessário declarar uma função para cada componente e utilizar *hooks* e *side effects*.

Para o gerenciamento de estado global, Svelte conta com *stores*, que armazenam valores de forma bem granular e, diferentemente de contextos do React, não requerem *wrappers* para os componentes e podem ser tipados por inferência. As *stores* também podem ser derivados de outras *stores* como apenas leitura, o que facilita referenciar dados muito aninhados (como os de música).

Além disso, Svelte tem uma melhor integração com tecnologias web fora do próprio ecossistema, uma vez que não depende de DOM virtual. Isso facilitou a implementação de alguns componentes que utilizam *requestAnimationFrame* para atualizar os dados em tempo real.

4.5.2 Interface Gráfica

A interface gráfica utiliza para estilização o framework Tailwind CSS⁴, que funciona com classes prontas e específicas para a maior parte dos atributos. Isso permite estilizar de forma mais direta, a nível de elementos, melhorando a experiência de desenvolvimento, rapidez de programar e evita que regras gerais do CSS alterem outros elementos por engano.

² <https://svelte.dev>

³ <https://react.dev>

⁴ <https://tailwindcss.com>

Além disso, esse framework CSS permite que classes herdem propriedades de outras classes. Por exemplo, na aplicação existe a classe para botões comuns e a classe para botões com ícones (que também são botões comuns), como mostrado em Código 1:

Código 1 – Classe derivada com Tailwind CSS

```
.common-button {
  @apply rounded p-2 gap-0.5;
}
.common-button:not(:disabled) {
  @apply hover:bg-white/10;
}

.icon-button {
  @apply common-button;
}
.icon-button:active:not(:disabled) .svelte-fa {
  @apply scale-75 duration-75;
}
```

Fonte: Autor

Por fim, Tailwind CSS não ocupa muito espaço em produção (já que classes que não forem usadas são removidas automaticamente na etapa de *build*), ajuda na padronização de estilo e possui modificadores que facilitam manter a página responsiva sem *media queries* manuais.

4.5.3 Back-end

O *back-end* utiliza SvelteKit⁵, que é o framework oficial do Svelte para aplicações. Ele possui uma série de funcionalidades que surgem da sua abordagem opinada e uso de compilador, listadas a seguir.

O framework possibilita montar as páginas tanto no lado do servidor como no lado do cliente. Isso permite usar a mesma *codebase* para criação de páginas dinâmicas (que permitem maior interatividade) e estáticas (muito usadas para tornar o conteúdo indexável por buscadores).

⁵ <https://kit.svelte.dev/>

Essa versatilidade é usada por padrão para reduzir o tempo de espera do usuário: Quando um usuário abre o site, a primeira página é montada no lado do servidor já com os dados, cortando o tempo que seria necessário entre carregar a página e carregar os dados via JavaScript. E durante a navegação o framework carrega apenas os recursos ausentes (mantém o bundle Javascript e CSS da primeira solicitação). Além disso, por padrão os dados são pré-carregados quando o usuário move o cursor para a área de um link.

Além disso, o carregamento de dados pelo *front-end* possui várias vantagens: O *framework* elimina a necessidade da maior parte das requisições manuais com JavaScript, possui tipagem automática das respostas do *back-end* para o *front-end*, e conta com uma semântica simples para mostrar elementos em página de acordo com o estado de *promises*.

Graças a essas vantagens do *framework*, da geração de áudio ser do lado do cliente e do uso *Event Sourcing*, o servidor da aplicação é consideravelmente simples.

4.5.4 Motor de áudio

A implementação do motor de áudio foi realizada utilizando a linguagem de programação Rust⁶. A escolha dessa linguagem foi baseada em vários fatores significativos. As principais foram performance e integração com a web: Rust pode ser compilado para WebAssembly (Wasm), permitindo que seja executado em navegadores web com uma performance quase igual à das aplicações nativas.

O gerenciamento de memória em Rust também foi um aspecto determinante para sua escolha. Diferente de outras linguagens que conseguem lidar com baixo nível, como C e C++, em Rust não há a necessidade de alocação e desalocação manual de memória, que é associada a erros difíceis de detectar e resolver. Ao mesmo tempo que Rust não possui *garbage collector*, evitando assim uma latência periódica percebida como "cliques" na saída de áudio.

