Universidade Federal de Pernambuco
Centro de Informática

Bacharelado em Ciência da Computação

Uma adaptação do *client-side prediction* para ambientes musicais colaborativos *online*

Luiz Henrique Tavares Caúla

Trabalho de Graduação

Recife
29 de abril de 2021

Universidade Federal de Pernambuco
Centro de Informática

Luiz Henrique Tavares Caúla

Uma adaptação do *client-side prediction* para ambientes musicais colaborativos *online*

Trabalho apresentado ao Programa de Bacharelado em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.

Orientador: *Filipe Carlos de Albuquerque Calegario*

Recife
29 de abril de 2021

Agradecimentos

Ao meu professor orientador, Filipe Calegario, que apoiou a ideia original ainda na fase de concepção, me incentivou a aprofunda-la e me orientou brilhantemente neste curto tempo. Também, ao professor Giordano Ribeiro, que não apenas aconselhou algumas decisões para este trabalho, como lecionou meus cursos favoritos na graduação e me deu oportunidade de ser seu monitor.

Aos meus colegas de turma, que estiveram presentes em todos os momentos da minha graduação e me forneceram momentos inesquecíveis e incríveis. Aos meu companheiros do PET-Informática que, apesar do pouco tempo que participei ativamente, me proporcionaram uma experiência incrível. Aos meus colegas e gestores de trabalho no estágio e em meu primeiro emprego, que me guiaram fenomenalmente para que pudesse me tornar o profissional que sou hoje. Minha profunda gratidão a todos e todas.

Aos meus pais, meu irmãos e irmã, à Nazaré, à Marilia e à minha namorada Lucylle, que, desde o momento que me conheceram, me forneceram todo suporte que precisei. Se conseguir deixá-los orgulhosos, considero meu objetivo aqui cumprido.

Ao Centro de Informática da UFPE, pelos anos, amigos e momentos sensacionais que não esquecerei.

Resumo

Devido à alta sensibilidade do aparelho auditivo humano, para performar em conjunto com outros artistas, músicos necessitam que haja pouca latência entre a saída dos instrumentos de seus colegas e seu retorno. Dessa forma, proporcionar um ambiente colaborativo em tempo real via Internet torna-se um desafio pertinente na área de Computação Musical e Rede de Computadores. Algumas soluções procuram otimizar a conexão entre os computadores construindo infraestruturas dedicadas, outras abandonam o requisito de tempo real por completo e entregam experiências assíncronas. No entanto, tais abordagens não abrangem, de forma síncrona, casos em que não haja acesso a uma conexão de alta velocidade ou que exista uma grande distância entre os participantes.

Este trabalho propõe, dessa forma, uma adaptação do algoritmo *Client-Side Prediction* para ambientes colaborativos musicais *online*, utilizado originalmente em *videogames*, experimentando dois modelos preditivos para gerar novas sequências de áudio baseando-se nas anteriores - predição de sequências com LSTM (*Long Short-Term Memory*) e indexação e identificação de sequências com DTW (*Dynamic Time Warping*). De tal forma, espera-se que, não sendo necessária a espera pela saída do cliente transmissor, haja uma redução da percepção de latência por parte dos participantes.

Utilizando duas métricas de sucesso - corretude das previsões e tempo de processamento - avaliamos que o modelo gerador com LSTM não performou bem comparado ao modelo indexador com DTW, que apresentou resultados bastante promissores e que pode ser utilizado para músicas de gêneros com tendências menos improvisacionais.

Palavras-chave: latência, áudio, predição de sequência, streaming, transmissão, online, música, client-side prediction, rollback, dtw, lstm

Sumário

Lista de Figuras

Lista de Tabelas

CAPÍTULO 1

Introdução

A performance artística musical, quando praticada em conjunto, requer alto nível de colaboração entre os participantes. Em música, sobretudo gêneros com tendências improvisacionais como *jazz*, *blues* e *rock*, o ato de ouvir e reagir ao som de seus companheiros é tão importante quanto aquele produzido individualmente. Dessa forma, o *feedback* auditivo de baixa latência dos instrumentos tocados é fundamental para que haja uma sensação fluida entre os participantes.

Normalmente, músicos performando em conjunto em um mesmo ambiente físico raramente possuem problemas relacionados à latência. No entanto, em um contexto de distanciamento social, encorajado durante à Pandemia de COVID-19, músicos ao redor do mundo viram-se obrigados a transferirem esse ambiente para um virtual *online*. Além dos *delays* causados pelas conversões de sinais analógicos para digitais e vice-versa e do tempo de escrita no *buffer* em memória [1], a latência apresentada pela transmissão de pacotes pela Internet apresenta-se como um dos maiores desafios para sobrepor, onde, a depender de fatores como distância entre os músicos e velocidade da rede, pode ser o maior gargalo do processo de *streaming* de áudio.

Aplicações comuns de videoconferências, como *Zoom*, *Google Meet* e *FaceTime*, que oferecem plataformas para conversações em tempo real, também possuem baixa tolerância a latência, permitindo no máximo 150 ms para manter uma conversa inteligível [2] - limite atingível em velocidades medianas de conexões. No entanto, no contexto da prática musical, o limite é ainda menor, variando entre 10 ms e 55 ms [3], demonstrando que tais aplicações não podem ser utilizadas para esse propósito.

Para lidar com esse problema, *softwares* voltados especificamente para a colaboração musical *online* apresentam uma variedade de abordagens diferentes. *LoLa* [4], *SoundJack* [5] e *JamKazam* [6], por exemplo, implementam otimizações na camada de rede - como conectar clientes diretamente entre si via *P2P* (*peer-to-peer*) - oferecendo latências razoáveis entre distâncias medianas. Outras aplicações, como o *Jammr* [7] e *JamTaba* [8], dispensam o requisito de tempo real e apresentam soluções assíncronas, nas quais os músicos ouvem os áudios deliberadamente atrasados produzidos por seus companheiros, porém, de acordo com os batimentos sincronizados de um metrônomo.

No entanto, tais abordagens não abrangem casos onde músicos residem entre grandes distâncias ou não é possível ter acesso a conexões dedicadas e *hardwares* de alto valor financeiro, de forma a ainda oferecer uma plataforma de performance colaborativa em tempo real.

Ao observar o contexto de videogames, encontramos requisitos de latência similares. Gêneros que utilizam reações como mecânica de jogabilidade, como luta e FPS (*first-person shooter*), para oferecerem aos jogadores uma experiência fluida, necessitam de latências máximas de até 100 ms [9]. O algoritmo mais popular e efetivo para solucionar esse problema, *client-*

side prediction [10], baseia-se em prever os próximos *inputs* imediatos dos jogadores e agindo antes mesmo que os dados de seu oponente sejam transmitidos; dessa forma, removendo a necessidade inicial de espera. Uma vez que os *inputs* são recebidos, estes são comparados com a previsão realizada e, caso sejam incongruentes entre si, o jogo é retornado ao estado anterior do momento da previsão inicial.

Por possuir requisitos semelhantes de baixa latência, a mesma implementação baseada em previsões tem o potencial de resolver o problema descrito anteriormente para ambientes musicais colaborativos *online*. Caso seja possível prever os próximos sinais digitais produzidos pelos artistas remotos, não haveria necessidade de espera e, portanto, a latência de rede tornaria-se irrelevante.

Evidentemente, as diferenças entre os contextos de *videogames* e práticas musicais não são negligenciáveis. Ao contrário dos jogos, por apresentar uma linearidade no tempo, não é possível retornar ao último momento da música anterior à previsão imediata sem prejudicar a experiência dos artistas. Além disso, a natureza discreta dos *inputs* dos *videogames* e a quantidade bruta de dados produzida é ínfima em comparação a áudios digitais - estima-se que os jogadores profissionais de *Super Smash Bros. Melee* mais técnicos produzem em média 6 *inputs* por segundo [11]; uma transmissão de áudio com *sample rate* de 44,1 kHz produz, por definição, 44.100 diferentes valores no mesmo espaço de tempo [12]. Portanto, a aplicação do algoritmo de *client-side prediction* para transmissão de música *online* não é trivial, e a necessidade que o modelo preditivo seja o mais acurado possível é ainda mais importante.

Dois ciclos de estudos foram realizados para explorar essa abordagem. No primeiro ciclo, foram utilizados métodos de aprendizagem de máquina da biblioteca Keras [13], especificamente LSTM (*Long Short-Term Memory*)[14]. Modelos foram construídos e treinados com cortes de arquivos de áudio, reproduzindo trechos de músicas tocados por um único instrumento. A partir desses modelos, novas previsões de sequência foram geradas.

O segundo ciclo seguiu uma abordagem de criar um banco de dados de referência e, para cada nova entrada, identificou-se o corte mais semelhante. A semelhança foi calculada a partir um algoritmo de identificação baseado em DTW (*Dynamic Time Warping*)[15], implementado pela biblioteca Librosa [16]. Possuindo uma janela de referência identificada, a próxima sequência a ela foi assim escolhida como predição da continuidade da música.

Para avaliar os dois modelos, utilizamos dois critérios de sucesso: (1) quão corretas foram as previsões realizadas e; (2) o tempo necessário para gerar as previsões. Dados esses parâmetros, o modelo gerador com LSTM não teve boa performance, falhando nos dois critérios. Entretanto, o modelo indexador com DTW mostrou-se bastante promissor, tendo sucesso nas duas medidas. Em particular para músicos que performam a trilha de base das músicas, com menos improvisos, esse método pode ser explorado futuramente em ambientes reais.

No Capítulo 2, contextualizamos o problema enfrentado pelos ambientes musicais colaborativos *online* atualmente, explorando as atuais abordagens utilizadas que procuram solucioná-lo. No Capítulo 3, detalhamos nossa solução proposta - uma adaptação da técnica *client-side prediction* para os ambientes musicais - entrando também em detalhes sobre os dois modelos preditivos experimentados - o modelo gerador de novas sequências (LSTM) e o identificador e indexador de sequências (DTW). No Capítulo 4 e Capítulo 5, exploramos o primeiro e segundo ciclo de estudos respectivamente, onde simulamos um ambiente virtual para testar os modelos

preditivos propostos. Finalmente, no Capítulo 6, realizamos uma análise crítica dos resultados apresentados em nossos experimentos, apontando, também, possíveis trabalhos futuros para aprimorá-los.

CAPÍTULO 2

Contexto

Neste Capítulo, descreveremos em maiores detalhes o problema da latência no contexto de aplicações musicais colaborativas *online*, além de explorar soluções que procuram resolvê-lo. Por último, demonstraremos a inspiração da solução proposta, o *client-side prediction*, dentro do contexto de videogames, como ele se relaciona com o problema original e como propomos aplicá-lo no ambiente musical.

2.1 O problema

A execução musical praticada em conjunto é altamente colaborativa. Em gêneros com tendências improvisacionais, como *jazz*, *blues* e *rock*, é comum que hajam um ou mais integrantes que usam como base os ritmos e conjunto de acordes praticados por seus colegas para executar *solos* que reagem de acordo com as mudanças da harmonia. É importante, portanto, que haja um *feedback* fluido entre o improvisador e a base musical para que todos possuam conhecimento se o que está sendo tocado está de acordo com suas intenções.

Em gêneros mais metódicos, como a música clássica, onde os músicos seguem instruções de partituras e um condutor, reação à mudanças é menos importante. Porém, ainda é necessária a garantia que todos estejam tocando em sincronia uns com os outros. Para o condutor, é importante que possua ciência do que está sendo executado por cada conjunto de instrumentos, de forma a garantir totalmente a harmonia e ritmo entre as partes [17].

A tolerância máxima de latência na prática musical é bastante restritiva. Para instrumentos digitais, Wessel e Wright sugerem não mais que 10 ms entre o *input* do artista e o *output* do som [18]. Carôt, por sua vez, define o objetivo de alcançar latências menores que 30 ms para que os músicos tenham a sensação de estarem tocando em um mesmo ambiente físico [19]. Ele observa também que, após esse limite, os artistas mudam a forma sobre como performam para adaptarem-se.

Quando performado em conjunto no mesmo ambiente físico, normalmente não há percepção de latência entre os integrantes. Tal latência é baseada na velocidade em que o som se propaga no ar em temperatura ambiente que, em uma velocidade de 343,2 m/s [20], produz cerca de 3 ms de atraso por metro de distância entre cada músico, sendo, portanto, negligenciável para a sensação de fluidez dos músicos.

No entanto, em ambientes colaborativos *online*, outros fatores podem interferir na latência total. Primeiramente, para que os computadores do participantes consigam processar o áudio, é necessário que haja uma conversão dos sinais analógicos para digitais. Tal processamento consome cerca de 1 ms [1], podendo ser negligenciado. Uma vez convertidos, os sinais pre-

cisam ser escritos e lidos no *buffer* em memória, processo que dura entre 10 ms e 12 ms [1], dependendo do poder de processamento das máquinas envolvidas. Ambos processos ocorrem no lado do transmissor (aquele que produz a música) e dos receptores, portanto, dobrando o tempo total.

O maior gargalo, no entanto, pode ocorrer na transmissão dos pacotes de dados via Internet. Devido à abordagem de "melhor esforço" na qual a Internet foi originalmente projetada - sob a suposição de que não é possível garantir o recebimento de todos os pacotes transmitidos - protocolos de transmissão de voz como *VoIP* (*Voice over Internet Protocol*) e serviços provedores videoconferências necessitam implementar medidas compensatórias, como grandes *buffers* de rede e retransmissão de pacotes. Tais medidas, consequentemente, garantem a corretude dos dados transmitidos, sob o custo do aumento na latência total [19].

