UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA ELÉTRICA E DE COMPUTAÇÃO

TRABALHO DE CONCLUSÃO DE CURSO

Harmonizer: Um Coral de Vozes Sintéticas Controladas por Sinais MIDI

ALUNO: Josué Ferreira de Oliveira | RA: 219132
E-MAIL: j219132@dac.unicamp.br

ORIENTADOR: Bruno Sanches Masiero
E-MAIL: masiero@unicamp.br

Campinas, Novembro de 2024

Conteúdo

Resumo

Isso é para ser um resumo

1 Introdução

1.1 *Harmonizer*

O nome *Harmonizer* foi introduzido em 1975 pela *Eventide, Inc* ao apresentar o *H910*, um processador de áudio capaz de modificar a frequência fundamental dos sinais recebidos [1]. A empresa continua detendo a marca registrada [2].

Figura 1: Eventide H910 Harmonizer.
Adaptada de [3].

Diferentemente de outras formas de alterar a frequência fundamental de um sinal disponíveis à época, o *H910* era capaz de executar o processamento em tempo real e sem alterar a duração do sinal [4].

O processador rapidamente se tornou popular, sendo utilizado por nomes como Jon Anderson, vocalista da banda *Yes*, Frank Zappa e Tony Visconti, produtor que trabalhou com David Bowie [5].

Mais recentemente, o termo *harmonizer* passa a ser utilizado de forma metonímica para referir-se a *softwares* ou equipamentos que permitem combinar um sinal com uma versão deste cuja frequência fundamental foi alterada, permitindo a criação de harmonias a partir de um único sinal [6][7].

Em particular, citamos como inspiração para esse projeto a implementação de *harmonizer* criada por Ben Bloomberg para o cantor, multi-instrumentalista e compositor Jacob Collier [8][9]. Diferentemente do *H910*, essa implementação permite que se especifique a frequência fundamental alvo que se deseja que o sinal de saída tenha, ao invés de somente a razão entre as frequências fundamentais do sinal de entrada e de saída. Adicionalmente, essa implementação é polifônica e recebe as frequências fundamentais alvo de cada nota de saída através de comandos *MIDI*.

Para reproduzir esse efeito, é necessário não somente um *phase vocoder* que permita alterar dinâmicamente a frequência fundamental do sinal de entrada como um método de detecção de frequência fundamental que permite informar ao *phase vocoder* qual deve ser a razão f_a/f_i entre

as frequências fundamentais do sinal de entrada e de saída.

1.2 *Phase Vocoders*

Em um *harmonizer*, procuramos por técnicas que modifiquem a frequência fundamental do sinal de interesse mas mantenham suas características de timbre e sua duração. As abordagens mais comuns para esse problema podem ser divididas em duas categorias: as que se baseiam no algoritmo *overlap and add* e as que se baseiam em *phase vocoders* [10].

Os algoritmos de *pitch shifting* baseados em variações de *overlap and add* operam diretamente no domínio do tempo. Nesses, o sinal de entrada $x(n)$ é dividido em m trechos curtos $x_m(r)$, geralmente de duração de dezenas de milisegundos. Driedger e Müller[10] definem cada trecho como:

$$x_m(r) = \begin{cases} x(r + mH_a), \;\; r \in [-N/2 : N/2] \\ 0, \;\; c.c \end{cases} \tag{1}$$

Onde H_a é a distância, em amostras, entre o começo de x_m e x_{m+1} e N é o número de amostras em cada trecho.

Os trechos de áudio podem então ser deslocados no tempo e o valor de suas amostras somados. Seja H_s a nova distância entre o início de blocos de áudio adjacentes. Se $H_s < H_a$, temos que o sinal resultante terá menor duração que o sinal de entrada. Por outro lado, $H_s > H_a$ resultará em um sinal mais "alongado "no tempo.

Esperamos, no entanto, que a frequência fundamental do sinal permança inalterada, uma vez que o período do sinal foi preservado localmente em cada bloco de áudio.

O sinal resultante pode então ser reamostrado de forma a resultar em áudio com a mesma duração do de origem mas com frequência fundamental distinta.

A forma mais simples do algoritmo *overlap and add* (OLA) não garante, no entanto, a preservação de padrões periódicos locais do áudio. Além disso, há a aparição de artefatos nos pontos em que há sobreposição de blocos de áudio.

Diversas propostas, como *WSOLA*, *SOLA* e *PSOLA* buscam modificar o espaçamento bloco a bloco de forma a reduzir os artefatos enquanto mantém a distância média H_s. No entanto, de forma geral, algoritmos baseados em *phase vocoders* produzem menos artefatos para sinais majoritariamente periódicos [10].

Inicialmente, a ideia por trás dos *phase vocoders* foi proposta no âmbito de telecomunicações[11].

Na proposta de Flanagan e Golden[12], um sinal de voz seria representado pelas amplitudes e fase das componentes em frequência nele presentes em curtas janelas de tempo. Uma vez que essas quantidades variam mais lentamente que a amplitude do sinal no tempo, tal abordagem seria capaz de reduzir a quantidade de dados necessária para transmitir o sinal.

Posteriormente, tal representação mostrou-se útil ao processamento de sinais musicais, uma vez que permite modificar características de timbre, *pitch* e duração do sinal de forma independente.

Na abordagem de *phase vocoder*, cada bloco $x_m(r)$ é janelado no tempo e então é aplicada a ele a transformada rápida de Fourier (FFT) ou forma similar da transformada discreta de Fourier. O método de *phase vocoder* utiliza a informação de fase presente no espectrograma gerado pela FFT para obter estimativas das frequências instantâneas em cada raia da FFT. Esse procedimento é necessário uma vez que a resolução em frequência oferecida pela FFT de um bloco curto de amostras é limitada.

Com as estimativas de frequência instantâneas calculadas, as fases de cada componente em frequência são ajustadas ao deslocar os blocos no tempo para que não haja descontinuidades de fase nas sobreposições de blocos.

Como dual das técnicas baseadas em OLA, as técnicas baseadas em *phase vocoders* garantem a continuidade de sinais periódicos modificados pelo algoritmo. No entanto, transientes, isto é, momentos em que há mudanças abruptas na fase das componentes em frequência do sinal, não são bem preservados.