Outro fator levado em conta foi a experiência do desenvolvedor. Rust tem se destacado nesse aspecto, conquistando o primeiro lugar como a linguagem mais admirada na enquete de desenvolvedores do Stack Overflow de 2023 – quando o

⁶ <https://www.rust-lang.org>

termo “admirada” começou a ser usado – e de mais amada nas sete edições anteriores.

Também foram testadas outras opções para a implementação do motor de áudio. Uma delas foi a utilização de áudios gravados como sons para as notas. No entanto, essa abordagem foi descartada devido à complexidade de lidar com diversos arquivos de áudio para cada instrumento, ajustando-os com efeitos de áudio e combinando-os para polifonia e mixagem.

Outra possibilidade considerada foi programar o motor de áudio utilizando o framework TensorFlow em sua versão para JavaScript⁷. Esse framework permite executar código com um alto grau de paralelismo, graças ao uso de GPU. Essa opção funcionou bem para algumas partes, como a geração de ondas simples, envelopes e síntese aditiva. Contudo, mostrou-se inviável para muitos efeitos que dependem de processamento sequencial e linear, como filtros de frequências e reverberação. Além disso, apresentou sérios problemas de performance, exigindo muita memória e interrompendo a saída de áudio devido à latência gerada a cada execução do *garbage collector*.

Por fim, foi testada a geração de áudio através de integrações com DAWs já existentes. No entanto, essa abordagem não funcionou devido às dificuldades de integração, manipulação do estado da aplicação e falta de padronização de dados de softwares de terceiros para o sistema de aprendizado de máquina.

4.5.5 Aprimorador

O aprimorador (usado para sugerir edições) foi implementado com TensorFlow.js . Essa biblioteca foi escolhida porque permite exportar o modelo de Inteligência Artificial diretamente para o *front-end*, cortando os custos com GPUs que seriam necessários no *back-end*.

A arquitetura usada foi uma rede neural com *Gated Recurring Unit* (GRU). O plano inicial previa usar a arquitetura de transformadores, descrita por VASWANI, Ashish et al. (2017). Mas depois percebeu-se que a versão em JavaScript ainda não possui suporte para todos os tipos de camadas usadas nessa arquitetura. E mesmo que houvesse suporte, transformadores costumam demandar muito mais memória e são mais propensos a *overfitting*.

⁷ <https://js.tensorflow.org/api/latest>

5 RESULTADOS

O objetivo geral foi parcialmente cumprido. Sonhin funciona para criar músicas completas manualmente (como uma DAW), e é capaz de gerar instrumentos novos a partir de poucos exemplos, mas não é capaz de gerar sequências de notas de forma coerente e interessante, e não consegue sugerir alterações relevantes.

A modelagem de instrumentos como sintetizadores baseados em efeitos, e efeitos como grafos que se conectam uns com os outros, se mostrou promissora. Isso permitirá que, em versões futuras, os próprios usuários possam criar novos efeitos e conseqüentemente sons e músicas com características novas. Além disso, essa forma de representar instrumentos e efeitos se mostrou bastante adequada para algoritmos genéticos, requerendo poucos exemplos iniciais. Por exemplo, foram necessários apenas 10 exemplos para criar instrumentos razoáveis para *kick* de bateria.

Por outro lado, modelar padrões dessa mesma forma se mostrou errôneo. Esperava-se que durante o desenvolvimento do projeto fossem descobertos alguns efeitos gerais para notas (tais como arpejo, variação aleatória de alturas e *staccato*) que o fossem o bastante para gerar sequências razoáveis. Mas, aparentemente, as notas em música apresentam muita dependência temporal e de relacionamento entre si.

O sistema de geração de sugestões também não funcionou, mas por escassez de dados para treinamento e conseqüente *overfitting*. No começo do projeto esperava-se que a quantidade de dados de um único usuário fosse o bastante, pois era planejado o uso de dados sintéticos (variações válidas de dados reais). Mas no decorrer do projeto a ideia se mostrou difícil de implementar.

6 CONCLUSÃO

Este trabalho descreveu a ferramenta Sonhin, a oportunidade de melhoria na produção musical para que motivou sua criação, algumas ferramentas já existentes

para comparação, seus detalhes de arquitetura, implementação e resultados obtidos.