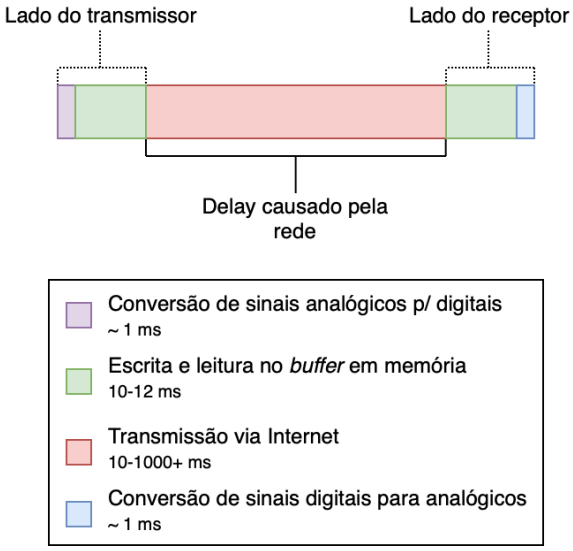


Figura 2.1: Latências de cada fase do processo de *streaming* de áudio pela Internet

Aplicações comerciais de videoconferências, como o *Zoom*, *Google Meet* e *FaceTime* também possuem sensibilidade de tempo para manter conversas compreensíveis - a *Cisco* define a latência máxima aceitável de uma implementação *VoIP* em até 150 ms [2]. Este limite pode ser alcançado por conexões de velocidades medianas, mesmo considerando fatores como processamento de áudio, distância entre os participantes e infraestruturas de rede compartilhadas. No entanto, para a prática colaborativa musical, onde tolerância máxima é bastante restritiva, o uso de aplicações que implementam *VoIP* mostra-se inviável.

A tolerância de atraso para a performance musical, portanto, é um dos maiores desafios na implementação de ambientes colaborativos musicais *online*. Afinal, mesmo considerando a velocidade da luz no vácuo (aproximadamente 300 km/ms [21]) a menor distância possível para obter uma latência ótima de 10 ms é de aproximadamente 3.000 km - um pouco mais que a distância entre Recife, PE e Porto Alegre, RS[1].

Dessa forma, métodos para redução da latência entre os participantes - ou a sensação de sua existência - possuem suma relevância para músicos. Em contextos onde a presença física dos

[1]Distância calculada utilizando o Google Maps. ©2021 Google

participantes não é possível de se obter, como o recomendado pelo distanciamento social como método de prevenção de contaminação na Pandemia de COVID-19, a colaboração *online* é o único meio onde músicos conseguem tocar em conjunto.

2.2 Estado da arte

Como forma de reduzir o efeito da latência em transmissões *online* de música, nós propomos a previsão local de áudio baseado em entradas anteriores. Não foram encontradas soluções, estudos ou aplicações que procuram resolver esse problema dessa forma.

No entanto, para resolver o problema da latência em ambientes musicais, identificamos duas abordagens principais: (1) otimizações na camada de transporte da Internet e (2) criação de ambientes assíncronos, onde os músicos não performam em tempo real uns com os outros.

2.2.1 Otimizações na camada de transporte

Esta abordagem procura atacar o problema de forma mais linear, implementando melhorias na conexão pela Internet entre os participantes ou utilizando infraestruturas de rede específicas.

LoLa (*LOw LAtency audio visual streaming system*) [4], um sistema de *streaming* audio visual desenvolvido pelo *Conservatorio di Musica G. Tartini*, consegue atingir conexões de baixa latência, utilizando infraestruturas de rede dedicadas. Foi usado em diversas apresentações de até 3.500 km de distância entre os músicos entre os anos de 2010 e 2013 [22]. No entanto, deixa muito claro que sua solução não é necessariamente acessível, sendo recomendado no mínimo 1 Gigabit por segundo de banda em todas as pontas. De acordo com estudo realizado pela Viavi Solutions em 2019, apenas 5% da população mundial possui conexões tão rápidas [23], e a média de velocidade mundial é de apenas 97.52 Megabits por segundo, pelos dados apresentados pelo Speedtest Global Index [24]. Portanto, para o público em geral, não é uma solução de fácil acesso.

SoundJack [5], por outro lado, utiliza-se de alguns métodos que o torna mais acessível para o usuário comum. Ele consegue atingir velocidades mais rápidas que aplicações comerciais como *Zoom*, *Teams* e *FaceTime* por dois fatores: (1) conecta usuários diretamente através de P2P (*peer-to-peer*), ao contrário das soluções baseadas em *VoiP* citadas na Seção 2.1, que transportam dados entre servidores centrais e; (2) não otimiza a qualidade o áudio/vídeo automaticamente, deixando como responsabilidade do usuário; caso prefiram, os músicos podem optar por conexões de menores latências assumindo o custo de obter piores qualidades de áudio. Nas configurações recomendadas, infraestruturas comerciais comuns residenciais são suportadas via Ethernet. Entretanto, *SoundJack* requer um poder computacional relativamente alto para atingir baixas latências - recomenda no mínimo processadores i7 Quad Core, custando cerca de R$2456,90 [2], também oferecendo um dispositivo próprio dedicado à aplicação, vendendo por €226,05 [3].

A aplicação comercial *JamKazam* [6] também baseia-se em entrega síncrona dos pacotes de

[2]Preço encontrado na Amazon Brasil em 04/04/2021.

[3]Preço encontrado na Symonics em 04/04/2021.

áudio para construir um ambiente musical colaborativo *online*. De acordo com seus resultados apresentados, é possível performar em conjunto com baixas latências onde os músicos estejam em um mesmo estado, numa distância média de 490 km [25]. Porém, entre maiores distâncias ou infraestruturas não baseadas em fibra ótica, a latência excelente recomendada pela aplicação de 30 ms [26] é difícil de ser obtida.

Em termos de acessibilidade para o público geral, a solução *open source JackTrip* fornece guias de instalação para Raspbery Pi [27], além de uma instalação simples para Linux, OS X e Windows. Também de código aberto, o SonoBus [28] fornece um *download* gratuito, além de guias de instalação e dicas para melhoria na latência.

2.2.2 Criação de ambientes assíncronos

Uma solução que possibilita a percepção de baixa latência é abdicar do requisito de entregar uma experiência em tempo real. Caso seja possível atrasar a entrega dos pacotes de forma não perceptível e, portanto, aumentando a janela mínima de espera, não é necessário possuir baixas latências.

A aplicação *jammr* [7] aproveita o conceito de progressão de acordes da teoria musical a seu favor. A cada *loop*, os participantes ouvem o que foi tocado nos últimos quatro compassos pelos seus colegas. Dessa forma, por não ser necessário manter as performances em sincronia, há uma tolerância muito maior às latências causadas pela camada de transporte. No entanto, tal solução necessita que os músicos toquem a mesma progressão em *loop*, funcionando bem para improvisações simples (*jam sessions*), porém, para músicas mais complexas, o sistema não é ideal.

O *JamTaba* [8], um cliente *open source* de servidores NINJAM [29], utiliza uma solução semelhante. A aplicação funciona utilizando latências grandes, porém sincronizadas entre os participantes. Para cada intervalo, os músicos gravam o áudio e, uma vez finalizado, a gravação é reproduzida pelos outros clientes. Então, todos ouvem a última gravação de seus colegas, porém, ouvindo a si mesmo em tempo real.

2.2.3 Considerações

Cada abordagem para o problema da latência apresenta um conjunto de vantagens e desvantagens entre si. Dessa forma, podemos compará-las na Tabela 2.1, juntamente à nossa solução proposta, de acordo com quatro características: (1) se entregam os áudios em tempo real; (2) se oferecem baixo custo financeiro para usá-las; (3) se funcionam bem em conexões lentas, de acordo com as valores demonstrados na Seção 2.1 e; (4) se suportam improvisações dos músicos.

Observa-se, portanto, que nenhuma das soluções avaliadas consegue fornecer, simultaneamente, uma experiência em tempo real, que seja acessível e funcione em conexões lentas. Propomos, portanto, no Capítulo 3, uma adaptação do algoritmo de *client-side prediction*, utilizada em videogames *online*, que proporciona uma experiência com essas três características.

Pela natureza preditiva desse algoritmo, improvisações musicais não podem ser suportadas, umas vez que sua tendência imprevisível torna essa tarefa bastante desafiadora. Outras soluções, que reproduzem o áudio produzido pelos músicos fidedignamente, cumprem esse

		Tempo real	Baixo custo	Funciona em conexões lentas	Suporta improvisações
Soluções síncronas	LoLa	X			X
	SoundJack	X			X
	JamKazam	X			X
	JackTrip	X	X		X
	SonoBus	X	X		X
Soluções assíncronas	jammr		X	X	X
	JamTaba		X	X	X
Solução proposta	Client-side prediction	X	X	X	

Tabela 2.1: Matriz morfológica comparando as diferentes abordagens para solucionar o problema da latência em ambientes musicais colaborativos *online*.

requisito perfeitamente.

2.3 Predição no lado do cliente

No contexto de videogames, alguns gêneros possuem problemas semelhantes aos que os ambientes musicais enfrentam. Aqueles que utilizam reações como uma das mecânicas de *gameplay*, como luta e FPS (*first-person shooter*), para implementar funcionalidades *online*, necessitam que haja pouco atraso entre os *inputs* dos jogadores.

Há duas vertentes de implementação de jogabilidade *online* em videogames - (1) *delay-based* (“baseado em atraso”)[30] e (2) *client-side prediction* (“predição no lado do cliente”, popularmente conhecido como *Rollback Netcode*)[10].

2.3.1 *Delay-based*

Nessa abordagem, todos os *inputs* dos transmissores são esperados antes que as ações correspondentes possam ser executadas [30]. Essa implementação é trivial e garante a corretude dos dados transmitidos; no entanto, para conexões de alta latência, nas quais o tempo de transmis-

são via Internet é maior que a latência de ação local, os jogadores experienciam uma sensação de "lentidão" ou "travamento".

Esse limite máximo de latência de transmissão varia a cada jogo. Para oferecerem uma experiência fluida, de acordo com o tempo de reação à estímulos visuais, é esperado que não haja mais que 100 ms de atraso entre as ações dos jogadores [9]. Esse limite, no entanto, é apenas uma estimativa - idealmente, a melhor implementação deve garantir que não haja diferença entre jogar *online* ou localmente. Esse limite dependerá de especificações de cada jogo, como FPS (*frame-per-second*), a latência natural causada pelo dispositivo de controle (*input delay*) e suas mecânicas de jogabilidade. Desenvolvedores também podem adicionar um atraso artificial, objetivando aumentar a tolerância de tempo causado pela transmissão dos pacotes via Internet.

No contexto de ambientes musicais *online*, podemos comparar esse método com as soluções síncronas citadas na Subseção 2.2.1. Nesses casos, também é notável alta sensibilidade à latência causada pela transmissão de pacotes via Internet. As soluções citadas, portanto, focam em reduzir o tempo total de transmissão, mas são limitadas por fatores externos, exigindo bandas de alta velocidade e/ou proximidade geográfica.

2.3.2 *Client-side prediction*

A implementação do *Client-side prediction*, ilustrada na Figura 2.2[4], por outro lado, aumenta a tolerância da espera dos pacotes propondo a previsão dos *inputs* dos jogadores antes que cheguem via Internet utilizando dados já recebidos anteriormente. Caso as previsões sejam incorretas, o estado de jogo no momento em que a previsão foi realizado é retornado, dando origem ao o nome popular de "*rollback*" (reversão) [30].

A necessidade da implementação do *client-side prediction* surge em 1996, em um contexto onde a maioria dos usuários de Internet possuíam conexões discadas com banda entre 28 Kb/s e 34 Kb/s [31]. *Duke Nukem 3D*, um jogo do gênero FPS (*first-person shooter*, "tiro em primeira pessoa"), foi pioneiro na utilização desse algoritmo para prover sincronia entre os jogadores *online* [32], que podem possuir diferentes velocidades de Internet ou estarem distantes entre si. Os *inputs* dos jogadores remotos eram previstos no lado do cliente e enviados a um servidor central, que comparava os *inputs* corretos e enviava as correções necessárias.

O modelo de previsão, isto é, o algoritmo utilizado para prever os *inputs* dos jogadores remotos, pode variar de acordo com as necessidades específicas dos jogos. O modelo proposto por Bernier, utilizado no jogo *Half-Life* (1998), apenas repete os últimos *inputs* reconhecidos pelo servidor de sincronização [10]. Esse modelo assume que os *inputs* tendem a se repetir com frequência a cada *frame*, portanto, apenas repeti-los e corrigir aqueles que não se enquadram funciona bem para a maioria dos casos.

A utilização de *client-side prediction* expande-se para diferentes gêneros, como o de luta, e sua adoção é vista positivamente pelos jogadores [33]. Por não necessitar esperar os *inputs* dos adversários para execução do jogador localmente, a sensação de fluidez é mais aparente do que as implementações baseadas em atraso.

[4]Tradução da imagem criada por GerardSN, CC BY-SA 4.0. Disponível em https://commons.wikimedia.org/w/index.php?curid=97477279.

