Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Tradução Inteligente da Soletração em Libras com Redes Neurais Recorrentes

Breno C. Avelino Lima
Pedro R. Diógenes Macedo

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Ciência da Computação

Orientador
Prof. Dr. Marcus Vinicius Lamar

Brasília
2025

Universidade de Brasília — UnB
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Bacharelado em Ciência da Computação

Coordenador: Prof. Dr. Marcelo Grandi Mandelli

Banca examinadora composta por:

Prof. Dr. Marcus Vinicius Lamar (Orientador) — CIC/UnB
Prof.ª Dr.ª Carla Cavalcante Koike — CIC/UnB
Prof. Dr. Li Weigang — CIC/UnB

CIP — Catalogação Internacional na Publicação

Lima, Breno C. Avelino.

Tradução Inteligente da Soletração em Libras com Redes Neurais Recorrentes / Breno C. Avelino Lima, Pedro R. Diógenes Macedo. Brasília : UnB, 2025.
71 p. : il. ; 29,5 cm.

Monografia (Graduação) — Universidade de Brasília, Brasília, 2025.

1. Tecnologia Assistiva, 2. Redes Neurais Recorrentes, 3. LIBRAS

CDU 004

Endereço: Universidade de Brasília
Campus Universitário Darcy Ribeiro — Asa Norte
CEP 70910-900
Brasília–DF — Brasil

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Tradução Inteligente da Soletração em Libras com Redes Neurais Recorrentes

Breno C. Avelino Lima
Pedro R. Diógenes Macedo

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Ciência da Computação

Prof. Dr. Marcus Vinicius Lamar (Orientador)
CIC/UnB

Prof.ª Dr.ª Carla Cavalcante Koike
CIC/UnB

Prof. Dr. Li Weigang
CIC/UnB

Prof. Dr. Marcelo Grandi Mandelli
Coordenador do Bacharelado em Ciência da Computação

Brasília, 25 de julho de 2025

Agradecimentos

Agradeço, em primeiro lugar, à minha mãe, Adriana, e aos meus avós maternos, Francisco e Lenira, que me criaram e estiveram ao meu lado em todos os momentos da vida. Foram eles que sempre me apoiaram incondicionalmente, mesmo nas fases mais difíceis, e que torceram pelo meu sucesso em cada etapa da minha trajetória acadêmica e pessoal. Sou grato também à minha tia Maria Laís, que me acolheu e me auxiliou durante o período em que vivi em Brasília.

Agradeço igualmente aos amigos que fiz ao longo do curso, que tornaram essa caminhada mais leve e enriquecedora. Compartilhamos desafios, conquistas e aprendizados, sempre nos apoiando mutuamente. Um agradecimento especial vai para o Pedro, com quem dividi grande parte dessa jornada acadêmica. Estivemos juntos em diversas disciplinas, frequentemente trabalhando lado a lado nos mesmos grupos, e foi com ele que desenvolvi este trabalho.

Por fim, expresso minha gratidão ao meu orientador, professor Lamar, que sempre se mostrou disponível e comprometido em nos orientar. Seu acompanhamento atento e suas contribuições foram fundamentais para o desenvolvimento deste trabalho ao longo deste último ano.

Breno C. Avelino Lima

Em primeiro lugar, agradeço aos meus pais, que sempre me deram suporte e incentivo para alcançar meus objetivos e sonhos, dedicando suas vidas a mim. Agradeço também à minha noiva, Beatriz, que esteve ao meu lado durante todo o tempo em que estive na UnB, oferecendo seu apoio e sua companhia nos altos e baixos desse período.

Sou grato, ainda, a todos os amigos que fiz ao longo dos semestres, em especial ao Breno, com quem compartilhei diversas disciplinas, trabalhos em grupo e, agora, este trabalho de conclusão de curso.

Por fim, agradeço ao professor Lamar, que nos orientou na realização deste trabalho, sempre presente semana após semana, disposto a nos ajudar e a oferecer a melhor orientação possível.

Pedro R. Diógenes Macedo

Resumo

Este trabalho propõe um sistema experimental de tradução automática da soletração em Língua Brasileira de Sinais (LIBRAS) para o português escrito. A abordagem utiliza redes neurais recorrentes do tipo Elman, integradas a bibliotecas de visão computacional e modelos de linguagem (*Large Language Model* (LLM)). O sistema captura gestos manuais em vídeo, aplica algoritmos para detecção das articulações das mãos, realiza filtragem de ruído e corrige a saída com apoio dos LLMs, gerando a palavra mais provável. A avaliação foi realizada com vídeos de um usuário soletrando o alfabeto manual de LIBRAS. Após o treinamento do modelo neural personalizado, o sistema foi testado na soletração de um conjunto de palavras, alcançando acurácia entre 61% e 89%. Como próximos passos, propõe-se ampliar o sistema para reconhecer sinais lexicais completos, incluir expressões faciais e corporais, e evoluir para uma ferramenta mais robusta de apoio à comunicação entre surdos e ouvintes.