Surgem então diversos métodos para detectar transientes e permitir que haja descontinuidade de fase no espectrograma modificado nesses instantes em específico.

1.3 Deteção de *pitch*

Pitch ou altura é entendida como um atributo da sensação de audição que permite diferenciar sons numa escala entre sons baixos e sons altos [13]. Essa é uma sensação subjetiva difícil de ser quantificada mas que frequentemente está correlacionada à frequência fundamental do sinal, que por sua vez é uma quantidade física e mensurável [14].

A informação de *pitch* é essencial à prosódia da fala [15] e, em línguas tonais, auxilia na diferenciação de categorias lexicais. Portanto, um sistema que consiga obter estimativas do *pitch* de um sinal de áudio a partir de suas características acústicas é útil ao desenvolvimento de sistemas de reconhecimento de fala.

Em contextos musicais, estimativas de *pitch* podem ser utilizadas como parte de um sistema

de transcrição automática, ou em outras formas de processamento como a obtenção de envoltórias espectrais e reconhecimento de timbre de instrumentos [16].

Há diversas técnicas voltadas para a estimação de frequência fundamental de um sinal de áudio. De forma geral, essas podem ser divididas em três grupos: um que se baseia em informação temporal, um que se utiliza das características em frequência do sinal e outro composto por abordagens híbridas [17].

Algoritmos de detecção de *pitch* que operam diretamente no domínio do tempo baseiam-se primariamente na sequência de autocorrelação do sinal sob análise. Dado um bloco $X_m(r)$ com N amostras do sinal de entrada, definimos a sequência de autocorrelação $R(k)$ do bloco como:

$$R(k) = \frac{1}{N} \sum_{r=0}^{N-1-k} X_m(r) X_m(r+k), \;\; k = 0, 1, ..., N-1 \tag{2}$$

Onde k é o atraso de autocorrelação. Assumindo que $X_m(r)$ seja um trecho de um sinal periódico, a quantidade $R(k)$ deve se maximizar quando k for tal que $\tau \approx k \cdot \frac{1}{f_s}$, onde τ é o período do sinal e f_s é a frequência de amostragem.

No entanto, a abordagem em que se assume que o maior valor de $R(k)$ está sempre diretamente relacionado ao período do sinal apresenta pouca robustez, sobretudo para sinais de baixa frequência e tamanhos de janela N relativamente pequenos [17].

Algoritmos como *AUTOC*, *YIN* e *PYIN* aplicam heurísticas a essa abordagem básica a fim de torná-la mais precisa e robusta à presença de ruído [16][18] [17].

Mais recentemente, surgem algoritmos de detecção de *pitch* baseados em redes neurais artificiais, como *CREPE*[19] e *SPICE*[20]. Tais algoritmos substituem as heurísticas por uma abordagem em que a função que mapeia o vetor de amostras $S(n)$ em uma frequência fundamental é obtida diretamente com base em dados.

1.4 Protocolo MIDI

O protocolo MIDI (*Musical Instrument Digital Interface*) foi inicialmente proposto como uma forma de normatizar a comunicação entre sintetizadores de diferentes fabricantes. Com a introdução do formato *General MIDI*, tornou-se também uma forma comum de representar em arquivos digitais informações de performance musical [21].

Inicialmente, mensagens MIDI eram transmitidas por meio de uma interface serial assíncrona a uma taxa de $31,25$ Kilobits por segundo [21]. Mais recentemente, produtos com interface *USB* podem transmitir sequências de comandos MIDI diretamente para computadores pessoais através

da especificação *USB MIDI Class* [22].

Podemos citar como principais comandos do protocolo MIDI os comandos de eventos *Note On* e *Note Off*. O primeiro indica o início da execução de uma nota e inclui também informação sobre a dinâmica com que a nota foi executada, isto é, uma medida relacionada à intensidade sonora. Os comandos *Note Off* por sua vez indicam quando uma nota se encerra, isto é, no caso de um instrumento de teclas, quando a tecla deixa de ser pressionada [23].

Por sua ampla aceitação e relativa simplicidade, o protocolo MIDI permite desacoplar a interface entre quem executa a música e o dispositivo que de fato produz os sons escutados. Com isso, surge uma vasta gama de *controladores MIDI*, dispositivos que permitem ao usuário gerar sequências de comandos MIDI utilizando diversos tipos de interface física.

1.5 Geradores de envoltória (ou *envelope*)

Num sintetizador, o gerador de envoltórias (ou gerador de *envelopes*, empréstimo do inglês) é responsável por modular a amplitude do sinal gerado em função do tempo. Tal modulação permite que o som sintetizado imite a envoltória de um instrumento real ou crie sons que evoluem temporalmente, de forma geral [24].

Trata-se, portanto, de um componente básico frequentemente presente em sintetizadores subtrativos, FM ou aditivos. Geradores de envoltória nesses instrumentos frequentemente são baseados no modelo *ADSR*, em que a envoltória é dada por uma função paramétrica definida por partes que pode ser dividida em quatro fases distintas:

- *Attack* (ataque): Fase em que a amplitude da envoltória vai de seu valor mínimo até o valor máximo
- *Decay* (decaimento): Amplitude transiciona do valor máximo para valor de sustentação
- *Sustain* (sustentação): Amplitude permanece constante até que a haja um sinal para progredir para a próxima fase.
- *Release* (soltura): Amplitude decai do patamar de sustentação de volta para o valor mínimo.

Esse modelo de gerador de envoltórias permite simular a envoltória de uma vasta gama de sons, desde sons produzidos por instrumentos em que notas são sustentadas, como instrumentos de sopro ou cordas friccionadas, até instrumentos em que o perfil de amplitude é similar ao de uma exponencial decrescente, como instrumentos percussivos ou de cordas dedilhadas.