O objetivo geral deste trabalho foi parcialmente atingido, uma vez que esse trabalho mostrou a viabilidade de construir uma DAW com geração de instrumentos sem *overfitting* a partir de poucos exemplos utilizando algoritmos genéticos em dados com um formato propício para isso. Contudo não foi possível gerar sequências de notas de forma satisfatória seguindo a mesma modelagem baseada em efeitos.

Dentre suas realizações técnicas se destacam: criar um motor de áudio capaz de sintetizar uma ampla gama de sons distintos; imbuir o sistema de sugestões de alterações por IA diretamente no *front-end*, conseguir estruturar os diferentes módulos de forma coerente, e criar uma forma útil de estruturar instrumentos e efeitos a partir de grafos.

7 TRABALHOS FUTUROS

Durante o desenvolvimento desse projeto, foi possível notar uma série de melhorias que podem ser realizadas futuramente.

No momento, o aprimorador usa apenas um comando por vez. Isso limita a escolha das alterações que são boas apenas após outras alterações. Por exemplo, adicionar um instrumento pode ser uma escolha ruim por si só caso o instrumento precise de uma segunda alteração para ficar com o volume adequado. Versões futuras do projeto talvez contenham a avaliação de sequências longas de alterações, assim como o uso de destilação e amplificação iterativa, como presente no Alpha Zero do Google, para o treinamento do avaliador.

Atualmente um projeto só pode ser editado por um único usuário. Seria interessante permitir múltiplos usuários em colaboração. Tal funcionalidade não foi implementada na iteração atual porque seria necessário lidar com problemas de sincronização e resolução de conflitos nas mudanças.

Por enquanto só é possível usar uma única progressão harmônica e escala para a música toda. Isso torna a música mais monótona e desinteressante. A versão atual não contou com esse recurso devido a dificuldade de encontrar interfaces

adequadas para um sistema de valores padrões hierárquicos. Uma possível abordagem seria a usada pela linguagem CSS em que qualquer elemento pode ter qualquer atributo, independentemente de se isso altera o resultado final. Dessa forma, seria possível definir progressões harmônicas para seções individuais, tipos de seções e para a música inteira, seguindo essa ordem de especificidade.

REFERÊNCIAS

AYODELE, T. O. Types of Machine Learning Algorithms. Em: ZHANG, Y. (Ed.). **New advances in machine learning**. [s.l.] InTech, 2010. p. 19–49.

BARREAU, P. **How AI could compose a personalized soundtrack to your life**. Disponível em: <<https://www.youtube.com/watch?v=wYb3Wimn01s>>. Acesso em: 14 out. 2024.

BASHEER, I. A.; HAJMEER, M. Artificial neural networks: fundamentals, computing, design, and application. **Journal of microbiological methods**, v. 43, n. 1, p. 3–31, 2000.

BENWARD, B.; SAKER, M. **Music in theory and practice volume 1**. 10. ed. Columbus, OH, USA: McGraw-Hill Education, 2020.

BRIOT, J.-P.; PACHET, F. Deep learning for music generation: challenges and directions. **Neural computing & applications**, v. 32, n. 4, p. 981–993, 2020.

BURMEISTER, J.; WILES, J. **The challenge of Go as a domain for AI research: a comparison between Go and chess**. Proceedings of Third Australian and New Zealand Conference on Intelligent Information Systems. ANZIIS-95. Anais...IEEE, 2002.

CHNG, Z. M. **Loss Functions in TensorFlow**. Disponível em: <<https://machinelearningmastery.com/loss-functions-in-tensorflow/>>. Acesso em: 4 out. 2024.

CHRISTIANO, P. et al. **Deep reinforcement learning from human preferences**. 2017. Disponível em: <<http://arxiv.org/abs/1706.03741>>.

CUNNINGHAM, P.; CORD, M.; DELANY, S. J. Supervised Learning. Em: **Machine Learning Techniques for Multimedia**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008. p. 21–49.

ELDRITCH. **Boring formless nonsense: Experimental music and the aesthetics of failure**. Nova Iorque, NY, USA: Continuum Publishing Corporation, 2013.

FAJAR, K. E.; SUKMAYADI, Y. **Advantages of “DAW” composing music for the effectiveness of learning the process of musical practice**. Proceedings of the 3rd International Conference on Arts and Design Education (ICADE 2020). Anais...Paris, France: Atlantis Press, 2021.