Palavras-chave: Tecnologia Assistiva, Redes Neurais Recorrentes, LIBRAS

Abstract

This work proposes an experimental system for the automatic translation of fingerspelling in Brazilian Sign Language (LIBRAS) into written Portuguese. The approach employs Elman-type recurrent neural networks, integrated with computer vision libraries and Large Language Models (LLMs). The system captures hand gestures in video, applies algorithms to detect hand joint positions, filters noise, and refines the output using LLMs to generate the most likely word. Evaluation was conducted using videos of a user fingerspelling the LIBRAS manual alphabet. After training a personalized neural model, the system was tested on a set of words, achieving accuracy ranging from 61% to 89%. As future work, the system aims to expand to recognize complete lexical signs, incorporate facial and body expressions, and evolve into a more robust tool to support communication between deaf and hearing individuals.

Keywords: Assistive Technology, Recurrent Neural Networks, LIBRAS

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

Adam *Adaptive Moment Estimation.*

API Interface de Programação de Aplicações (do inglês *Application Programming Interface*).

ASL *American Sign Language.*

BPTT *Backpropagation Through Time.*

BSL *British Sign Language.*

CNNs Redes Neurais Convolucionais (do inglês *Convolutional Neural Networks*).

GRU *Gated Recurrent Unit.*

GUI Interface Gráfica do Usuário (do inglês *Graphical User Interface*).

IA Inteligência Artificial.

IBGE Instituto Brasileiro de Geografia e Estatística.

IMU *Inertial Measurement Unit.*

INES Instituto Nacional de Educação de Surdos.

ISL *Indian Sign Language.*

JSL *Japanese Sign Language.*

k-NN *k-Nearest Neighbors.*

LIBRAS Língua Brasileira de Sinais.

LLM *Large Language Model.*

LSF *Langue des Signes Française.*

LSTM *Long Short-Term Memory.*

MLP Perceptron Multicamadas (do inglês *Multilayer Perceptron*).

OpenCV *Open Source Computer Vision Library.*

PNS Pesquisa Nacional de Saúde.

RBFN Redes de Função de Base Radial (do inglês *Radial Basis Function Network*).

ReLU *Rectified Linear Unit.*

RNAs Redes Neurais Artificiais.

RNN Rede Neural Recorrente (do inglês *Recurrent Neural Network*).

SAAF *Shape Autotuning Activation Function.*

SALSTM-SAAF *Self-Attention Long-Short-Term Memory with Shape Autotuning Activation Function.*

SGD Gradiente Descendente Estocástico (do inglês *Stochastic Gradient Descent*).

TS *Time Step.*

Capítulo 1

Introdução

De acordo com a Pesquisa Nacional de Saúde (PNS), divulgada em 2019 pelo Instituto Brasileiro de Geografia e Estatística (IBGE), o Brasil conta com aproximadamente 10,7 milhões de pessoas com algum grau de deficiência auditiva [1]. Dentre esse contingente, cerca de 2,3 milhões de brasileiros com 2 anos ou mais declararam ter muita dificuldade para ouvir ou não ouvir de modo algum, representando cerca de 1,1% da população total na época da pesquisa. Além disso, a PNS trouxe uma contribuição inédita ao investigar o conhecimento da Língua Brasileira de Sinais (LIBRAS) entre brasileiros, independentemente de possuírem deficiência auditiva. Os dados indicam que cerca de 4,6 milhões de pessoas com cinco anos ou mais afirmaram saber utilizá-la, o que corresponde a aproximadamente 2,2% da população brasileira.

A LIBRAS desempenha um papel fundamental para a inclusão social, pois é o principal meio de comunicação para pessoas surdas, possibilitando a expressão e compreensão em contextos onde a língua oral não é acessível. No entanto, sua difusão ainda é limitada, já que a LIBRAS não é amplamente ensinada nas escolas regulares nem disponibilizada em muitos espaços públicos, dificultando sua aprendizagem e o intercâmbio comunicativo entre surdos e ouvintes.

Nesse cenário, a crescente popularização e avanço dos treinamentos de redes neurais artificiais e de técnicas de Inteligência Artificial (IA) apresentam uma oportunidade promissora para o desenvolvimento de ferramentas capazes de traduzir e interpretar a LIBRAS de maneira automatizada. Essas tecnologias têm o potencial de facilitar não apenas a comunicação entre surdos e ouvintes, mas também o processo de aprendizado da própria língua de sinais, contribuindo para a inclusão, acessibilidade e valorização cultural da comunidade surda.