2 Objetivos

Propôs-se a elaboração de um software em *Python* que implementa um *harmonizer*. Isto é, a aplicação deve, em tempo real:

- Receber um bloco de amostras de um áudio monofônico e comandos MIDI especificando as frequências fundamentais dos sinais de saída.
- Utilizar o algoritmo *CREPE* para identificar a frequência fundamental do bloco de amostras do sinal de entrada recebido.
- Determinar as razões f_a/f_i para cada uma das notas recebidas por MIDI.
- Produzir, utilizando a biblioteca *Rubber Band Library*, uma saída monofônica para cada nota especificada, preservando o timbre do sinal de entrada mas com as novas frequências fundamentais.
- Reproduzir uma combinação linear (*mixagem*) do sinal de entrada e cada um dos novos sinais produzidos.

Escolheu-se o algoritmo *CREPE*[19] para estimação de frequência fundamental em tempo real. A inferência do algoritmo é executada num acelerador de redes neurais *Coral Edge TPU* [25].

As demais etapas de processamento foram realizadas num computador pessoal com CPU *Intel Core i5-8365U*, 16 *GiB* de RAM e disco SSD padrão *NVME*.

3 Componentes da aplicação

3.1 PyAudio

A biblioteca [26], cujo código é disponibilizado sob licença MIT, oferece um módulo multiplataforma para capturar e reproduzir sinais de áudio.

A bibilioteca facilita o processamento de áudio em tempo real e foi utilizada no *harmonizer* proposto para receber os segmentos do áudio de entrada e reproduzir as versões processadas desse.

3.2 pyrtmidi

O módulo *pyrtmidi*[27] fornece uma interface para a biblioteca *rtmidi*. Essa, por sua vez, permite à aplicação receber comandos enviados por mensagens MIDI tanto de dispositivos MIDI

virtuais (i.e, outras aplicações) quanto de dispositivos de hardware, como controladores e teclados MIDI.

A bibiloteca, de código aberto, está disponível sob licença MIT e é compatível com *Linux*, *Windows* e *MacOS*.

3.3 Algoritmo CREPE

3.3.1 O algoritmo

O algoritmo CREPE [19] (*Convolutional Representation for Pitch Estimation*) se enquadra numa nova classe de algoritmos de detecção de *pitch* baseados em redes neurais artificiais.

Especificamente, CREPE consiste numa rede neural convolucional profunda cuja arquitetura é apresentada na figura 2. A entrada da rede consiste em 1024 amostras do sinal sob análise, diretamente no domínio do tempo. As seis camadas convolucionais da rede transformam então esse vetor de amostras numa representação em espaço latente de dimensão 2048.

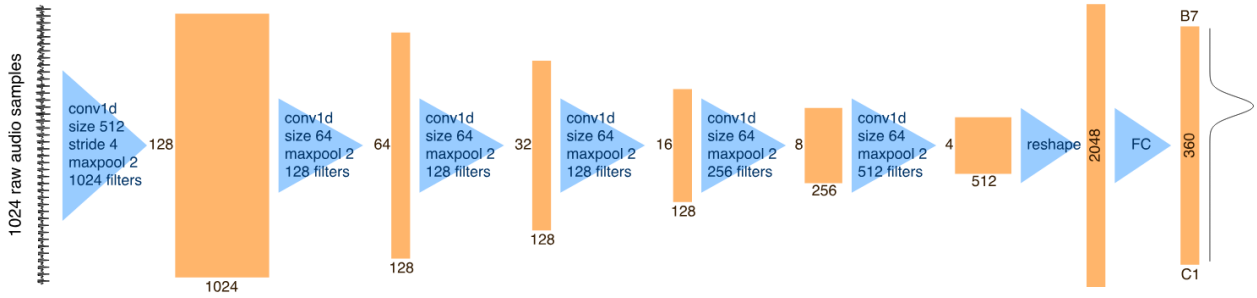

Figura 2: Arquitetura da rede CREPE.
Adaptada de [19]

Finalmente, a representação em espaço latente é transformada em uma saída categórica por uma camada densa com ativação *softmax* cujo vetor de saída tem dimensão 360. Cada entrada no vetor de saída representa a probabilidade estimada de que a frequência fundamental do sinal de entrada esteja num determinado intervalo de frequências. A rede é treinada para que o vetor de saídas aproxime uma função densidade de probabilidade Gaussiana.

Nota-se que a rede é treinada de forma supervisionada, o que requer um banco de dados de áudios monofônicos com anotações de frequência fundamental instantânea extremamente precisas. Os bancos de dados utilizados consistem em dados sintetizados, fornecendo portanto anotações exatas de frequência fundamental. Parte dos bancos de dados foi gerada através do método descrito em [28] em que gravações de sons naturais são resintetizadas de forma que sua frequência fundamental seja conhecida.

Comparado aos algoritmos PYIN[18] e SWIPE[29], ambos baseados em abordagens não-neurais, o algoritmo CREPE apresentou melhor acurácia nas estimativas de frequência fundamental e maior estabilidade sob presença de ruído.

3.3.2 Inferência com acelerador neural *Coral Edge TPU*

TPUs ou *Tensor Processing Units* são uma nova classe de *ASICs* (*Application-Specific Integrated Circuit)* cuja arquitetura é projetada especificamente para tarefas de aprendizado de máquina. Em comparação com *GPUs*, que são processadores de aspecto mais geral, *TPUs* podem apresentar eficiência até 80 vezes maior [30].

No *harmonizer* proposto, o processamento do algoritmo CREPE é executado num acelerador de redes neurais *Coral Edge TPU*, mostrado na figura 3. O acelerador tem aplicação voltada à inferência de modelos profundos em dispositivos embarcados, portanto, com baixo consumo de energia e alta eficiência.

Mesmo com potência total de $2\,W$, a TPU é capaz de executar inferência de modelos como a $ResNet-50$ mais rapidamente que CPUs com potência dezenas de vezes maior [31].

Tal feito é possível parcialmente graças à relativa resiliência de redes neurais profundas à quantização de pesos [32]. Isto é, para inferência no acelerador, os pesos e ativações da rede devem ser convertidos de números de ponto flutuante em números inteiros de 8 bits. Apesar da redução de precisão na representação numérica, a acurácia dos modelos pode ser similar à dos modelos em ponto flutuante, dada uma boa escolha de coeficientes de quantização [33].