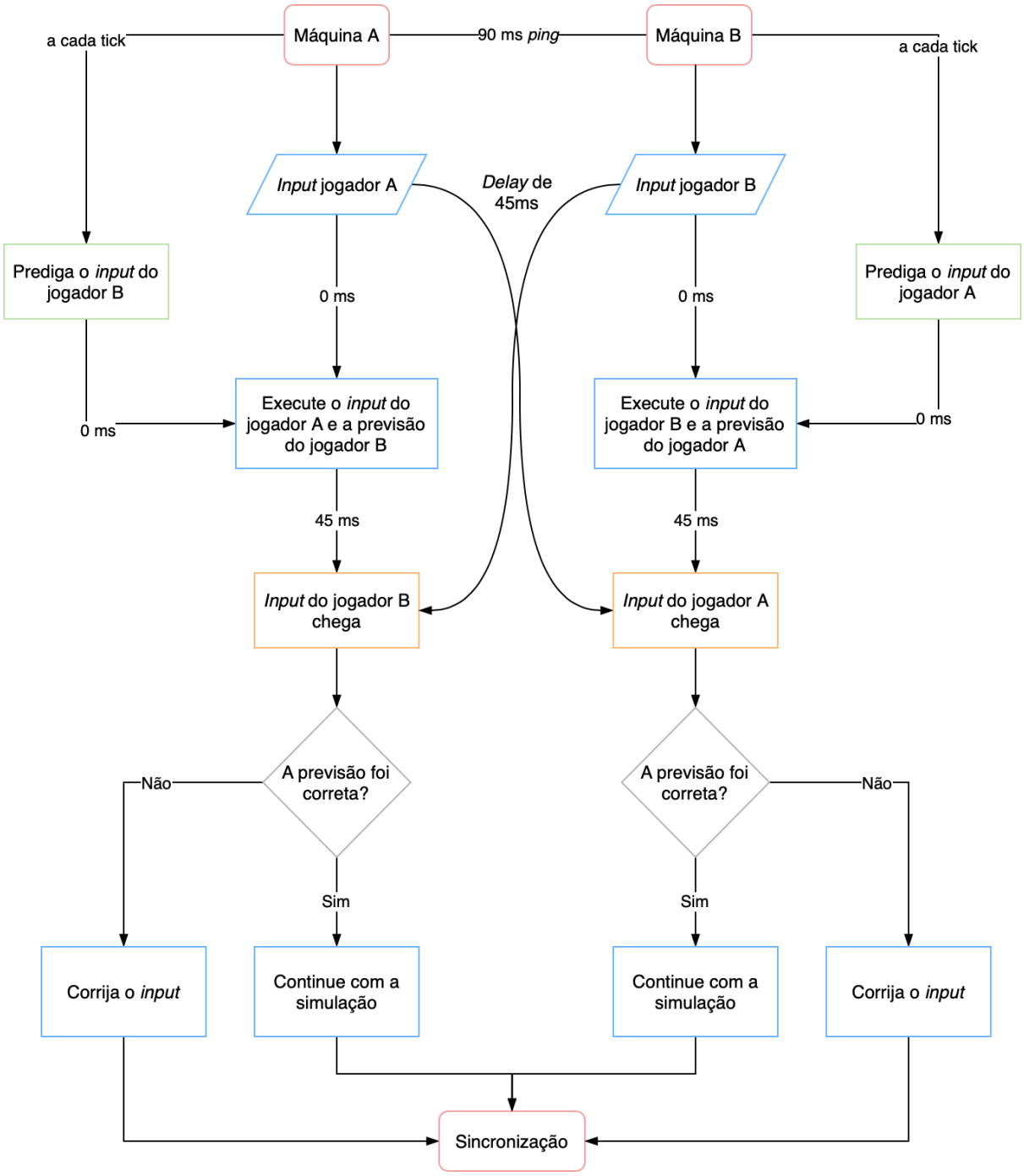


Figura 2.2: Diagrama demonstrando o processo de execução e sincronização dos *inputs* de dois jogadores (com *ping* de 90 ms entre eles) em um jogo *online* utilizando *client-side prediction* em um modelo *peer-to-peer*.

No entanto, por não garantir a corretude dos *inputs* imediatamente após a previsão, é necessário que haja a correção do estado do jogo em caso de erro de previsão. Essa correção pode causar, por exemplo, que um jogador perceba que seu adversário está em uma determinada posição no espaço e, em outro momento, mudar de lugar instantaneamente, causando uma sensação de "teletransporte". Em jogos de FPS, algumas medidas podem ser tomadas para amenizar tal sensação, como algumas animações que deixam ambígua a posição real dos jogadores [34]. Porém, inevitavelmente a regressão de estado ocorrerá, principalmente em casos onde a latência é alta e as previsões realizam mais suposições errôneas.

CAPÍTULO 3

Proposta de solução

No contexto de videogames que utilizam reações visuais/auditivas como mecânica essencial de jogabilidade, existe uma alta sensibilidade à latência de *inputs*. Como forma de mitigar esse problema, a técnica de previsão no lado do cliente, como descrita na Seção 2.3, é implementada em diversos *games* e é muito bem vista pelos jogadores [33].

Similarmente, ao observar ambientes musicais *online*, percebe-se o mesmo requisito de baixa latência para manter a sensação fluidez e sincronia entre os participantes, como descrito na Seção 2.1. Portanto, questiona-se: é possível aplicar técnicas de predição no lado do cliente nesse contexto, de forma a permitir sessões artísticas satisfatórias entre os músicos?

Evidentemente, apesar de compartilharem um requisito de baixa tolerância a latência, as naturezas dos problemas são significativamente divergentes. A mera implementação de previsão no lado do cliente no contexto musical implica em dois grandes problemas: (1) a impossibilidade de retornar ao último momento da música em caso de erro na previsão e; (2) a enorme dimensionalidade da representação digital de áudio.

A técnica de previsão no lado do cliente, quando aplicada a videogames, baseia-se no fato que os eventuais retornos aos estados anteriores às previsões em caso de erros não são suficientemente prejudiciais à experiência do jogador. No contexto musical, no entanto, devido sua natureza contínua na linha do tempo, o conceito de estados não pode ser replicado e, portanto, não faz sentido retornarmos a um momento anterior.

Ademais, a quantidade de *inputs* produzidos pelos jogadores é ínfima quando comparada a representação de áudio digital. Estima-se que os jogadores profissionais mais técnicos de *Super Smash Bros. Melee*, um jogo de luta em plataformas, produzem em média cerca de 6 *inputs* por segundo [11], variando de acordo com o momento do jogo. Por outro lado, uma transmissão de áudio com *sample rate* de 44,1 kHz produz consistentemente, por definição, 44.100 diferentes valores no mesmo espaço de tempo [12]. O modelo preditivo proposto por Bernier [10] replica os últimos *inputs* reconhecidos pelo servidor; se aplicássemos o mesmo método no contexto musical, efetivamente estaríamos "atrasando" os *inputs* de áudio, portanto, perdendo a sincronia entre os participantes.

Portanto, a acurácia do modelo preditivo em ambientes musicais é de suma importância, uma vez que a "volta no tempo" é impossível. Atingir esse alto nível de acurácia, por sua vez, é um enorme desafio, dada a dimensionalidade da representação de áudio digital. Naturalmente, o tempo total gasto na geração da previsão não pode exceder o tempo da janela prevista - caso ocorra, retornaremos ao mesmo problema enfrentado pelas soluções síncronas apresentadas na Seção 2.2.1.

3.1 Adaptação do *client-side prediction* no contexto musical *online*

Propomos, então, uma variação da implementação de Bernier [10] de previsão no lado do cliente, para sua aplicação no contexto musical (Figura 3.1). Similarmente à técnica original, janelas de áudios são previstas baseando-se nas entradas anteriores. Entretanto, nenhuma correção é realizada em casos de erro, mantendo a linearidade da música.

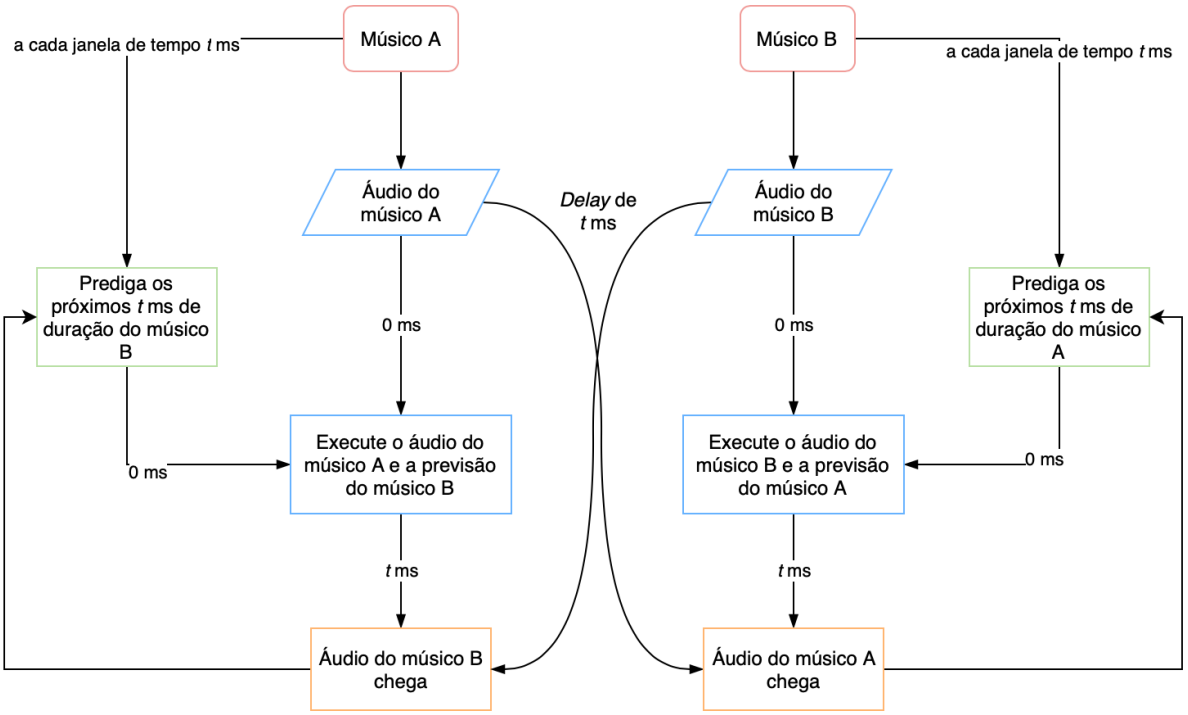


Figura 3.1: Diagrama demonstrando a adaptação do algoritmo de previsão no lado do cliente aplicado para ambiente colaborativo musical *online*. Na imagem, t representa a duração da janela de previsão, medido em milissegundos.

Em Bernier, a janela de tempo de cada conjunto de previsões é definida de acordo com o FPS do jogo e a velocidade de conexão entre os participantes. Para adaptação musical, além do tempo de ida e volta dos pacotes entre os participantes (denominado *ping*), propomos a utilização de outros parâmetros para a decisão dessa janela, como o BPM (batimentos por minuto) da música tocada junto à informação dos compassos musicais e também, por simplicidade, valores múltiplos de 1 segundo. A escolha dessa janela é fundamental - durações muito longas possuem muita informação, porém, são mais difíceis de processar e mais suposições terão que ser realizadas na previsão; e o inverso ocorre para janelas muito curtas.

Propomos, portanto, dois modelos preditivos para música, explorados em dois ciclos de pesquisa. No Capítulo 4, no primeiro ciclo, utilizamos a arquitetura de aprendizagem de máquina em camadas LSTM (*Long Short-Term Memory)* [14] para gerar sequências de sinais digitais, baseando-se nas entradas anteriores. Já no Capítulo 5, no segundo ciclo, usamos o algoritmo DTW (*Dynamic Time Warping*) [15] para identificar janelas semelhantes em uma base de dados e, a partir dessa informação, reproduzir a próxima janela de áudio, também armazenada na

mesma base de dados.

É válido mencionar que, no entanto, pela natureza imprevisível das improvisações musicais, esse caso de uso não deve ser bem aplicado em nossa solução proposta. Porém, para bases e sequências de acordes, onde é mais fácil prever as próximas entradas, o uso de nossa solução é melhor adequado.

3.2 Métricas de sucesso dos modelos preditivos

Como mencionado, existem alguns requisitos que os modelos preditivos propostos precisam cumprir para serem considerados bem sucedidos. Em cada ciclo, diferentes metodologias foram utilizadas para calcular e interpretar essas métricas.

3.2.1 Corretude das previsões

Devido à característica de linearidade de tempo que a música possui, não é possível "retornar" a um estado passado da música. Dessa forma, é importante que os modelos sejam os mais acurados possíveis.

Dada a natureza da predição, não é esperado que as sequências geradas sejam integralmente fidedignas às sequências reais. No entanto, isso não é um requisito para produzir resultados positivos, sendo realmente importante que os áudios previstos tenham a mesma "intenção" que a reprodução original.

Dessa forma, o critério de sucesso dessa métrica é, para as sequências previstas, que seja imperceptível, em tempo real, que haja diferenças entre as sequências originais; de forma que um músico, ao ouvir ambas as sequências, reagiria musicalmente da mesma maneira ou de maneira similar.

Note que, por se tratar de um critério abstrato, quantificar essa métrica objetivamente pode não representar bem o seu sucesso. Em cada um dos ciclos, utilizamos diferentes metodologias (demonstradas na Seção 4.1 e na Seção 5.1) para medi-la, visando sempre satisfazer o critério estabelecido.