1.1 Problema

O presente trabalho tem como objetivo investigar a viabilidade de utilizar técnicas de redes neurais para a tradução automática de gestos em Língua Brasileira de Sinais a partir de vídeos. O problema central que buscamos abordar consiste no desenvolvimento de um sistema capaz de capturar, por meio de vídeo, os gestos realizados por uma pessoa e traduzi-los em texto escrito.

Neste estudo, optamos por restringir o escopo ao reconhecimento da soletração manual, ou seja, da representação do alfabeto por meio de sinais realizados exclusivamente com as mãos. Essa escolha se justifica por duas razões principais: primeiro, a soletração constitui uma subárea bem delimitada da LIBRAS, baseada em um conjunto finito de sinais manuais; segundo, ela dispensa o uso de expressões faciais, corporais e posturais, o que simplifica a coleta e o tratamento dos dados visuais, além de reduzir a complexidade do modelo de rede neural necessário.

Essa abordagem inicial, mais controlada, permite estabelecer uma base sólida para o desenvolvimento de sistemas mais complexos no futuro, capazes de reconhecer sinais completos — que envolvam não apenas as mãos, mas também o movimento do corpo e expressões faciais — e traduzir palavras ou frases inteiras.

Este trabalho teve como objetivo desenvolver um sistema capaz de traduzir palavras soletradas em LIBRAS para o português escrito, utilizando redes neurais aplicadas ao reconhecimento de gestos em vídeo. Para alcançar esse objetivo geral, uma série de etapas foi realizada: inicialmente, foram coletados vídeos de participantes executando a soletração do alfabeto manual em LIBRAS. Em seguida, os pontos de referência das mãos (*landmarks*) foram extraídos de cada quadro dos vídeos utilizando a biblioteca *MediaPipe*. Com esses dados, foram treinadas redes neurais — com ênfase na rede de Elman — para reconhecer as sequências de letras. Após o reconhecimento, foi implementado um pós-processamento para eliminar ruídos nas sequências geradas, e por fim, modelos de linguagem (LLMs) foram utilizados para sugerir a palavra mais provável com base na sequência gerada no pós-processamento.

Este trabalho, entretanto, possui limitações importantes. A principal limitação deste trabalho diz respeito à falta de generalização do sistema. Os modelos foram treinados e testados com dados das mesmas pessoas, o que significa que o sistema aprendeu padrões específicos de quem realizou os gestos, comprometendo sua aplicabilidade prática com usuários diferentes. Além disso, a coleta foi feita com um número reduzido de participantes, o que restringe a variabilidade nos dados e limita o potencial de generalização do sistema para diferentes perfis de usuários, especialmente surdos com fluência em LIBRAS. Ainda assim, os resultados obtidos demonstram o potencial da abordagem proposta e servem como ponto de partida para futuras melhorias e extensões.

1.2 Estrutura

A estrutura desta monografia está organizada da seguinte forma: o Capítulo 2 – Fundamentos apresenta a fundamentação teórica necessária para a compreensão dos conceitos e ferramentas utilizados ao longo do trabalho. O Capítulo 3 – Metodologia descreve as decisões de projeto tomadas e detalha o processo de implementação do protótipo proposto. Em seguida, o Capítulo 4 – Resultados expõe os testes realizados durante o desenvolvimento do sistema, assim como a análise e discussão dos resultados obtidos. Por fim, o Capítulo 5 – Conclusões apresenta as considerações finais sobre o trabalho realizado, sintetiza os principais achados e propõe direções para pesquisas futuras.

Capítulo 2

Revisão Teórica

Este capítulo tem como objetivo apresentar os fundamentos teóricos necessários para a compreensão do desenvolvimento do sistema de tradução da soletração em LIBRAS proposto nesta monografia. Para isso, são abordados os principais conceitos, tecnologias e pesquisas relacionadas que sustentam a construção do protótipo.

Inicialmente, será feito um panorama histórico da linguagem de sinais, com ênfase na evolução da LIBRAS no contexto brasileiro, destacando sua importância sociocultural e suas características linguísticas. Em seguida, serão discutidas as principais tecnologias e ferramentas utilizadas no projeto, como redes neurais artificiais, a arquitetura de Elman e a biblioteca *TensorFlow*, bem como os critérios que motivaram sua adoção. Por fim, será apresentada uma revisão do estado da arte, abordando estudos e abordagens recentes voltadas à tradução de línguas de sinais por meio de técnicas de Inteligência Artificial, com o intuito de contextualizar o presente trabalho frente aos avanços da área.