No repositório do *GitHub* do modelo CREPE [34], são disponibilizadas cinco versões distintas do modelo pré-treinado, listadas abaixo:

- *tiny*: Modelo com 487.096 parâmetros.
- *small*: Modelo com 1.629.192 parâmetros.
- *medium*: Modelo com 5.879.464 parâmetros.
- *large*: Modelo com 12.751.176 parâmetros.
- *full*: Modelo com 22.244.328 parâmetros.

Essas diferem somente quanto à quantidade de *kernels* em cada camada convolucional, sendo a versão *full* a apresentada na figura 2. Foi escolhida para a aplicação a versão *medium* da rede, uma vez que essa é a versão com mais parâmetros que, depois de quantizada, ocupa menos de $8\,MiB$,

a quantidade total de *SRAM* disponível no acelerador. A versão *medium* do modelo apresenta os seguintes números de *kernels* por camada:

- Camada 1: 512 *kernels*
- Camada 2: 64 *kernels*
- Camada 3: 64 *kernels*
- Camada 4: 64 *kernels*
- Camada 5: 128 *kernels*
- Camada 6: 256 *kernels*

O modelo foi quantizado utilizando o *TensorFlow Lite*.

Figura 3: Acelerador de redes neurais USB baseado no chip *Coral Edge TPU*. Adaptada de [35].

A tempo de execução, a biblioteca *PyCoral* permite a comunicação com a TPU, carregamento do modelo e processamento de inferências.

3.4 *Rubber Band Library*

A biblioteca *Rubber Band Library*[36] implementa um *phase vocoder* de alta qualidade. De código aberto, disponível sob licença GPL, a biblioteca possui *bindings* para diversas linguagens.

No *harmonizer* proposto, foi utilizado o módulo *pylibrb*[37] que provê acesso de baixo nível à *Rubber Band Library*.

O *phase vocoder* implementado pela biblioteca utiliza-se de diversas heurísticas, como detecção de transientes e interpolação de fator de alteração de frequência fundamental para maximizar a coerência de fase entre blocos adjacentes dos sinais gerados e simultaneamente preservar transientes do sinal.

4 Arquitetura da aplicação

O anexo I contém um diagrama da arquitetura completa do *harmonizer* proposto. Aqui, serão discutidos aspectos específicos desta.

4.1 *Threads* gestoras

O sub-sistema de *threads* gestoras coordena as demais *threads* da aplicação e é composta por:

- *Thread* de comandos MIDI: Recebe mensagens de dispositivos MIDI e, a partir destas, mantem uma lista de notas sendo tocadas no momento. Com a lista de notas ativas, atribui frequências fundamentais alvo f_a a cada uma das *threads* de *phase vocoder*. Também é responsável por informar à *thread* geradora de envoltórias sobre o início das fases de ataque e de *release* de cada envoltória.

- *Thread* geradora de envoltória: Responsável por atualizar estado dos geradores de envoltória associados a cada *thread* de *phase vocoder*, além de efetivamente calcular os valores de amplitude correspondentes a cada estado.

4.1.1 Gerador de envoltórias

A cada instante n, a saída $e_k(n)$ do gerador de envoltórias para a k-ésima *thread* de *phase vocoder* é dada por $e^{s_k(n)}$.

O parâmetro $s_k(n)$, por sua vez, é dado, para as fases de ataque e *release* do gerador, pela relação recorrente:

$$s_k(n) = \begin{cases} s_k(n-1) - \frac{\alpha_{min}}{a/\tau_e},\ s_k(n-1) < 0 \\ s_k(n-1) + \frac{log(s)}{d/\tau_e},\ s_k(n-1) \geq 0 \text{ e } e^{s_k(n-1)} > s \end{cases}$$

Onde a e s são os parâmetros de ataque e *sustain* do gerador, α_{min} é constante que dá o valor mínimo de $s_k(n)$ e τ_e é o inverso da frequência de atualização do gerador de envoltórias. Temos que α_{min} deve ser negativa e os demais parâmetros positivos.

Durante a fase de *sustain*, $s_k(n) = s$. Já durante a fase de *release*:

$$s_k(n) = \begin{cases} s_k(n-1) + \frac{\alpha_{min}}{r/\tau_e},\ s_k(n-1) > \alpha_{min} \\ \alpha_{min},\ c.c \end{cases}$$

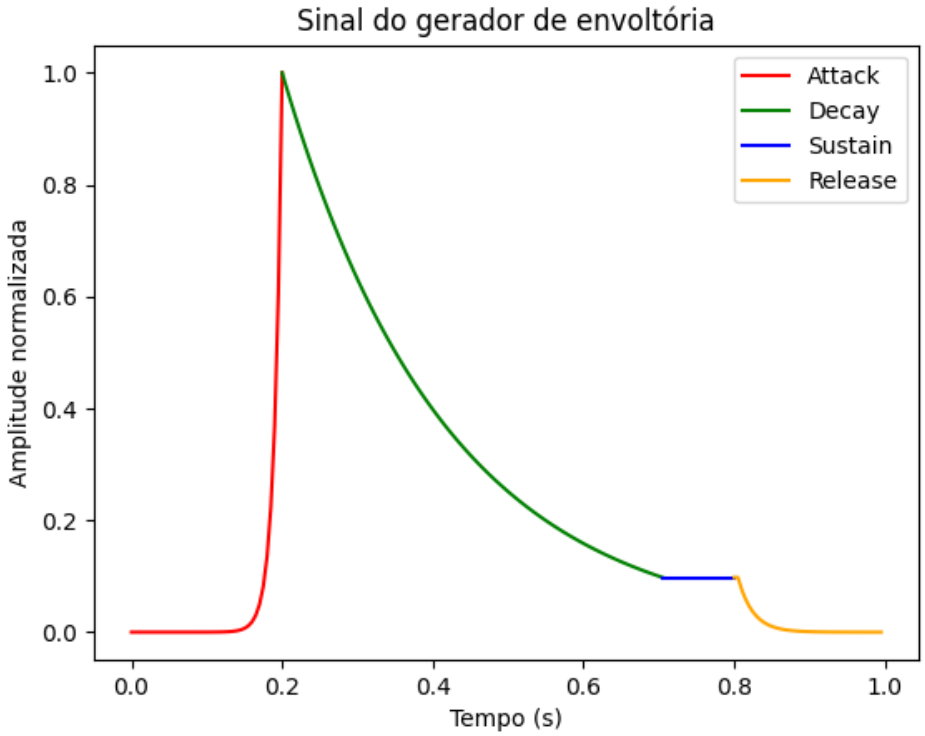

Figura 4: Exemplo de saída do gerador de envoltória.