3.2.2 Tempo de geração de previsões

Supondo que t é o tempo escolhido das janelas de previsão de áudio e p é o tempo levado para gerar essas previsões, é imprescindível para o sucesso dos modelos preditivos que $t \leq p$. Caso contrário, o tempo excedente de processamento cria um *delay* entre o que está sendo executado pelo músico remoto e o que está sendo reproduzido localmente e, consequentemente, causando dessincronia entre os músicos.

Ao contrário do critério de corretude na Subseção 3.2.1, essa métrica consegue ser medida precisamente. Será considerado bem sucedido o modelo preditivo que produzir sequências em um tempo menor que a duração do áudio gerado.

3.3 Coleta de dados e simulação do ambiente musical *online*

Para realizar os experimentos, procuramos simular, de forma simples, um ambiente colaborativo musical *online peer-to-peer*, do ponto de vista do músico local. Nosso conjunto de dados, dessa forma, contém pares de arquivos de música com as seguintes regras:

1. Ambos arquivos reproduzem a mesma sequência de uma música, porém, em performances diferentes;
2. Ambos arquivos consistem de apenas um instrumento;
3. Ambos arquivos consistem da mesma instância de instrumento, com as mesmas configurações de som e efeitos (ex.: reverberação).

A Regra 1 visa simular diferentes reproduções de uma a mesma sessão de uma música, porém, de formas diferentes, similarmente a como ser humano o faria. A Regra 2, apesar de não obrigatória em transmissões em casos reais, simplifica o processamento e geração de previsão dos áudios; além disso, não é esperado que vários músicos compartilhem do mesmo canal de transmissão. Por último, a Regra 3 complementa a as duas regras anteriores, garantindo a mesma sequência de áudio performada de diferentes maneiras pelo mesmo instrumento.

Para cada par, podemos nomear o primeiro arquivo *A* e o segundo *B*, onde *A* é classificado como o conjunto de treinamento o *B* o conjunto de teste. Dessa forma, conseguimos simular um ambiente onde um dos músicos possui o conjunto *A* treinado em sua máquina e está recebendo o conjunto *B* do músico remoto.

Para coletar arquivos com esses requisitos, a seguinte abordagem foi realizada, ilustrada na Figura 3.2: (1) buscou-se músicas onde a mesma sequência de acordes e notas era reproduzida em diferentes sessões (por exemplo, uma introdução que serve de motivo para a música); (2) depois, isolou-se apenas um instrumento; (3) identificou-se as sessões repetidas e, por último; (4) separou-se as sessões isoladamente e criou-se os dois arquivos, um para treinamento e outro para simulação do áudio trasmitido remotamente.

Essa metodologia de separação de arquivos implica em dizer que a adaptação proposta não pode utilizar nenhum dado futuro do arquivo *B*, pois, em um ambiente real, essa informação também não estaria presente. O arquivo *A*, por sua vez, pode ser analisado integralmente. Inclusive, tal análise pode ser realizada em momento anterior às previsões, já que o único tempo relevante a ser metrificado é o que foi levado para gerá-las. Em um ambiente real, o treinamento pode ser realizado antes que os músicos toquem juntos.

As músicas utilizadas nos experimentos foram *Hotel California* (*Eagles*, 1976) - tendo a trilha do violão acústico isolada - e *Message In A Bottle* (*The Police*, 1979) - onde a trilha da guitarra elétrica de base foi isolada.

Os arquivos coletados estão formatados com a extensão WAV, por sua simplicidade, além de suportar armazenamento de dados não comprimidos, nos fornecendo o máximo de informações possível das trilhas de áudio. O *sample rate* dos arquivos foi de 44,1 kHz e *bit depth* de 16 bits, a mesma qualidade que a escolhida pela mídia dos CD's [35].

Os experimentos para cada modelo, demonstrados no Capítulo 4 e Capítulo 5, foram escritos e executados utilizando a versão gratuita do *Google Colab*, uma aplicação que permite rodar

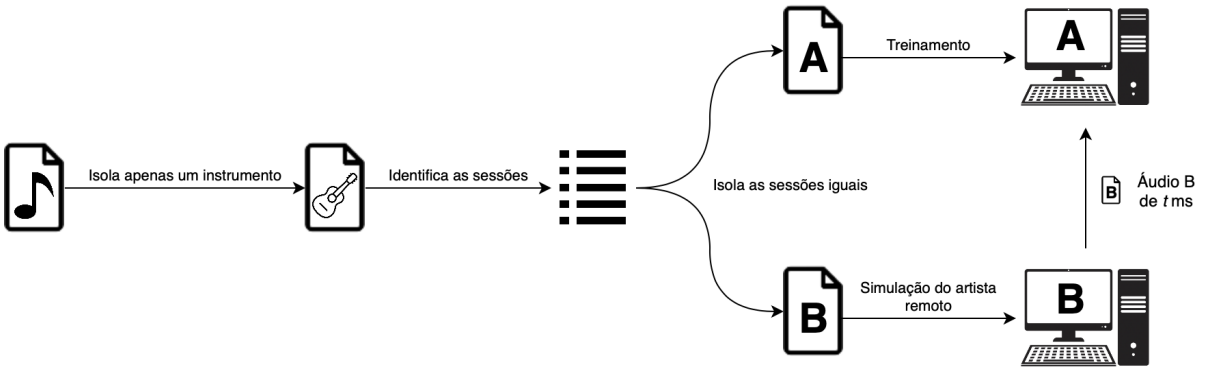


Figura 3.2: Processo de coleta de dados e simulação de um ambiente musical colaborativo *peer-to-peer*. Os arquivos *A* e *B* representam uma mesma sessão da música, porém, em performances diferentes.

Jupyter Notebooks em computadores remotos. Suas especificações são, para CPU, o *Intel(R) Xeon(R) @ 2.20GHz* e, para memória, *Nvidia Tesla K80* [36].

3.4 Modelos preditivos

Dada a solução proposta, este trabalho estuda a viabilidade de dois modelos preditivos - previsão através de (1) geração de novas sequências e (2) indexação e identificação de sequências. Ambas baseiam-se em extrair informações a partir de entradas anteriores de áudio e apontar um *output* sobre o que classificam ser o mais próximo do dado real futuro - no entanto, possuem diferenças sobre a forma como atingem esse objetivo.

3.4.1 Geração de novas sequências

Gerar novas sequências baseando-se em entradas anteriores encaixa-se intuitivamente no conceito de "previsão", como descrito no algoritmo de *client-side prediction*. Evidentemente, seu uso em uma adaptação para ambientes musicais foi explorado.

Em nossa adaptação do *client-side prediction*, anteriormente à sessão entre os músicos, um modelo de aprendizagem de máquina é treinado utilizando áudios já conhecidos - semelhantes, mas não iguais aos que serão produzidos pelo músico remoto. Este modelo receberá como entrada sequências de áudio de t ms de duração e produzirá saídas de mesma duração, como descrito na Figura 3.3.

Uma das possibilidades para prever sequências musicais pode ser encontrado no campo de continuação de músicas baseados em um estilo. Dhariwal et. al propõem o *Jukebox*, da *OpenAI*, que usa redes neurais para aprender diferentes gêneros e produzir continuações para músicas [37]. Para o uso em previsões, poderíamos treinar modelos com a música a ser tocada e, para cada pequena sequência, gerar uma continuação. Entretanto, os autores deixam claro que uma das limitações de seu uso é o tempo de renderização - cerca de 8 horas para cada

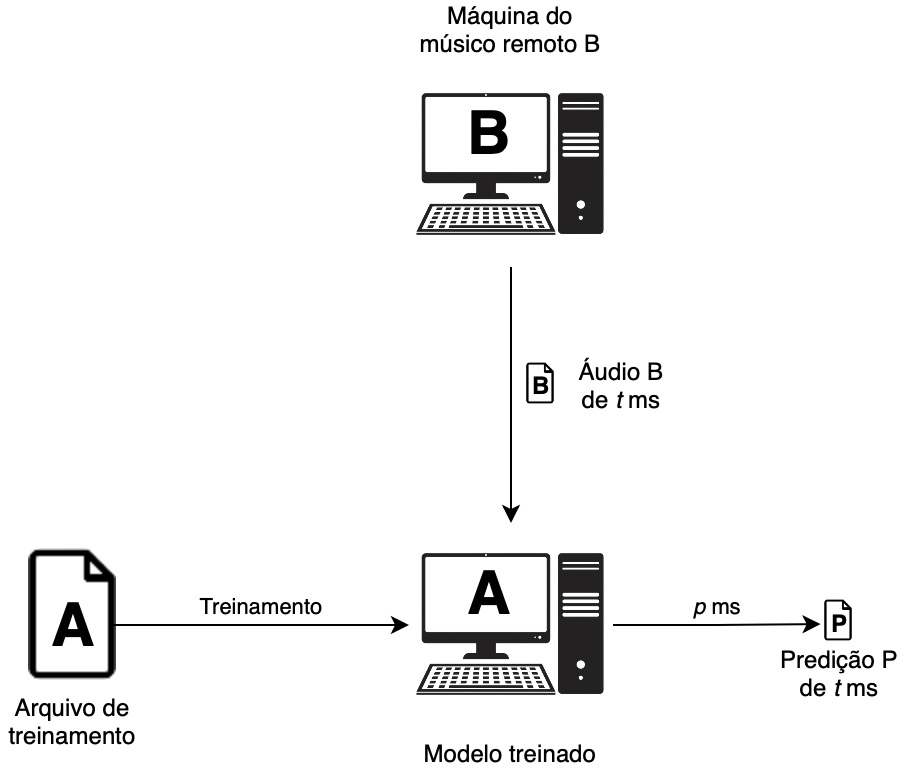


Figura 3.3: Processo de geração de novas sequências através de modelos treinados.

minuto de áudio gerado [37]. Como o tempo de geração das previsões é bastante sensível em nossa adaptação, essa abordagem foi descartada.

O campo da Aprendizagem de Máquina que visa gerar novas sequências baseando-se nas anteriores é a de Previsão de Séries Temporais (*Time Series Forecasting*). Essa abordagem procura aplicar técnicas para prever continuações de conjunto de dados onde o tempo é uma de suas dimensões [38]. Podemos classificar, portanto, que previsões de sequências musicais é um subconjunto dos problemas desse campo e, dessa forma, adaptá-la para uso em nossa solução proposta.

Uma das abordagens utilizadas para resolver problemas do conjunto *Time Series Forecasting* é a aplicação das redes neurais recorrentes LSTM (*Long Short-Term Memory*) [14]. Tais redes são capazes de aprender conexões de longo prazo. Dessa maneira, elas possuem um memorável poder de predição, funcionando bem em diversos problemas, sendo amplamente utilizadas atualmente.

A biblioteca *Keras* [13], escrita na linguagem de programação Python, implementa modelos de aprendizagem LSTM. Dessa forma, seu uso é bastante promissor para nossa adaptação musical de *Client-Side Prediction*. Exploramos-o no primeiro ciclo de estudos, descrito no Capítulo 4.

3.4.2 Indexação e identificação de sequências anteriores

No primeiro ciclo de estudos, uma das possibilidades estudadas foi, ao invés de treinar um grande modelo, utilizar cada janela de previsão e treinar pequenos modelos. Para isso, no momento da previsão, precisaríamos de alguma forma de identificar qual modelo melhor se encaixa na sequência de entrada. Com isso, pesquisamos métodos para realizar a identificação das sequências de um banco de dados e, com isso, surgiu a ideia de, ao invés de gerar novas sequências, apenas entregar a que veio após a identificada.

Para previsão de sequências, a geração de novos valores não é um requisito. Se possuirmos um conjunto de dados, reunindo diferentes sequências e informações sobre quais vieram após tais sequências, poderíamos apenas reproduzi-las. Dessa forma, sugerimos um modelo preditivo que, ao invés de fornecer sequências de áudio inéditas, identifica sequências similares e reproduz a que veio em seguida, como ilustrado na Figura 3.4.

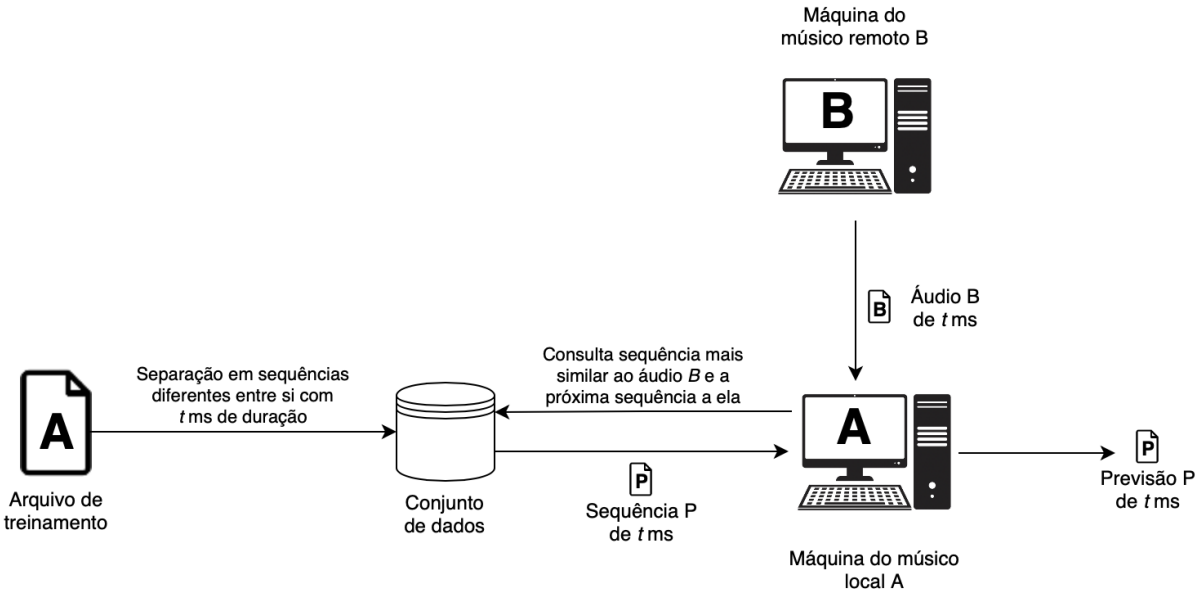


Figura 3.4: Processo de identificação de sequência semelhante à entrada B e entrega da previsão P, baseada na que veio após a sequência identificada.