FEUERRIEGEL, S. et al. Generative AI. **Business & information systems engineering**, v. 66, n. 1, p. 111–126, 2024.

FINDEISEN, F. **The Addiction Formula: A holistic approach to writing captivating, memorable hit songs. With 317 proven commercial techniques and 331 examples, incl rather be, happy and all of me**. [s.l: s.n.].

FL Studio Online Manual. Disponível em: <<https://www.image-line.com/fl-studio-learning/fl-studio-online-manual/>>. Acesso em: 2 out. 2024.

FRANÇOIS-LAVET, V. et al. An introduction to deep reinforcement learning. **Foundations and Trends® in Machine Learning**, v. 11, n. 3–4, p. 219–354, 2018.

FREYBERG, K. **Introducing v3**. Disponível em: <<https://suno.com/blog/v3>>. Acesso em: 2 out. 2024.

GODT, I. Music: A practical definition. **The musical times**, v. 146, n. 1890, p. 83, 2005.

GRFENSTETTE, G. Tokenization. Em: **Text, Speech and Language Technology**. Dordrecht: Springer Netherlands, 1999. p. 117–133.

GRINDE, B. A biological perspective on musical appreciation. **Nordisk tidsskrift for musikkterapi**, v. 9, n. 2, p. 18–27, 2000.

HABIBI, A.; DAMASIO, A. Music, feelings, and the human brain. **Psychomusicology**, v. 24, n. 1, p. 92–102, 2014.

HAIMING, L. The Advantages of Using Computer Programs and Artificial Intelligence in The Creation of Film Music. **International Journal of Mechanical Engineering**, p. 1731–1735, 2021.

HASHEMI, M. Enlarging smaller images before inputting into convolutional neural network: zero-padding vs. interpolation. **Journal of big data**, v. 6, n. 1, 2019.

How does SOUNDRAW's AI work? Disponível em: <<https://soundrawhelp.zendesk.com/hc/en-us/articles/18862874435225-How-does-SOUNDRAW-s-AI-work>>. Acesso em: 13 out. 2024.

HUANG, A. **Modular synthesis EXPLAINED**. Disponível em: <<https://m.youtube.com/watch?v=cWslSTTKiFU>>. Acesso em: 15 out. 2024.

HUTCHINGS, P. E.; MCCORMACK, J. Adaptive music composition for games. **IEEE transactions on games**, v. 12, n. 3, p. 270–280, 2020.

JILLINGS, N.; STABLES, R. **An Intelligent audio workstation in the browser**. Conference: 3rd Web Audio Conference. Anais...23 ago. 2017. Disponível em: <https://www.researchgate.net/publication/318597538_An_Intelligent_audio_workstation_in_the_browser>

KANIA, A. **The philosophy of music**, Metaphysics Research Lab, Stanford University, , 2024. (Nota técnica).

KARITA, S. et al. **A Comparative Study on Transformer vs RNN in Speech Applications**. 2019 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU). Anais...IEEE, 2019.

KLAPURI, A.; DAVY, M. (EDS.). **Signal processing methods for music transcription**. 2006. ed. Nova Iorque, NY, USA: Springer, 2006.

KUMAR, M. et al. Genetic algorithm: Review and application. **SSRN Electronic Journal**, 2010.

LAUBHEIMER, P. **Input Controls for Parameters: Balancing Exploration and Precision with Sliders, Knobs, and Matrices**. Disponível em: <<https://www.nngroup.com/articles/sliders-knobs/>>. Acesso em: 10 out. 2024.

MAASØ, A.; HAGEN, A. N. Metrics and decision-making in music streaming. **Popular communication**, v. 18, n. 1, p. 18–31, 2020.

MAHESH, B. Machine learning algorithms - A review. **International journal of science and research (Raipur, India)**, v. 9, n. 1, p. 381–386, 2020.

MCADAMS, S.; MATZKIN, D. Similarity, invariance, and musical variation. **Annals of the New York Academy of Sciences**, v. 930, n. 1, p. 62–76, 2001.

MIRJALILI, S. Genetic Algorithm. Em: **Studies in Computational Intelligence**. Cham: Springer International Publishing, 2019. p. 43–55.

MO, Y. et al. **Large Language Model (LLM) AI text generation detection based on transformer deep learning algorithm**. 2024. Disponível em: <<http://arxiv.org/abs/2405.06652>>.