A Figura 4 mostra a saída do gerador de envoltória para parâmetros $a = 0,2$, $d = 0,5$, $s = 0,1$, $r = 0,4$ e taxa de atualização do gerador de 200 Hz

4.2 *Threads* detectoras de *pitch*

Sub-sistema composto por duas *threads*:

- *Thread* de fila de amostras: Responsável por receber blocos de áudio da *API PyAudio* e enfileira-los para consumo pela *thread* detectora de *pitch* em si.

- *Thread* detectora de *pitch*: Consome blocos de áudio capturados do dispositivo de entrada e os prepara para processamento pelo algoritmo CREPE, o que inclui normalizar a amplitude das amostras e filtrar e reamostrar os blocos para a frequência de amostragem esperada pelo CREPE. Faz então a comunicação com o acelerador de redes neurais através da *API PyCoral* e faz o pós-processamento dos dados, o que inclui o uso do algoritmo de Viterbi para obter estimativas refinadas da frequência fundamental baseado em estados anteriores do estimador e a seleção da frequência de maior probabilidade. As estimativas de frequência fundamental são então enviadas para filas individualizadas para cada *thread* de *phase vocoder*.

4.3 *Threads* de *phase vocoder*

As *threads* de *phase vocoder* são efetivamente responsáveis pela geração de áudio no sistema.

Recebem das *threads* gestoras e da *thread* detectora de *pitch* informações de amplitude de seu gerador de envoltória, frequência fundamental alvo f_a e frequência fundamental do áudio de entrada f_i.

Buscam então blocos do áudio de entrada da *API PyAudio* que são enviados à biblioteca *Rubber Band Library*, juntamente com a razão f_a/f_i para o bloco. A amplitude final do áudio recebido do *phase vocoder* é ajustada conforme o valor recebido pelo gerador de envoltória e o bloco de áudio resultante é enviado para reprodução pela *PyAudio*.

5 Resultados

5.1 Análise descritiva

Foram feitos diversos testes com o sistema, utilizando-se tanto um controlador MIDI em *hardware* quanto sinais MIDI provenientes de outras aplicações de *software*.

Os testes foram feitos com voz masculina grave, isto é, de frequência fundamental entre 70 e 300 Hz.

Notou-se que, em média, o sistema é capaz de produzir vozes sintetizadas similares ao sinal de entrada natural. No entanto, a síntese não é livre de artefatos, sendo que a execução de diversas instâncias do *phase vocoder* frequentemente exaure os *buffers* de saída do *PyAudio*, resultando em *cliques* audíveis.

Adicionalmente, há flutuações ocasionais na frequência fundamental dos sinais sintetizados. Atribuímos esse efeito a blocos específicos do sinal de entrada que aumentam a incerteza das

estimativas do CREPE. As *threads* sintetizando os sinais de saída rejeitam estimativas com menos de 30% de certeza, a fim de amenizar os efeitos do ruído no sinal de entrada.

5.2 Comparação: TPU vs CPU

Comparamos a performance de inferência do modelo CREPE da TPU e CPU em dois aspectos distintos: tempo de processamento e precisão das estimativas de frequência fundamental.

Para avaliar o tempo de processamento do algoritmo, foram feitos 20 ensaios. Em cada ensaio, um vetor de entrada de variáveis aleatórias com distribuição uniforme no intervalo $[-1, 1)$ é gerado. O tamanho do vetor de entrada no n-ésimo ensaio é dado por $\lfloor 1024 + \frac{5120}{20} \cdot n \rfloor$, onde $\lfloor \cdot \rfloor$ denota a função piso e sendo n inicializado como $n = 0$.

Em cada ensaio, foram executadas cinco inferências na TPU e cinco na CPU usando o mesmo vetor de entrada e os tempos médios de execução para cada sistema foram obtidos, em função do tamanho do vetor de entrada. Os resultados dos ensaios são mostrados na Figura 5.

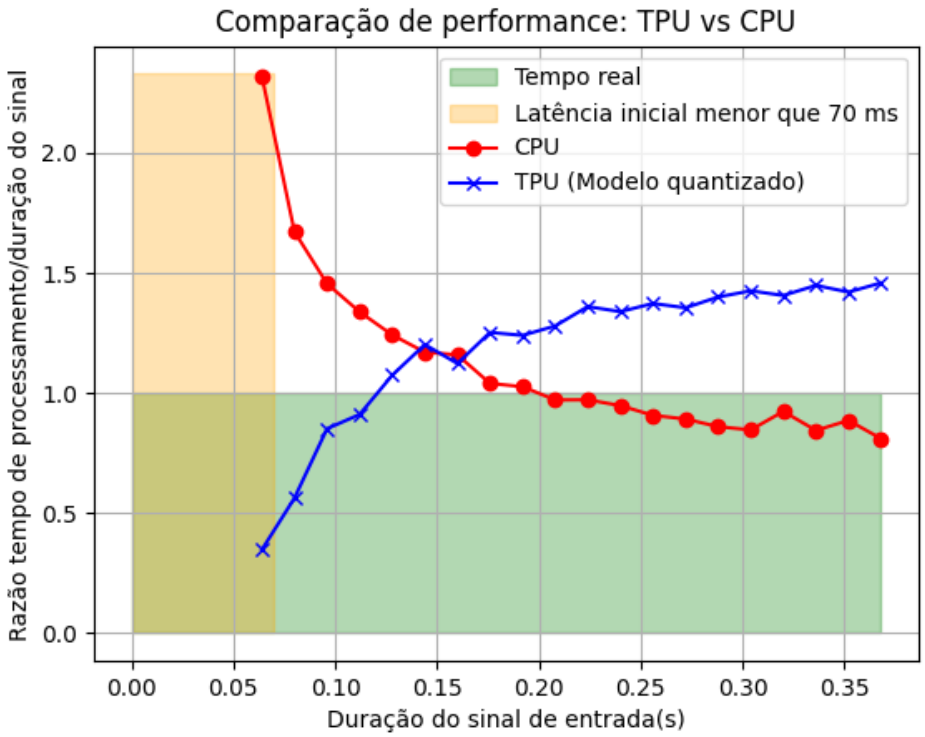

Figura 5: Comparação de tempo de inferência utilizando CPU vs TPU.