Além do conjunto de regras apresentados na Seção 3.3, o conjunto de dados de referência que será consultado nesse modelo possui as seguintes regras:

1. Todos os arquivos necessitam representar diferentes sessões da música entre si;
2. Todos os arquivos necessitam possuir duração maior ou igual à t ms, onde t é a duração do arquivo utilizado para consulta de similaridade;
3. Todos os arquivos requerem que exista um, e apenas um, arquivo que represente a sequência tocada após ele mesmo.

A Regra 1 garante que, ao realizar uma *query* de similaridade entre a sequência de entrada e as armazenadas, haverá no máximo uma correspondência. Tal requisito é importante, pois,

se houver mais de uma, haverá mais de uma previsão, causando ambiguidade. A Regra 2 garante que as previsões entregues possuirão pelo menos a mesma duração que as sequências de entrada. Caso sejam menores, a diferença entre as durações causará um atraso de entrega, causando o problema descrito na Subseção 3.2.2. Finalmente, a Regra 3 garante que todos os arquivos necessitam possuir outro que represente o que foi tocado após ele, que será de fato entregue como previsão, evitando ambiguidades.

Portanto, tal modelo requer duas técnicas: (1) uma para indexar as sequências de música, respeitando as regras descritas e (2) uma para identificar similaridade entre a sequência de entrada (transmitidas pelo músico remoto) e as sequências armazenadas.

Neste trabalho, realizamos a primeira técnica manualmente. O processo é demonstrado no Capítulo 5.

Para a segunda técnica, estudamos alguns que métodos podem ser utilizados para identificação de sequências. A biblioteca Librosa [16], também escrita em Python, implementa algumas ferramentas que podem auxiliar em tal tarefa, como o cálculo da centroide espectral [39] de sequências de áudio, que encontra uma média central entre as frequências presentes em cada janela de tempo da sequência de entrada. Para identificação, poderíamos pré-calcular as centroides de cada sequência no conjunto de dados e comparar com a sequência transmitida pelo músico remoto. Entretanto, apenas a informação das frequências principais não é suficiente para identificar semelhança, uma vez que tal informação não varia tanto para cada janela, principalmente para aquelas que reproduzem o mesmo acorde ou nota.

A mesma biblioteca implementa o algoritmo DTW (*Dynamic Time Warping*) [15], um algoritmo utilizado para comparar e alinhar duas séries temporais. Em nossa adaptação de *client-side prediction*, podemos aplicar DTW nas sequências transmitidas contra o banco de dados. Caso haja uma janela semelhante, o algoritmo a entregará como *output* os *timestamps* representando o início e fim da identificação.

Dessa forma, o Capítulo 5 também demonstra como utilizamos o DTW em nossas experimentações, além de sua taxa de sucesso na identificação de janelas semelhantes. Caso essa taxa seja alta o suficiente e a indexação das previsões seja acurada, será possível reproduzir um áudio bastante similar ao transmitido pelo músico remoto.

3.4.3 Comparações entre os modelos

Apesar de ambos os modelos basearem-se em sequências anteriores para gerar ou apontar previsões musicais, as duas abordagens diferem na forma que funcionam. As diferenças implicam em alguns pontos que uma abordagem pode realizar melhor que a outra, assim como o inverso pode ocorrer, descritos na Tabela 3.1.

Por entregar sequências inéditas, o modelo preditivo gerador de novas sequências, descrito na Subseção 3.4.1, é mais flexível que o modelo indexador, descrito na Subseção 3.4.2. Afinal, entradas nunca vistas anteriormente sempre terão previsões geradas, mesmo que não sejam precisas com a realidade. Se aplicarmos sequências não vistas no modelo indexador, nenhuma janela será identificada e, portanto, nenhuma previsão poderá ser realizada acuradamente.

Além disso, para identificar uma janela precisamente, o modelo indexador requer mais informações que o modelo gerador, aumentando, portanto, o tempo da janela de previsão. Quanto maior a janela prevista, mais suposições terão que ser feitas sobre a previsão e, portanto, menor

	Flexível a novas entradas	Requer pouca informação	Predição de alta resolução	Rápido processamento
Modelo gerador	X	X		
Modelo indexador			X	X

Tabela 3.1: Matriz morfológica comparando os dois modelos propostos para adaptação do *client-side prediction* para música.

a probabilidade de replicar, de forma similar, a sequência de entrada.

Por outro lado, pela natureza complexa da representação de áudio, gerar uma sequência de alta qualidade de definição sonora, ainda que semelhante à sequência real do músico remoto, é um grande desafio para o modelo gerador. Por entregar sequências já armazenadas em alta qualidade, o modelo indexador sempre entregará sequências limpas ao ouvido humano, evitando desconfortos dos músicos.

Ademais, a utilização de LSTM, devido à complexidade das redes neurais, pode requerer grandes tempos de processamento [40] para gerar as previsões. A busca com DTW, apesar de ser um algoritmo de complexidade $O(N^2)$, pode ser otimizado para grandes bases de dados [41]. Como o tempo de previsão é sensível em nossa adaptação do *client-side prediction* para música, é essencial que o modelo preditivo seja eficiente.

CAPÍTULO 4

Ciclo 1: Geração de novas sequências com LSTM

No primeiro ciclo de estudos, exploramos a implementação de um modelo preditivo gerador de novas sequências baseado em entradas anteriores. Utilizamos como método principal a rede neural LSTM (*Long Short-Term Memory*) para implementá-lo.

Neste Capítulo, descreveremos a metodologia utilizada para realizar os experimentos, demonstraremos seus resultados e, a partir deles, tiraremos conclusões, de acordo com as métricas de sucesso descritas na Seção 3.2.

4.1 Metodologia dos experimentos

Para definir nossa metodologia, precisamos estabelecer seguintes passos necessários para realizar nossos experimentos: (1) como as camadas da rede neural foram organizadas; (2) a formatação dos dados de entrada para aprendizagem; (3) o processo de previsão das sequências.

4.1.1 Camadas da rede neural

A API Keras [13], uma biblioteca em Python voltada para desenvolvimento de aplicações *deep learning*, implementa a camada *LSTM* em seu modelo *Sequential*. Em nossos experimentos, utilizaremos uma camada de *input*, uma camada LSTM, que realizará as previsões, e uma camada *Dense*, que reunirá as camadas em uma única lista de *output*. A Figura 4.1 ilustra essa rede neural, considerando que queremos prever sequências simulando uma latência de 50 ms.

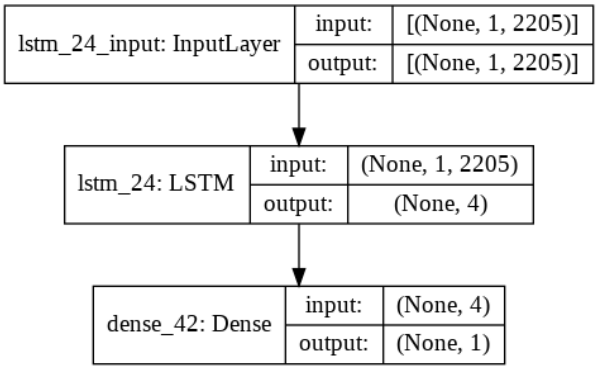


Figura 4.1: Visualização da rede neural utilizada nos experimentos com LSTM para uma latência simulada de 50 ms.

4.1.2 Formatação dos dados de aprendizagem

O nosso conjunto de dados, como exemplificado na Seção 3.3, consistirá de músicas com apenas um instrumento isolado, de forma a simular a transmissão do músico remoto. A música é representada como uma lista de sinais digitais, com uma frequência de 44.100 valores a cada segundo (44,1 kHz). Com um *bit depth* de 16 bits, os valores estão contidos no intervalo $[-32.768, +32.767]$.

Como entrada para realizar a aprendizagem, nossa rede neural requer duas listas:

1. Uma lista X de sequências numéricas de tamanho LB e;
2. Uma lista Y de sequências numéricas futuras de tamanho LB.

As listas estão organizadas de forma que, dado a sub-lista X_i, a sub-lista Y_i representa o que veio em seguida - isto é, para uma sequência na posição i na lista X, a sequência na mesma posição i na lista Y representa a sequência que veio logo a seguir. O valor de LB, representando o *lookback* que devemos utilizar para analisar a lista do passado e a geração de previsões, serve como o número de elementos de cada lista X_i e Y_i.

Por exemplo, supondo que queremos analisar a sequência numérica $(1,2,3,4,5,6)$. Para um $LB = 2$, as listas de entrada são organizadas de tal forma:

$$\begin{aligned} X &= ((1,2),(3,4)) \\ Y &= ((3,4),(5,6)) \end{aligned} \tag{4.1}$$

Note que o conjunto X não possui a sequência $(5,6)$. Isso se dá pois, dado o valor de $LB = 2$, não há nenhuma outra sequência futura de tamanho 2.

Para simplificar o processamento da aprendizagem e da previsão, normalizamos as sequências numéricas no intervalo $[-1, +1]$.

Em nossos experimentos, utilizamos os dois primeiros segundos da introdução da música *Hotel California* (*Eagles*, 1976), com a trilha isolada do violão acústico. Tal sessão se repete em um total de três vezes ao longo da música, porém, com diferentes execuções. A primeira delas, por possuir menos desvios do motivo da música, é usada como entrada para o treinamento.

O valor do *lookback*, em nossa adaptação, representa o tamanho da janela de previsão. A escolha desse valor está intrinsecamente conectada ao sucesso do algoritmo - valores pequenos possuem pouca informação, porém, são mais rápidos para processar; reciprocamente, o inverso ocorre com valores maiores. É importante mencionar que esse valor necessita ser maior que a latência apresentada pelo transporte dos pacotes pela Internet, afinal, queremos compensar por ela ao realizar previsões.

Portanto, para calcular o valor de LB, precisamos definir o valor de LAG, que simula a latência apresentada pela Internet. Em nossos experimentos, escolhemos dois valores: (1) 50 ms e (2) 100 ms. O primeiro valor foi definido utilizando de uma média de testes de *ping* entre Recife e o servidor 8.8.8.8, hospedado pelo Google e localizado no estado da Califórnia, Estados Unidos. O segundo é o dobro desse valor, de forma a compensar por possíveis picos de latência apresentados pela incerteza da entrega de pacotes pela Internet.

Então, dado os valores *SR* e *LAG*, onde *SR* representa o *sample rate* das sequências de áudio medido em kHz e *LAG* a latência simulada da Internet, medida em milissegundos, podemos calcular *LB* aplicando a seguinte fórmula:

$$LB = \frac{SR}{1.000} * LAG \tag{4.2}$$

A divisão entre *SR* e 1.000 nos dá a quantidade de *samples* a cada milissegundo de áudio. Finalmente, a multiplicação desse valor por *LAG* nos dá a quantidade de *samples* por cada unidade de latência apresentada pela Internet. Esse cálculo, portanto, nos dá o total de valores que o modelo precisará prever para compensar pelo tempo de transporte dos pacotes. Portanto, para o valor de latência de 50 ms, $LB = 2.205$ e; para 100 ms, $LB = 4.410$.

Para o treinamento do modelo *Sequential*, precisamos definir, também, um valor E de iterações rodadas no treinamento, denominadas *epochs*. Definimos $E = 3$, pois, em nossos experimentos, percebemos que a função de perda calculada pelo modelo não apresentava mudanças significativas depois de três iterações.

Apesar de não ser uma das métricas de sucesso definidas na Seção 3.2, o tempo de treinamento também foi registrado em nossos experimentos. Idealmente, queremos evitar tempos muito longos visando reduzir o período de pré-processamento e permitindo treinamentos em máquinas menos potentes.

Definidos os valores de X, Y, *LB* e E, podemos, então treinar nossa rede neural. Uma vez treinado, podemos usá-la para realizar as predições.

4.1.3 Processo de previsão

Nos experimentos de nossas predições, precisamos organizar nossos dados de teste. Escolhemos a terceira repetição da introdução tocada em *Hotel California* pois, em análise manual, soou semelhante à primeira reprodução, porém, com algumas pequenas improvisações. Por exemplo, em algumas transições entre acordes, é tocado uma nota intermediária, o que não foi observado na primeira repetição da introdução. Essa característica é ideal para uma simulação de um ambiente musical *online* colaborativo - a mesma sequência de acordes foi tocada, porém, com leves variações da forma que é executada.