NAEEM, S. et al. An Unsupervised Machine Learning Algorithms: Comprehensive Review. **International journal of computing and digital systems**, v. 13, n. 1, 16 abr. 2023.

NATTIEZ, J.-J. **Music and discourse: Toward a semiology of music**. Tradução: Carolyn Abbate. Princeton, NJ, USA: Princeton University Press, 1990.

PAYNE, C. **MuseNet**. Disponível em: <<https://openai.com/index/musenet/>>. Acesso em: 1 out. 2024.

PLACK, C. J.; OXENHAM, A. J.; FAY, R. R. (EDS.). **Pitch: Neural Coding and Perception**. 2005. ed. Nova Iorque, NY, USA: Springer, 2005.

POSTOLACHE, E. **From Source Separation to Compositional Music Generation**. [s.l.] Sapienza University of Rome, 29 mar. 2024a.

PUGACHEV, A. A.; KHARCHENKO, A. V.; SLEPTSOV, N. A. Transforming the future: a review of artificial intelligence models. **RUDN Journal of Studies in Literature and Journalism**, v. 28, n. 2, p. 355–367, 2023.

REGELSKI, T. A. Amateuring in Music and Its Rivals. **Action, Criticism, and Theory for Music Education**, v. 6, p. 36, nov. 2007.

REUTER, A. Who let the DAWs out? The digital in a new generation of the digital audio workstation. **Popular Music & Society**, v. 45, n. 2, p. 113–128, 2022.

RIES, E. **The Lean Startup**. Illustrated edition ed. Nova Iorque, NY, USA: Crown Publishing Group, 13 setembro 2011.

SADOK, H.; SAKKA, F.; EL MAKNOUZI, M. E. H. Artificial intelligence and bank credit analysis: A review. **Cogent economics & finance**, v. 10, n. 1, 2022.

SAGIROGLU, S.; SINANC, D. **Big data: A review**. 2013 International Conference on Collaboration Technologies and Systems (CTS). Anais...IEEE, 2013.

SCHMIDT, R. M. **Recurrent Neural Networks (RNNs): A gentle introduction and overview**. 2019. Disponível em: <<http://arxiv.org/abs/1912.05911>>.

SEXTON, T. MuseNet. **Music reference services quarterly**, v. 26, n. 3–4, p. 151–153, 2023.

STURM, B. L. T. et al. AI Music Studies: Preparing for the Coming Flood. **AIMC 2024 (09/09 - 11/09)**, 29 ago. 2024.

THIEL, P.; MASTERS, B. **Zero to One: Notes on startups, or how to build the future**. New York: Crown Business, 2014.

TOKUI, N. **Towards democratizing music production with AI-Design of Variational Autoencoder-based Rhythm Generator as a DAW plugin**. 2020. Disponível em: <<http://arxiv.org/abs/2004.01525>>.

VASWANI, A. et al. **Attention is all you need**. 2017. Disponível em: <<http://arxiv.org/abs/1706.03762>>.

YANG, Y. Analysis of different types of digital audio workstations. **Highlights in Science, Engineering and Technology**, v. 85, p. 563–569, 2024.

YU, J. et al. Suno: potential, prospects, and trends. **Frontiers of Information Technology & Electronic Engineering**, v. 25, n. 7, p. 1025–1030, 2024.

ZHU, Y. et al. **A survey of AI music generation tools and models**. 2023. Disponível em: <<http://arxiv.org/abs/2308.12982>>.

ZÖLZER, U. **DAFX: Digital Audio Effects**. [s.l.] Wiley, 2011.

APÊNDICE A - Motivos do nome Sonhin

O nome Sonhin foi escolhido para otimizar SEO:

- Trata-se de um nome até então sem uso.
- O domínio sonhin.com pôde ser registrado por apenas R\$50,00.
- O nome de usuário sonhinmusics estava disponível nas principais redes sociais.
- Por ser um nome curto e específico, pode ser utilizado em uma gama de produtos, como acontece com Apple Watch e Google Docs.

Como motivo adicional, saguis (“sonhins” em dialeto popular) passam por mudanças na vocalização atualmente explicadas como fenômeno social. O mesmo acontece com humanos e alguns tipos de pássaros. Eles aprendem a produzir sons que recebam mais atenção, assim como um sistema de criação de música com machine learning.