Notamos que, para essa versão do algoritmo CREPE (*medium*), a inferência em tempo real com latência considerada aceitável é possível somente com auxílio da TPU.

Para sinais com duração maior que 144 ms, a CPU passa a executar as inferências mais rapidamente. Isso indica que há um *overhead* por inferência maior para a CPU do que para a TPU, apesar de em média a CPU possuir maior capacidade de processamento.

Determinamos o impacto da quantização do modelo na acurácia das estimativas utilizando o *dataset* Bach10, que contém anotações de frequência fundamental.

5.3 Avaliação dos sinais sintetizados

Para avaliar os sinais gerados pelo *phase vocoder*, foi feita uma comparação entre a gravação de uma nota cantada gerada naturalmente e uma gravação sintetizada pelo *vocoder*.

Num primeiro momento, registrou-se uma gravação da nota cantada Mi-4, de frequência fundamental $f_{0_a} = 164,8138\ Hz$. Em seguida, cantou-se a nota Mi-3, de frequência fundamental $f_{0_b} = 82,4069 Hz$ e, através de comandos MIDI, foi pedido ao sistema que sintetizasse a partir do sinal de entrada um sinal com frequência fundamental igual à f_{0_a}, isto é, de $164,8138\ Hz$.

Os espectrogramas dos sinais natural e sintetizado são comparados nas Figuras 6 e 7.

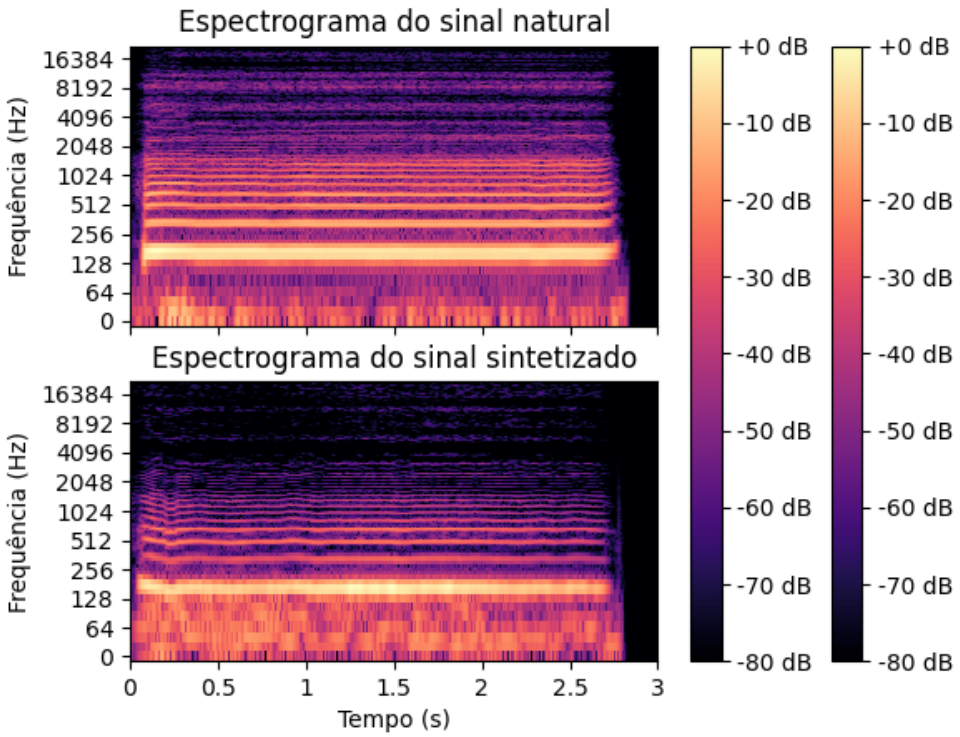

Figura 6: Comparação entre espectrogramas natural e sintetizado.

Da comparação, destacamos:

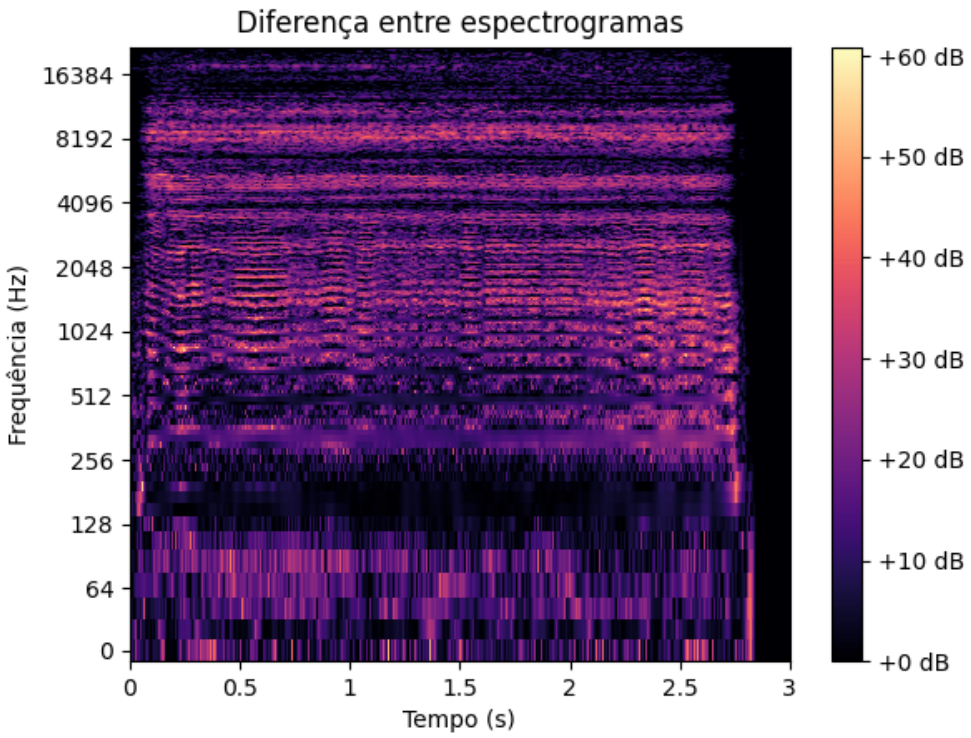

Figura 7: Valor absoluto da diferença entre espectrogramas natural e sintetizado.