Uma vez treinado, o modelo requer uma lista Z, no mesmo formato que a lista X, definida na Subseção 4.1.2 - uma conjunto de subconjuntos de tamanho *LB*. Além disso, precisaremos realizar a normalização inversa da previsão para retornar os valores ao intervalo original de $[-32.768, +32.767]$ para, de fato, podermos representar as previsões em arquivos WAV.

Portanto, para rodar as previsões com o arquivo completo da introdução de *Hotel California*, precisamos dividir tal arquivo durações de *LAG* ms e, para cada uma, convertê-las no formato requerido da lista Z.

Como uma das métricas de sucesso, definidas na Seção 3.2, é o tempo necessário para gerar as previsões, medimos o tempo para formatar cada arquivo de duração *LAG* ms e o de geração das previsões, para cada lista Z.

Ao final de todas as previsões, convertemos todas as sequências de sinais digitais para um

arquivo WAV para análise manual dos áudios produzidos [1].

4.2 Avaliação

Para analisar a efetividade desse modelo, vamos analisar os resultados dos experimentos sob a ótica os critérios de avaliação, definidos na Seção 3.2.

4.2.1 Corretude das previsões

Para cada um dos valores de *LAG*, definidos na Seção 4.1, todas as previsões realizadas tenderam a "repetir" as sequências de áudio de entrada. Comparando os sinais digitais do arquivo de teste e os gerados na previsão, ilustrados na Figura 4.2, é possível notar que, quando alinhado com a sequência real de teste, não há nenhuma correspondência. No entanto, ao compararmos com os dados de entrada, isto é, o que construímos a lista Z, é possível notar uma clara semelhança. O formato da onda sonora foi mantido, diferindo apenas por sua amplitude.

Devido à natureza de identificação ciclos que o LSTM possui para realizar usas previsões, acredita-se que, pela natureza caótica dos sinais digitais representando áudio natural, não foi possível encontrar nenhum padrão que se repita para poder realizar as previsões. A tendência, portanto, é replicar os dados de entrada, como pode-se observar na Figura 4.2.

Ademais, os áudios previstos soaram distorcidos quando comparados às sequências de entrada. Como o áudio é gerado pela rede neural, sem nenhuma instrução ou base para geração de áudios de boa qualidade, o resultado foi esperado. A Figura 4.2 demonstra que, apesar da previsão seguir o mesmo formato das ondas da sequência de entrada, o mesmo não ocorre com sua amplitude, sugerindo que há perda de informações e causando, consequentemente, a sensação de distorção.

Esses resultados foram observados em todas as janelas de entrada, tanto para os valores de *LAG* 50 ms e 100 ms.

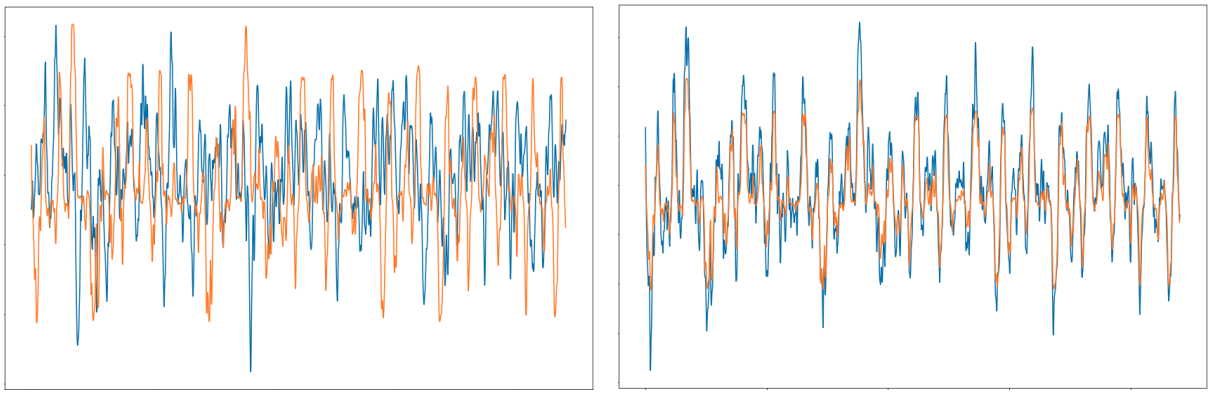
Caso aplicássemos esse modelo em uma aplicação real, efetivamente estaríamos repetindo os dados transmitidos e, como discutido no Capítulo 3, retornaremos ao problema que as soluções *delay-based* enfrentam. Portanto, não podemos atestar a corretude das previsões para o modelo preditivo gerador de novas sequências.

4.2.2 Tempo de geração de previsões

Na Tabela 4.1, podemos observar as médias de tempo para gerar as previsões para cada valor de *LAG* testado, assim com o tempo médio de treinamento. É notável que, para nenhum dos dois valores, o tempo de previsão foi menor que o tempo da janela de previsão. Dessa forma, podemos afirmar que este modelo preditivo, como foi implementado, não satisfaz a métrica de sucesso para o tempo de previsão.

Ademais, vale notar que, apesar de não ser uma métrica de sucesso, que o tempo médio de treinamento foi relativamente alto, além de depender do valor de *LAG*. Em uma aplicação

[1] O áudio gerado pode ser encontrado em `https://cutt.ly/cvNkBna`.



(a) Previsão alinhada com a sequência real de teste. (b) Previsão alinhada com a sequência de entrada.

Figura 4.2: Comparação entre as sequências digitais geradas na previsão de uma das listas Z, em laranja, com (a) a sequência real de teste e (b) a sequência de entrada, ambas em azul, para o valor $LAG = 50ms$.

Valor de LAG	Média de tempo de previsão	Média de tempo de treinamento por epoch
50 ms	150 ms	55 s
100 ms	380 ms	127 s

Tabela 4.1: Tabela comparando os tempos médios para gerar as previsões e para treinar os modelos para os valores 50 ms e 100 ms de simulação de latência da Internet.

real, onde a base de treinamento teria um alto volume de dados, é possível que o tempo de aprendizagem fosse alto o suficiente para não ser possível rodar em uma máquina comum, com poucos recursos disponíveis.

4.2.3 Considerações

Em suma, por não satisfazer os dois critérios de avaliação, não há indícios que o modelo preditivo gerador com LSTM deve ser utilizado em uma adaptação do *client-side prediction* para ambientes musicais. Os resultados indicam que outros métodos devem ser utilizados, de forma a aumentar a precisão das previsões e deixá-las mais eficientes.

CAPÍTULO 5

Ciclo 2: Indexação e identificação de sequências anteriores

Para o segundo ciclo de estudos, realizamos experimentos para o modelo preditivo baseado em indexação e identificação, como descrito na Subseção 3.4.2. Neste Capítulo, demonstraremos a metodologia aplicada na realização dos experimentos, assim como os resultados e as conclusões inferidas a partir deles, de acordo com as métricas de sucesso descritas na Seção 3.2.

5.1 Metodologia dos experimentos

A metodologia utilizada neste modelo pode ser dividida em duas partes: (1) o processo da indexação do banco de dados de referência e; (2) como funciona o processo de identificação das janelas do banco de dados e apontamento das previsões.

5.1.1 Indexação do banco de dados de referência

Como mencionado na Subseção 3.4.2, o banco de dados de referência para esse modelo respeita um conjunto de três regras. Portanto, para os nossos experimentos, precisamos organizar as sequências de referência respeitando-as.

Uma forma de visualizar como os dados podem ser estruturados é transformando as sessões de uma música em uma espécie de máquina de estados, como a ilustrada na Figura 5.1 para a música *Message In A Bottle* para a trilha da guitarra elétrica de base. Como exemplo, em um corte da partitura da música, demonstrada na Figura 5.2, é possível visualizar como esses estados estão organizados dentro dos compassos.

No entanto, é notável que alguns estados podem transitar para mais de um estado - por exemplo, a introdução pode transitar entre si mesma em *loop* ou para o *bridge*. Tal característica quebra a Regra 3, pois, caso o estado identificado fosse a introdução, não seria possível saber qual seria o próximo estado e duas previsões seriam entregues, causando ambiguidade.

Para lidar com isso, podemos reorganizar tal máquina apresentando estados de transição. Dessa forma, quando uma transição fosse identificada, apenas uma previsão seria gerada, como ilustrado na Figura 5.3. Note, entretanto, que nenhum dos estados "aponta" para um estado de transição. Isso significa os cortes de áudio que representam essas transições nunca será apontado como uma previsão, efetivamente perdendo essa informação na transmissão do músico remoto.

Ao dividir as janelas de referência, portanto, é necessário escolher cortes onde seja possível identificar as transições. Se usarmos a divisão de compassos que a partitura original oferece,

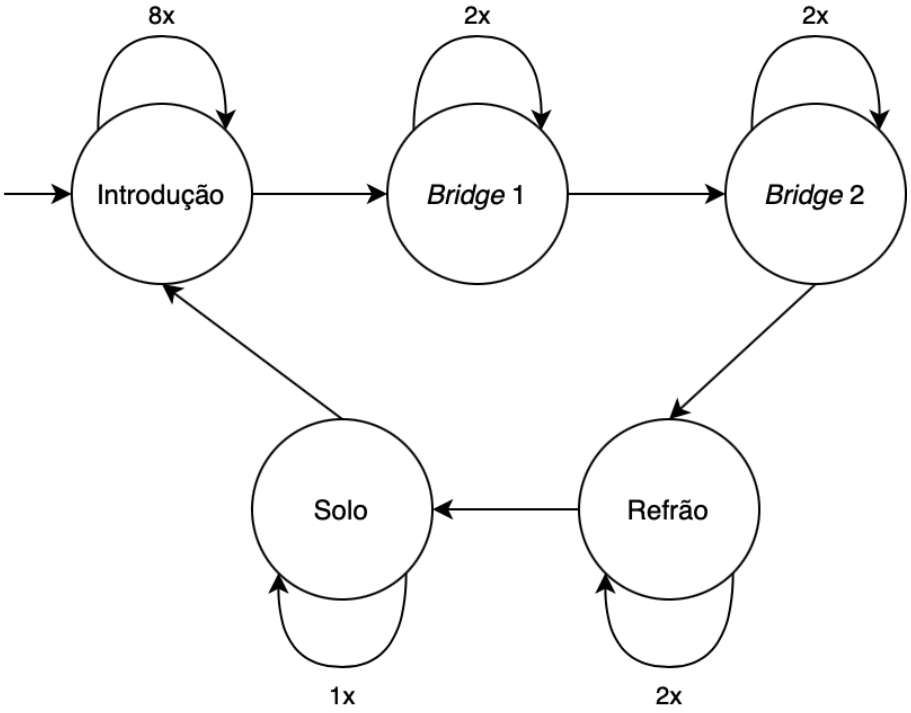


Figura 5.1: Máquina de estados para a trilha da guitarra elétrica de base da música *Message In A Bottle* (*The Police*, 1979).

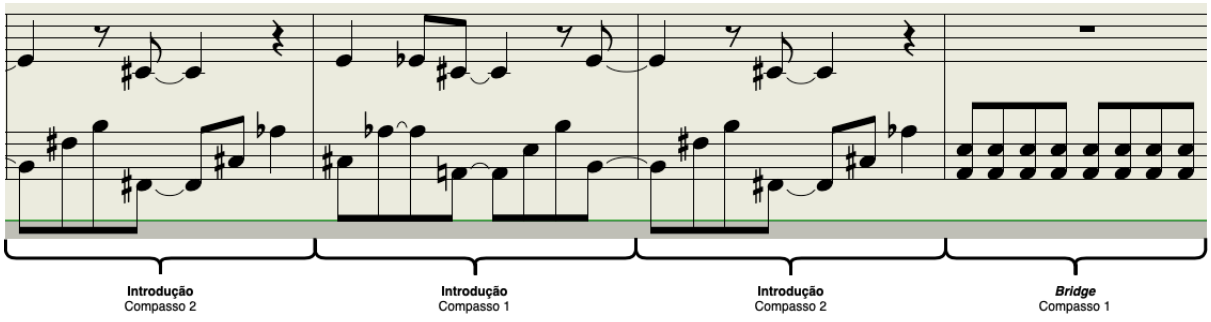


Figura 5.2: Relação dos estados da Figura 5.1 com compassos em um trecho da partitura da trilha da guitarra elétrica da música *Message In A Bottle* (*The Police*, 1979).