Apesar de tudo, o nome continua aberto para mudanças. Sugestões podem ser enviadas para sonhinmusics@gmail.com.

APÊNDICE B - Iterações anteriores do projeto

Segue-se a listagem de iterações que culminaram na versão atual do projeto.

Mysic

Iniciativa formada por cerca de dez projetos em linguagens variadas (Python, Javascript, C#, Java, Rusty, C++) criados unicamente para tentar renderizar áudio usando plugins no formato VST.

A hipótese era de que para coletar dados em volume e sem direitos autorais seria necessário criar uma DAW de fato. Como DAWs são extremamente limitadas sem instrumentos virtuais, cada projeto consistiu em instalar a biblioteca mais popular que desse suporte a VST e tentar renderizar algo simples.

A iniciativa foi encerrada depois da tarefa se mostrar complexa demais e ao perceber que a Steinberg, empresa responsável pelo padrão VST, deixou de fornecer permissão legal para novas implementações de host VST2 (que ainda é comum).

MAI

MAI, do inglês Musical Artificial Intelligence, é um sistema que surgiu da tentativa de replicar o sucesso da NVIDIA Canvas com rede generativa adversarial (GAN) e redes neurais feedforward (FNN) e convolucional (CNN).

Já contava com banco de dados e loop de treinamento, baseado no do artigo “Deep reinforcement learning from human preferences” da OpenAI (CHRISTIANO et al., 2017).

Terminou por incoerência na representação de músicas e a dificuldade em fazer alterações devido ao tamanho fixo dos resultados, o que tornava impossível ter uma música completa ou tornava o loop de treinamento lento demais.

MAI 1

Tentativa de consertar o problema de representação de MAI usando redes long short term memory (LSTM), ranged tensors, e loops e condicionais dentro do

modelo. Possibilitou resultados de tamanho variado. Terminou por que os loops e condicionais tornavam o cálculo sem derivada, com um total de 0 valores treináveis.

MAI 2

Tentativa de remover complexidade e limitações de tamanho usando apenas geração procedural em vez de machine learning. Funcionou bem para forçar semântica no design. Terminou por ser muito difícil de manter, devido à abordagem imperativa (em vez de declarativa) e não usar o princípio de design aberto-fechado.

MAI 3

Sistema com parâmetros treináveis no modelo, construído com os insights de semântica obtidos em MAI2, com loop de treinamento funcionando. Terminou após depois de se mostrar muito propenso a overfitting, entregando sempre os mesmos valores para todas as notas, em todas as execuções.

MAI 4

Tentativa de resolver o overfitting incluindo desvio padrão dos valores na função de loss. Funcionou em gerar resultados diferentes, mas era muito dependente do ruído aleatório usado como seed (devido ao caráter local de convolução).

Foram então projetados diferentes tipos de seed para diferentes elementos da música (melodia, harmonia, percussão, etc). Mas isso complicou o código ao ponto de ser basicamente geração procedural com uma etapa final que só aumentava a aleatoriedade. O projeto terminou ao demandar tanto fine tuning quanto geração procedural sem vantagens adicionais.

MAI 5

Tentativa de utilizar geração procedural (embora tenha começado como tentativa de usar mecanismos de atenção, como no ChatGPT).

Criado depois de descartar a hipótese de que era preciso um modelo mais geral para lidar com todas as diferentes partes da música simultaneamente. Aparentemente, decisões como notas para o baixo, divisão de tempo para

percussão e progressões para acordes podem ser modeladas por heurísticas simples.

Terminou por ter o código difícil de manter, sem muita divisão de responsabilidades.

MAI 6

Sistema totalmente sem valores treináveis, que gerava resultados melhores que todas as tentativas anteriores. Devido a baixa complexidade em funções especializadas para cada elemento da música, foi possível adicionar estrutura, separando a música em seções, e usar pequenos padrões conhecidos como ear candy.

Seus resultados se mostraram bons o bastante para receber algumas curtidas no TikTok.

Terminou apenas pela dificuldade de tipagem e serialização na linguagem em que foi implementada (Python).