- Falta de harmônicos de ordem superior no sinal sintetizado: Percebemos que a potência média do sinal sintetizado para frequências maiores que $3\,kHz$ é muito menor que no sinal natural. Esse aspecto revela uma limitação do *phase vocoder* utilizado.

- Oscilações de frequência fundamental: Notamos que, no sinal sintetizado, há instantes flutuação de frequência fundamental, acompanhada por oscilações em todos os harmônicos correspondentes. Atribuímos esses artefatos ao atraso presente entre o sinal de entrada e as estimativas de frequência fundamental geradas a partir deste.

Juntos, esses dois fatores geram artefatos audíveis que prejudicam a naturalidade do som sintetizado.

6 Conclusão

O presente trabalho demonstrou a viabilidade da construção de um *harmonizer* para operação em tempo real utilizando algoritmos de deteção de *pitch* neurais e *phase vocoders*.

Foi possível também avaliar as vantagens do uso de aceleradores neurais em relação à execução de inferência de modelos neurais em CPUs de uso geral.

Anexo I

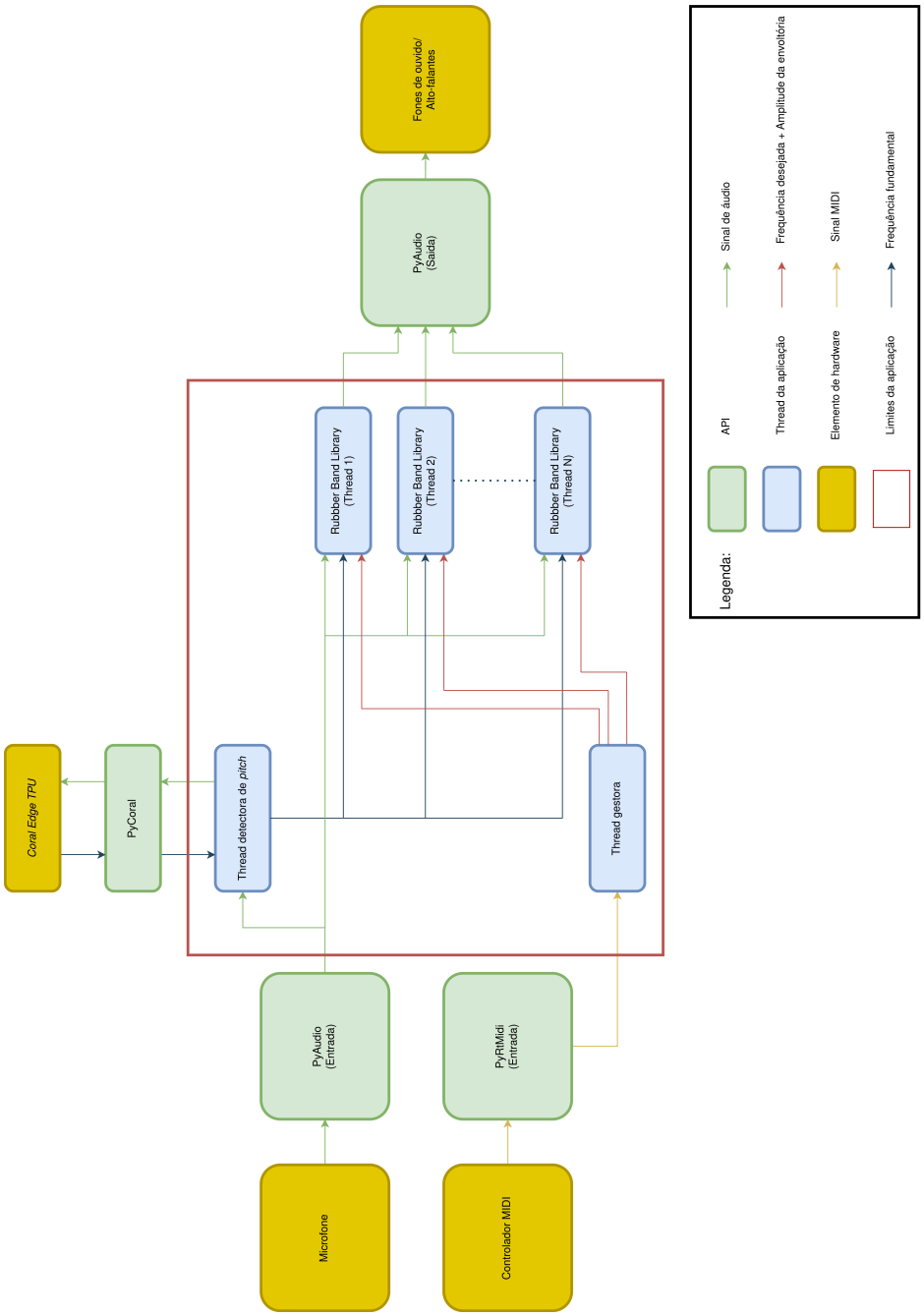

Figura 8: Arquitetura da aplicação proposta.