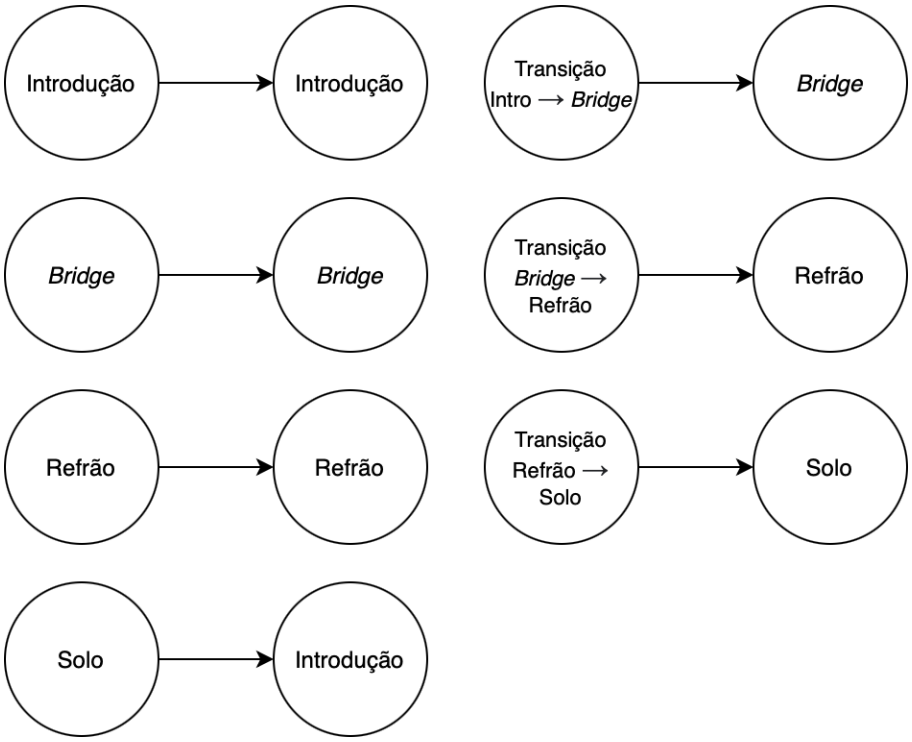


Figura 5.3: Máquina de estados adaptada para a trilha da guitarra elétrica da música *Message In A Bottle* (*The Police*, 1979).

não seria possível identificar quando termina um *loop* e quando a próxima sessão inicia. Portanto, cada sequência da música necessita possuir um pequeno corte à sua frente. A forma que fazemos isso é movendo cada janela em meio compasso para frente no tempo, como ilustrado na Figura 5.4 - dessa forma, dois compassos estarão presentes em uma mesma janela e, portanto, é possível identificar transições.

Em nossos experimentos, identificamos e separamos cada estado manualmente, assim como construímos a máquina de estados de referência, semelhante à apresentada na Figura 5.3. Para a música *Message In A Bottle*, utilizamos duas unidades de divisão para experimentação: uma onde cada janela de referência possuía dois compassos de duração (3,158 segundos) e outra onde possuíam um compasso de duração (1,579 segundos).

Também realizamos testes com a introdução da música *Hotel California*, que, diferentemente de *Message In A Bottle*, não possui *loops*. Com isto, não havia necessidade da identificação de janelas de transição, tornando a máquina de estados linear. Para ests música, experimentamos durações de 1, 2 e 3 segundos para cada janela de referência.

5.1.2 Processo de identificação das janelas e apontamento das previsões

Uma vez que possuímos o banco de dados de referência organizado, podemos realizar a identificação para as sequências de entrada, transmitidas pelo músico remoto. Como mencionado na Subseção 3.4.2, utilizamos o algoritmo de alinhamento de séries temporais, o DTW (*Dynamic*

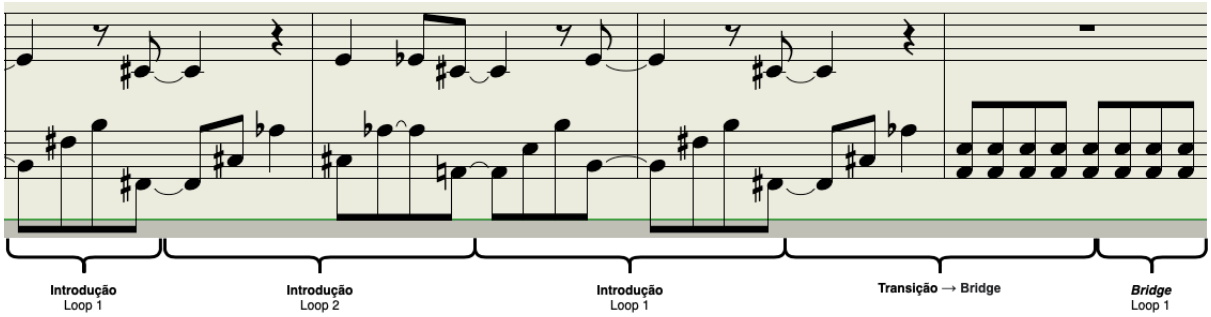


Figura 5.4: Relação dos estados da Figura 5.3 com um corte da partitura original da trilha da guitarra elétrica da música *Message In A Bottle* (*The Police*, 1979). Movendo as janelas meio compasso à frente, é possível criar janelas de transição.

Time Warping) [15], especificamente a implementação da biblioteca Librosa [16], em Python.

A função do DTW pode receber diversos argumentos. Para nossos experimentos, utilizamos três deles: (1) uma matriz de *features* da sequência de referência; (2) uma matriz de *features* da sequência de pesquisa e; (3) o booleano *subseq*, que indica se a sequência de pesquisa está contida como subsequência na primeira sequência - em nosso caso, enviamos esse argumento como *True*.

Note que os dois primeiros argumentos requerem matrizes de *features* das sequências de áudio. Para a primeira matriz, juntamos todos os arquivos de áudio da base de dados de referência em um único, e retiramos seu chromagrama [42], também implementado pela Librosa, que retorna uma tabela das notas musicais mais presentes na sequência em função do tempo. Para a segunda matriz, utilizamos a mesma função para a sequência de pesquisa, simulando o áudio transmitido pelo músico remoto.

Já que configuramos como "verdadeiro" o valor o booleano *subseq*, o algoritmo já nos dirá que corte do banco de dados de referência a nossa janela de pesquisa é mais semelhante. Observe, por exemplo, a Figura 5.5 para a música *Hotel California* - ao separar em janelas de 3 segundos, o DTW nos entregará os pontos, em segundos, sob a qual essa sequência de pesquisa se encaixa na base de referência. Com esses valores, é possível identificar qual janela de referência foi identificada - na imagem, foi a #1. No caso da música *Hotel California*, como sua máquina de estados é linear na introdução, a previsão seria a janela #2.

Para calcular a taxa de sucesso das identificações, montamos um arquivo de referência que diz, para cada sequência de entrada, qual deve ser a janela correta na base de dados - caso haja correspondência, consideramos a identificação como correta. Para as previsões, utilizamos o mesmo conceito, porém, subtraindo as instâncias onde janelas de transição foram identificadas, já que assumimos a impossibilidade de prevê-las.

Para a medição de performance, consideramos o tempo levado para identificar a janela com DTW e de reproduzir a previsão.

Uma vez que todas as janelas passaram pelo processo completo, juntamos todas as sequências de previsões apontadas em um arquivo WAV para análise [1].

[1] As previsões geradas podem ser encontradas em `https://cutt.ly/2vNvRfy`.

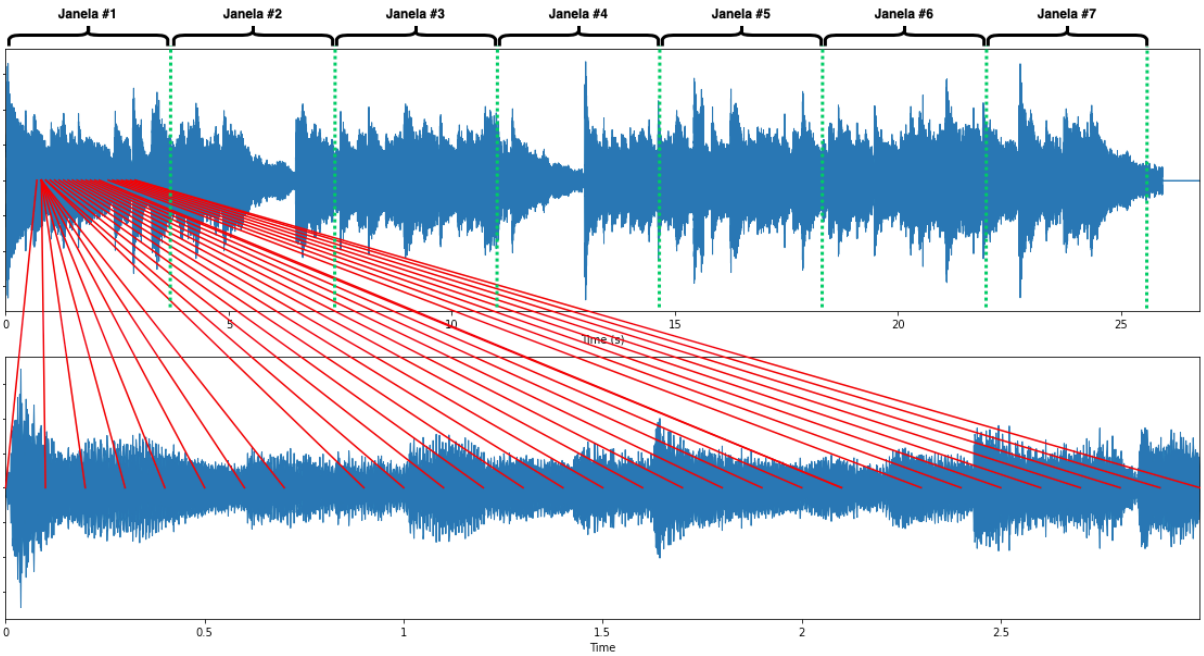


Figura 5.5: Execução do algoritmo DTW para a primeira janela de três segundos para a música Hotel California. Acima, a base de dados de referência; abaixo, a sequência de pesquisa, transmitida pelo músico remoto em nossa simulação; em vermelho, as linhas representam o alinhamento da série temporal entregues pelo algoritmo.

5.2 Avaliação

De acordo com as métricas estabelecidas na Seção 3.2, vamos analisar cada uma delas de acordo com os resultados dos nossos experimentos.

5.2.1 Corretude das previsões

Para esta métrica, devemos analisar dois aspectos:

1. A taxa de acerto das identificações das janelas;
2. A corretude sonora dos apontamentos das previsões.

É importante mencionar que o sucesso do Item 1 não indica, necessariamente, uma boa performance para este modelo preditivo. Por exemplo, para janelas de tempo grandes, há mais chances de identificar uma janela, porém, mais suposições serão realizadas na previsão. Uma improvisação musical, por exemplo, poderia ter uma janela identificada por sua sequência de acordes, porém, muita informação seria perdida.

Portanto, vejamos na Tabela 5.1, as taxas de acertos para cada música e cada duração escolhida para as janelas de identificação.

Notamos que, quanto maior a janela de duração, maior é a taxa de identificação. Isso corrobora nossa hipótese que, quanto maior a janela, mais informações possuímos para identificar

Música	Duração da janela (s)	Taxa de acerto das identificações	Taxa de acerto das previsões
Hotel California	1,0	61,9%	61,9%
	2,0	70%	70%
	3,0	100%	100 %
Message In A Bottle	3,158	100%	76,6%
	1,579	88,8%	81,4 %

Tabela 5.1: Taxas de acerto das identificações e previsões para cada música, para cada tempo de janela experimentado.

com sucesso uma sequência de referência em nossa base. Para a música *Hotel California*, já que não existe nenhum estado de transição, a taxa de acerto de identificação será igual a taxa de acerto das previsões.

Entretanto, na música *Message In A Bottle*, o mesmo não ocorre. Sempre que o músico remoto tocar uma transição, haverá um erro na previsão, pois nenhum estado aponta para transições. Porém, vale mencionar que esse erro será corrigido assim que possível - precisamente, a correção tem a duração de meio compasso, como determinado como *offset* para cada janela. É possível fazer uma comparação desse fenômeno à sensação de "teletransporte", mencionada na Seção 2.3, na implementação original de *client-side prediction* para jogos.

Quanto aos resultados, podemos considerar que foram bastante positivos. Vale mencionar que, dada nossa limitação de que a base de dados deve possuir janelas de referência similares às que serão testadas, não simulamos um ambiente real com total fidelidade, uma vez que, para que a aplicação seja generalista, o banco de dados necessita ter um grande volume de dados.

Além disso, o valor da duração das janelas utilizado nas experimentações em *Message In A Bottle* estão ligadas à duração dos compassos que, por sua vez, assumem um BPM constante durante a execução da música. Em um ambiente real, idealmente, não devemos restringir os músicos a tocarem em um BPM único.

5.2.2 Tempo de geração de previsões

Vejamos, na Tabela 5.2, as médias de tempo que este modelo levou para identificar as janelas e apontar previsões.

É notável que houve uma excelente performance nessa métrica. Para ser bem sucedida, os tempos precisam ser inferiores às durações das janelas e, para cada duração experimentada, esse objetivo foi alcançado com bastante tempo de sobra.

Vale notar que a base de dados experimentada continha apenas janelas semelhantes às que seriam testadas, deixando seu volume bastante inferior ao que uma aplicação real necessitaria.

Música	Duração da janela (s)	Média dos tempos de previsão (ms)
Hotel California	1,0	60
	2,0	110
	3,0	150
Message In A Bottle	3,158	190
	1,579	110

Tabela 5.2: Médias dos tempos necessários para apontar as previsões utilizando DTW para cada música e para cada janela de tempo experimentada.

Portanto, em grande escala, é possível que otimizações sejam necessárias para manter esse tempo aceitável, como indexar os dados pela nota musical principal, instrumento tocado, etc.

5.2.3 Considerações

Os resultados apresentados pelo modelo indexador com DTW foram bastante promissores. A taxa de identificação, para valores de janelas acima de 3 segundos, foi de 100% em ambas as músicas testadas. Para a música *Message In A Bottle*, por utilizarmos um tempo de janela com duração relacionada à música - a duração dos compassos - a taxa de identificação continuou relativamente alta para as duas durações de janelas escolhidas.

Além disso, por ter sido performática, há evidências de que é possível escalar essa solução para bases maiores, tornando-a mais generalista.