Sonhin 1

Uma vez encontrado um algoritmo razoável para compor músicas, a próxima iteração visou parametrizar o áudio renderizado e conseguir feedback granular o bastante para que o algoritmo pudesse ser treinado sem overfitting. Para resolver esses problemas criou-se, respectivamente, uma engine própria de renderização de música nomeada Wondrous, e uma aplicação web chamada Sonhin.

Sonhin 1 possui uma interface gráfica em que os parâmetros de composição e áudio podem ser alterados pelo usuário. Conta com o sistema de composição e renderização no lado do cliente, para reduzir delay e maximizar feedback.

Para gerenciar o código foi usada a ferramenta de monorepo Nx. Para a interface gráfica foi utilizado a biblioteca React. Para o gerenciamento de estado inicialmente foi usada a Context API e depois uma solução interna baseada em eventos. Isso foi feito para aumentar a performance na atualização de visualizações em múltiplos componentes, além de reduzir a verbosidade de redutores para objetos aninhados.

Terminou depois de testes com usuários que mostrarem que quem pesquisa por uma ferramenta de criação de música com AI tende a não ter experiência em compor músicas. Isso se tornou bem evidente observando gravações de tela, em que praticamente todos os usuários não sabiam usar knobs (um componente gráfico presente em praticamente todas as DAWs).

Sonhin 2

A partir de Sonhin 1, foi possível notar que o sistema não vai ter usuários reais até que a música gerada seja razoável e os controles fáceis de usar. Isso significa que as versões iniciais devem ser otimizadas para receber feedback do autor do projeto.

Pensando nisso, a segunda versão de Sonhin contém muitos instrumentos e configurações escritas manualmente, direto no código. Isso possibilitou confirmar a hipótese de que faltavam efeitos o bastante para gerar música de qualidade, além de mostrar que a estrutura dos dados estava mal definida. Por exemplo, as notas eram salvas no banco de dados, mesmo sendo inviável no momento trabalhar com machine learning a nível de notas, e o fato delas poderem ser recalculadas via efeitos.

Nesse projeto também foram feitas algumas melhorias, como a simplificação do código usando Svelte em vez de React, e removendo o Nx.


Foi encerrado pela dificuldade de manter o código. Principalmente porque os efeitos eram implementados como funções puras (que recebiam o estado como parâmetro e usavam uma abstração confusa para modificá-lo), e porque a aplicação dependia de uma pipeline complexa com muitas etapas.

Sonhin 3

Projeto que visava simplificar a pipeline de geração de notas e áudio através de grafos. Nele cada efeito altera o grafo adicionando nós. Por exemplo, o efeito de pitchSlide (transição de frequência no som de uma nota) gera um pitchSlideNode, cuja saída é usada como entrada para gerar a fase do som. Isso simplificou bastante a pipeline e permitiu criar efeitos mais facilmente.

Com essa facilidade, iniciaram-se experimentos para adicionar alguns dos efeitos mais comuns na produção musical: reverberação e equalização. Estes se provaram difíceis demais de implementar para a GPU, que era um requisito por questões de performance. Sem o suporte a GPU, dados eram trafegados para a CPU e vice-versa, o que tornava o sistema lento demais.

Essa iteração foi encerrada pelos problemas de performance para efeitos que não possam ser paralelizáveis.

	INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DA PARAÍBA
	Campus Cajazeiras - Código INEP: 25008978
	Rua José Antônio da Silva, 300, Jardim Oásis, CEP 58.900-000, Cajazeiras (PB)
	CNPJ: 10.783.898/0005-07 - Telefone: (83) 3532-4100

Documento Digitalizado Ostensivo (Público)

TCC de Ulisses Albuquerque Pereira

Assunto:	TCC de Ulisses Albuquerque Pereira
Assinado por:	Ulisses Pereira
Tipo do Documento:	Anexo
Situação:	Finalizado
Nível de Acesso:	Ostensivo (Público)
Tipo do Conferência:	Cópia Simples

Documento assinado eletronicamente por:

- **Ulisses Albuquerque Pereira, ALUNO (202012010022) DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS - CAJAZEIRAS**, em 23/10/2024 16:25:28.

Este documento foi armazenado no SUAP em 23/10/2024. Para comprovar sua integridade, faça a leitura do QRCode ao lado ou acesse <https://suap.ifpb.edu.br/verificar-documento-externo/> e forneça os dados abaixo:

Código Verificador: 1289245

Código de Autenticação: 225ffd5d1a