Referências

[1] I. Eventide. Flashback #4.1: The h910 harmonizer®. [Disponível online; acesso em 09-Novembro-2024]. [Online]. Available: https://www.eventideaudio.com/50th-flashback-4-1-the-h910-harmonizer/

[2] J. Trademarks. Harmonizer - trademark details. [Disponível online; acesso em 09-Novembro-2024]. [Online]. Available: https://trademarks.justia.com/731/06/harmonizer-73106071.html

[3] Wikipedia contributors, "Eventide, inc — Wikipedia, the free encyclopedia," 2023, [Disponível online; acesso em 09-Novembro-2024]. [Online]. Available: https://en.wikipedia.org/wiki/Eventide,_Inc

[4] I. Eventide. Flashback #4.2: H910 harmonizer® pt. 2. [Disponível online; acesso em 09-Novembro-2024]. [Online]. Available: https://www.eventideaudio.com/50th-flashback-4-2-h910-harmonizer-the-product/

[5] ——. H910 harmonizer®. [Disponível online; acesso em 09-Novembro-2024]. [Online]. Available: https://www.eventideaudio.com/plug-ins/h910-harmonizer/

[6] T. Electronic. Quintessence harmony. [Disponível online; acesso em 09-Novembro-2024]. [Online]. Available: https://www.tcelectronic.com/product?modelCode=0709-AGJ

[7] W. Audio. Waves harmony - real-time vocal harmonizer plugin. [Disponível online; acesso em 09-Novembro-2024]. [Online]. Available: https://www.waves.com/plugins/waves-harmony

[8] J. Collier. Danny boy. [Disponível online; acesso em 09-Novembro-2024]. [Online]. Available: https://www.youtube.com/watch?v=ZXIApugIuqk

[9] B. Bloomberg. bio. [Disponível online; acesso em 09-Novembro-2024]. [Online]. Available: https://ben.ai/bio/

[10] J. Driedger and M. Müller, "A review of time-scale modification of music signals," *Applied Sciences*, vol. 6, no. 2, 2016. [Online]. Available: https://www.mdpi.com/2076-3417/6/2/57

[11] M. Dolson, "The phase vocoder: A tutorial," *Computer Music Journal*, vol. 10, no. 4, pp. 14–27, 1986.

[12] J. L. Flanagan and R. M. Golden, "Phase vocoder," *Bell system technical Journal*, vol. 45, no. 9, pp. 1493–1509, 1966.

[13] B. Moore, *Hearing*, ser. Handbook of Perception and Cognition, Second Edition. Academic Press, 1995, pp. 267–290. [Online]. Available: https://books.google.com.br/books?id=OywDx9pxCMYC

[14] W. A. Yost, "Pitch perception," *Attention, Perception, & Psychophysics*, vol. 71, no. 8, pp. 1701–1715, 2009.

[15] K. Kasi and S. A. Zahorian, "Yet another algorithm for pitch tracking," in *2002 IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 1, 2002, pp. I–361–I–364.

[16] A. De Cheveigné and H. Kawahara, "Yin, a fundamental frequency estimator for speech and music," *The Journal of the Acoustical Society of America*, vol. 111, no. 4, pp. 1917–1930, 2002.

[17] L. Sukhostat and Y. Imamverdiyev, "A comparative analysis of pitch detection methods under the influence of different noise conditions," *Journal of Voice*, vol. 29, no. 4, pp. 410–417, 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0892199714002021

[18] M. Mauch and S. Dixon, "Pyin: A fundamental frequency estimator using probabilistic threshold distributions," in *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2014, pp. 659–663.

[19] J. W. Kim, J. Salamon, P. Li, and J. P. Bello, "Crepe: A convolutional representation for pitch estimation," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2018, pp. 161–165.

[20] B. Gfeller, C. Frank, D. Roblek, M. Sharifi, M. Tagliasacchi, and M. Velimirović, "Spice: Self-supervised pitch estimation," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 28, pp. 1118–1128, 2020.

[21] J. Heckroth, "A tutorial on midi and wavetable music synthesis," *Application Note, CRYSTAL a division of CIRRUS LOGIC*, 1998.

[22] U. I. Forum. Universal serial bus device class definition for midi devices. [Disponível online; acesso em 12-Novembro-2024]. [Online]. Available: https://www.usb.org/sites/default/files/USB%20MIDI%20v2_0.pdf

[23] H. M. de Oliveira and R. de Oliveira, "Understanding midi: A painless tutorial on midi format," *arXiv preprint arXiv:1705.05322*, 2017.

[24] T. Pinch and F. Trocco, *Analog days: The invention and impact of the Moog synthesizer*. Harvard University Press, 2004.

[25] C. AI. Usb accelerator datasheet. [Disponível online; acesso em 09-Novembro-2024]. [Online]. Available: https://coral.ai/products/accelerator

[26] H. Pham. Pyaudio. [Disponível online; acesso em 09-Novembro-2024]. [Online]. Available: https://people.csail.mit.edu/hubert/pyaudio/

[27] P. Stinson. pyrtmidi. [Disponível online; acesso em 09-Novembro-2024]. [Online]. Available: https://github.com/patrickkidd/pyrtmidi

[28] J. Salamon, R. Bittner, J. Bonada, J. Bosch, E. Gómez, and J. Bello, "An analysis/synthesis framework for automatic f0 annotation of multitrack datasets," 10 2017.

[29] A. Camacho and J. G. Harris, "A sawtooth waveform inspired pitch estimator for speech and music," *The Journal of the Acoustical Society of America*, vol. 124, no. 3, pp. 1638–1652, 2008.

[30] N. Jouppi, C. Young, N. Patil, and D. Patterson, "Motivation for and evaluation of the first tensor processing unit," *IEEE Micro*, vol. 38, no. 3, pp. 10–19, 2018.

[31] C. AI. Edge tpu performance benchmarks. [Disponível online; acesso em 09-Novembro-2024]. [Online]. Available: https://coral.ai/docs/edgetpu/benchmarks/

[32] M. Nagel, M. Fournarakis, R. A. Amjad, Y. Bondarenko, M. van Baalen, and T. Blankevoort, "A white paper on neural network quantization," 2021. [Online]. Available: https://arxiv.org/abs/2106.08295

[33] Y. Sun and A. M. Kist, "Deep learning on edge tpus," 2022. [Online]. Available: https://arxiv.org/abs/2108.13732

[34] J. Wook Kim. Crepe pitch tracker. [Disponível online; acesso em 09-Novembro-2024]. [Online]. Available: https://github.com/marl/crepe

[35] C. AI. Usb accelerator. [Disponível online; acesso em 09-Novembro-2024]. [Online]. Available: https://coral.ai/products/accelerator

[36] P. P. Ltd. Rubber band library. [Disponível online; acesso em 09-Novembro-2024]. [Online]. Available: https://breakfastquay.com/rubberband/

[37] P. Głomski. pylibrb. [Disponível online; acesso em 09-Novembro-2024]. [Online]. Available: https://pypi.org/project/pylibrb/