No entanto, revelou-se que, quando menor a duração da janela, menor foi a taxa de identificação e, portanto, a de previsão. Portanto, para músicos que realizarão improvisos, como *jam sessions*, essa aplicação não deve ser ideal.

CAPÍTULO 6

Conclusões

De acordo com nosso experimentos, o modelo preditivo gerador de novas sequências com LSTM não teve bom desempenho em nenhuma das duas métricas de sucesso estabelecidas na Seção 3.2. Por repetir os dados de entrada, caso utilizássemos esse modelo em uma aplicação real, estaríamos apenas atrasando o áudio transmitido e replicando o problemas que as soluções síncronas enfrentam hoje.

Além disso, devido ao alto tempo necessário para realizar a predição, tal solução mostrou-se inviável da forma que foi implementada. Para atingir tempos menores, seria necessário mais poder computacional, tornando essa solução inacessível para a população em geral.

Portanto, de acordo com nossos experimentos, o modelo gerador com LSTM não se mostrou promissor para ser utilizado em uma adaptação do *client-side prediction* para ambientes musicais. Porém, melhorias podem ser estudadas que viabilizem seu uso, exploradas na Seção 6.1.

Entretanto, no segundo ciclo, o modelo indexador e identificador mostrou-se bastante promissor em nossos experimentos. Apesar de possuir uma base da dados pequena, a taxa de acerto das previsões foi consideravelmente alta, de forma a trazer a sensação de fluidez nos momentos de acerto na previsão. Se relacionarmos com o *client-side prediction*, que corrige os *inputs* dos jogadores sempre que há um erro, os erros nas previsões realizadas podem ser aceitáveis.

Ademais, o tempo necessário para apontar as predições foi bastante baixo, deixando espaço suficiente para que máquinas menos potentes do que a utilizada nos experimentos possam usufruir dessa abordagem.

Porém, não podemos afirmar com certeza suficiente que o modelo indexador pode ser utilizado em aplicações reais. A base de dados utilizada foi propositalmente reduzida para testar a validade do modelo, possibilitando altas taxas de acertos na identificação e alta eficiência. Além disso, as janelas de áudio escolhidas são consideravelmente altas, o que impede seu uso em improvisações musicais.

Entretanto, o modelo indexador com DTW mostrou-se bastante promissor para uso na adaptação do *client-side prediction* para ambientes musicais, principalmente para músicos de base, que não tendem a improvisar em suas performances.

6.1 Trabalhos futuros

Para o modelo gerador de novas sequências, trazemos nesta Seção métodos que podem aumentar a corretude das previsões. Já para o modelo indexador, exemplificamos como realizar novos

experimentos que melhor simulem um ambiente real colaborativo, de forma a possuir mais propriedades para afirmar sua viabilidade. Tais métodos podem ser utilizados em trabalhos futuros, visando melhorar os resultados apresentados.

6.1.1 Modelo gerador

Além do LSTM, outros métodos de previsões de sequência podem ser utilizados em trabalhos futuros. Por exemplo, o *seq2seq*, um *framework* codificador/decodificador de propósito geral para o *Tensorflow* [43], é utilizado em aplicações que requerem traduções de uma sequências para outras de diferentes domínios. É bastante utilizado em traduções para diferentes idiomas, legendas automáticas de imagens, sumário de textos, entre outros. Sua aplicação em *Time Series Forecasting* considera dois domínios: passado e futuro. Dessa forma, o conjunto de dados constituiria de pares de sequências - o primeiro elemento representando a sequência de áudio tocado e o segundo o que foi reproduzido logo em seguida.

Na Subseção 3.4.1, mencionamos a possibilidade de uso de ferramentas de continuação de música baseada em um estilo como modelo preditivo. Tal abordagem, como trabalhos futuros, pode ser experimentada de outras maneiras. Pachet propõe o instrumento musical *The Continuator*, que é capaz de aprender com as últimas sequências do músico e reproduzir uma continuação [44]. Por ser interativo, as continuações são geradas em tempo real, sendo um candidato em potencial para nossa adaptação, dado o requisito de boa eficiência para geração de predições.

No entanto, ainda utilizando o método proposto com LSTM, é possível melhorar os dados de entrada. Em nossos experimentos, utilizamos os sinais digitais diretamente como entrada para aprendizagem e predição do modelo. Tal abordagem pode não ter sido a ideal, uma vez que é difícil extrair informações desses valores em sua forma básica. Além disso, a dimensionalidade e o volume desses dados é bastante grande, dificultando a aprendizagem.

Recomenda-se, portanto, que haja uma extração de informações antes de usar os dados para aprendizagem nas redes neurais. Por exemplo, extração de *features* como o espectrograma e informações de BPM podem ser utilizadas para concentrar as informações dos dados para aprendizagem.

Por exemplo, o Jukebox [37], mencionado na Subseção 3.4.1, enfrenta problemas semelhantes de dimensionalidade. Sua implementação utiliza a rica técnica de compressão VQ-VAE [45] para reduzir a dimensão dos dados de áudio, facilitando seu processamento. Portanto, seu uso é bastante pertinente em uma adaptação do *client-side prediction*.

6.1.2 Modelo indexador

Há duas principais melhorias para esse modelo: (1) aumentar a base de dados para torná-la mais generalista e; (2) reduzir a janela de áudio de identificação.

Em nosso experimentos, como a base de dados de referência foi reduzida, foi viável realizar a separação das janelas manualmente. No entanto, em uma aplicação real, tal tarefa teria que ser automatizada. Portanto, para trabalhos futuros, sugerimos que, para atingir o primeiro objetivo, encontre-se métodos que identifiquem sessões repetidas na música e automaticamente as indexe no conjunto de dados de referência.

Além disso, a medida em que a base de referência cresce, mais tempo o DTW levará para identificar as subsequências, dado que sua complexidade está na ordem de $O(N^2)$. Dessa forma, é importante que a base seja pré-processada, de forma a reduzir quantidade de dados em cada divisão, como uma espécie de *hash table*. Portanto, para trabalhos futuros, sugere-se pesquisas para encontrar técnicas para realizar esse pré-processamento - por exemplo, pode-se indexar essas divisões de acordo com a nota principal daquela janela de áudio.

Como continuação deste trabalho, esperamos implementar esse conjunto de técnicas, com a expectativa de melhoria nos resultados, de forma a possuir dados conclusivos sobre a validade, ou não, a adaptação proposta do *client-side prediction* para ambientes musicais colaborativos *online*.

Referências Bibliográficas

[1] J. Albano, "Monitoring latency (how low can you go?) : Ask.audio." https://ask.audio/articles/monitoring-latency-how-low-can-you-go, 04 2017. (Acessado em 29/03/2021).

[2] "Troubleshooting and debugging voip call basics - cisco." https://www.cisco.com/c/en/us/support/docs/voice/h323/14081-voip-debugcalls.html. (Acessado em em 04/02/2021).

[3] A. McPpherson, R. Jack, and G. Moro, "Action-sound latency: Are our tools fast enough?," in *Proceedings of the International Conference on New Interfaces for Musical Expression*, (Brisbane, Australia), Queensland Conservatorium Griffith University, 2016.

[4] "Lola." https://lola.conts.it/. (Acessado em 04/02/2021).

[5] I. Howell, "Soundjack: Comprehensive user guide." https://www.ianhowellcountertenor.com/soundjack-comprehensive-user-guide. (Acessado em 04/02/2021).

[6] "Jamkazam | live, in-sync music jamming over the internet." https://jamkazam.com/#jamtracks. (Acessado em 04/02/2021).

[7] "How it works | jammr." https://jammr.net/howitworks.html. (Acessado em 04/02/2021).

[8] "Jamtaba - a free app to play live music in online jam sessions." https://jamtaba-music-web-site.appspot.com/. (Acessado em 21/04/2021).

[9] "How fast is realtime? human perception & technology | pubnub." https://www.pubnub.com/blog/how-fast-is-realtime-human-perception-and-technology/. (Acessado em 04/02/2021).

[10] Y. Bernier, "Latency compensating methods in client/server in-game protocol design and optimization," 2003.

[11] I. Khan, "Competitive smash bros. is insanely fast and this video proves it | dot esports." https://dotesports.com/general/news/smash-bros-westballz-inputs-per-second-2658, 11 2015. (Acessado em 11/04/2021).

[12] P. Dhariwal, H. Jun, C. Payne, J. W. Kim, A. Radford, and I. Sutskever, "Jukebox: A generative model for music," *ArXiv*, vol. abs/2005.00341, 2020.

[13] F. Chollet, "keras." https://github.com/fchollet/keras, 2015.

[14] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, p. 1735–1780, Nov. 1997.

[15] M. Müller, *Fundamentals of Music Processing: Audio, Analysis, Algorithms, Applications*. Springer Publishing Company, Incorporated, 1st ed., 2015.

[16] B. McFee, C. Raffel, D. Liang, D. P. Ellis, M. McVicar, E. Battenberg, and O. Nieto, "librosa: Audio and music signal analysis in python," in *Proceedings of the 14th python in science conference*, vol. 8, 2015.

[17] C. Burton-Hill, "What does a conductor actually do? - bbc culture." https://www.bbc.com/culture/article/20141029-what-do-conductors-actually-do, 10 2014. (Acessado em 11/04/2021).

[18] D. Wessel and M. Wright, "Problems and prospects for intimate musical control of computers," *arXiv: Human-Computer Interaction*, 2020.

[19] A. Carôt, "Low latency audio streaming for internet-based musical interaction," in *Streaming Media Architectures, Techniques, and Applications: Recent Advances*, 2011.

[20] D. Bies and C. Hansen, *Engineering Noise Control: Theory and Practice, Fourth Edition*. Taylor & Francis, 2009.

[21] R. Penrose, *The Road to Reality: A Complete Guide to the Laws of the Universe*, p. 410. Vintage Series, Vintage Books, 2007.

[22] C. Drioli, C. Allocchio, and N. Buso, "Networked performances and natural interaction via lola: Low latency high quality a/v streaming system," in *ECLAP*, 2013.

[23] W. Wakka, "Internet acima de 1 gbps chega a 5% da população mundial - canaltech." https://canaltech.com.br/telecom/internet-acima-de-1-gbps-chega-a-5-da-populacao-mundial-156157/, 11 2019. (Acessado em 04/04/2021).

[24] "Speedtest global index – monthly comparisons of internet speeds from around the world." https://www.speedtest.net/global-index, 02 2021. (Acessado em 04/04/2021).

[25] "(12) jamkazam overview - youtube." https://www.youtube.com/watch?v=ylYcvTY9CVo&ab_channel=JamKazam, 03 2014. (Acessado em 04/04/2021).

[26] D. Wilson, "What is latency & why does it matter? : Jamkazam." https://jamkazam.freshdesk.com/support/solutions/articles/66000122532, 09 2020. (Acessado en 04/04/2021).

[27] S. O. Chris Chafe, "Jacktrip on raspberry pi," in *17th Linux Audio Conference*, vol. 8, 2019.

[28] "Sonobus." https://sonobus.net/. (Acessado em 21/04/2021).

[29] "Cockos incorporated | ninjam." https://www.cockos.com/ninjam/. (Acessado em 21/04/2021).

[30] Infilament, "Netcode," *Fightin' Words*, Outubro 2019.

[31] "The web back in 1996-1997," *Pingdom*, Setembro 2008.

[32] I. Dransfield, "The engine room: Build," *Retro Gamer*, Junho 2018.

[33] S. Negron, "Rollback netcode is the superior fighting game experience - here's why," *CBR.com*, Agosto 2020.

[34] R. Edwards, "Mmo combat, the fight against lag! - digressing and sidequesting." https://youtu.be/mR00P5x8_WQ, Janeiro 2016. (Acessado em 18/04/2021).

[35] U. Hoffmann, "Audio – basic technical information." https://www.lehman.edu/faculty/hoffmann/itc/techteach/audio/audiotechinfo.html, Abril 2005. (Acessado em 20/04/2021).

[36] "Colab pro faq." https://colab.research.google.com/signup#. (Acessado em 20/04/2021).

[37] P. Dhariwal, H. Jun, C. Payne, J. W. Kim, A. Radford, and I. Sutskever, "Jukebox: A generative model for music," *ArXiv*, vol. abs/2005.00341, 2020.

[38] J. Brownlee, "Sequence to sequence learning with neural networks," Dezembro 2016.

[39] A. Klapuri and M. Davy, *Signal Processing Methods for Music Transcription*. Springer Science & Business Media, 2006.

[40] V. Schmidt, "Lstm training is really slow · issue #1063." https://github.com/keras-team/keras/issues/1063, Novembro 2015. (Acessado em 19/04/2021).

[41] O. Gold and M. Sharir, "Dynamic time warping and geometric edit distance," *ACM Transactions on Algorithms (TALG)*, vol. 14, pp. 1 – 17, 2018.

[42] D. P. Ellis, "Chroma feature analysis and synthesis." http://labrosa.ee.columbia.edu/matlab/chroma-ansyn/, Abril 2007. (Acessado em 21/04/2021).

[43] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," *CoRR*, vol. abs/1409.3215, 2014.

[44] F. Pachet, "The continuator: Musical interaction with style," *Journal of New Music Research*, vol. 32, pp. 333–341, 08 2010.

[45] A. Razavi, A. van den Oord, B. Poole, and O. Vinyals, "Preventing posterior collapse with delta-vaes," *CoRR*, vol. abs/1901.03416, 2019.