2.1 Histórico das Linguagens de Sinais

As linguagens de sinais são sistemas linguísticos visuais-espaciais utilizados por comunidades surdas em todo o mundo como principal meio de comunicação [2]. Diferentemente do que muitos imaginam, elas não tentam imitar as línguas orais, são línguas naturais completas, com gramáticas próprias, estrutura sintática complexa, morfologia, semântica e variações regionais. Sua origem remonta a tempos muito anteriores à formalização da educação de surdos, sendo documentada desde a Antiguidade.

Os primeiros registros históricos sobre o uso sistemático de sinais para comunicação entre pessoas surdas remontam à Grécia Antiga [3]. No entanto, foi apenas no século XVI que surgiram os primeiros esforços estruturados de ensino e formalização de linguagens de sinais, especialmente na Europa. Na Espanha, o monge beneditino Pedro Ponce de León (1520–1584) é frequentemente creditado como o primeiro educador de surdos conhecido.

Ele desenvolveu métodos de ensino que combinavam sinais com escrita e leitura para instruir jovens surdos de famílias nobres.

A partir do século XIX, as línguas de sinais começaram a ser estudadas de forma mais sistemática. Contudo, enfrentaram resistência durante o Congresso de Milão de 1880, onde foi aprovado que a educação de surdos deveria privilegiar métodos oralistas, que desencorajavam o uso da linguagem de sinais nas escolas. Essa decisão teve impacto global, inclusive no Brasil, e gerou um declínio institucional do uso das línguas de sinais por várias décadas.

Somente a partir da segunda metade do século XX, com os avanços da linguística moderna e o fortalecimento dos movimentos sociais de pessoas com deficiência, as línguas de sinais passaram a ser reconhecidas novamente como línguas naturais legítimas. Atualmente, cada país possui sua própria língua de sinais, desenvolvida de forma independente, como a *British Sign Language* (BSL), *Langue des Signes Française* (LSF), *American Sign Language* (ASL), entre muitas outras. Todas essas línguas partilham a característica de terem emergido naturalmente dentro de comunidades surdas, com dinâmicas linguísticas próprias, frequentemente influenciadas por fatores culturais e históricos locais [3].

A Língua Brasileira de Sinais (LIBRAS) é a língua natural das comunidades surdas no Brasil, tendo-se desenvolvido historicamente como um sistema linguístico visual-espacial independente da língua portuguesa [4]. Assim como outras línguas de sinais ao redor do mundo, a LIBRAS não é uma tradução direta do português oral ou escrito, mas uma língua com gramática, sintaxe, morfologia e semântica próprias, adequada às necessidades comunicativas da população surda brasileira.

A origem da LIBRAS está fortemente ligada à chegada do educador francês surdo Ernest Huet ao Brasil, em meados do século XIX. Huet, influenciado pela *Langue des Signes Française* (LSF), contribuiu para a fundação do Instituto Imperial de Surdos-Mudos no Rio de Janeiro em 1857, atual Instituto Nacional de Educação de Surdos (INES). A introdução da LSF no ambiente educacional brasileiro teve papel importante no desenvolvimento da LIBRAS, especialmente na formação de seu alfabeto manual e de parte de seu vocabulário.

Com o tempo, a língua de sinais utilizada pelas comunidades surdas brasileiras evoluiu e se distanciou da LSF, incorporando elementos próprios, regionalismos e adaptações culturais. Assim, ela consolidou-se como uma língua única e autônoma, embora tenha mantido traços de sua origem francesa [4].

O reconhecimento oficial da LIBRAS no Brasil ocorreu com a promulgação da Lei nº 10.436, de 24 de abril de 2002, que estabelece a língua de sinais como "meio legal de comunicação e expressão", assegurando seu uso em instituições públicas e privadas, especialmente na educação. Esse marco legal foi reforçado pelo Decreto nº 5.626, de

22 de dezembro de 2005, que regulamentou a lei e determinou, entre outras coisas, a obrigatoriedade do ensino da LIBRAS nos cursos de formação de professores e em cursos de fonoaudiologia, além da oferta de profissionais intérpretes em ambientes educacionais e administrativos [5].

Atualmente, a LIBRAS é amplamente reconhecida como um instrumento fundamental para a inclusão social e educacional da comunidade surda no país. Seu ensino e uso vêm se expandindo, mas ainda enfrentam desafios relacionados à formação de intérpretes, ao desconhecimento da língua pela população em geral e à carência de recursos tecnológicos acessíveis. Nesse contexto, iniciativas que buscam utilizar tecnologias digitais para apoiar a comunicação e o aprendizado em LIBRAS, como o presente trabalho, desempenham papel importante na promoção da acessibilidade e da cidadania.

2.2 Língua Brasileira de Sinais (LIBRAS)

Língua Brasileira de Sinais (LIBRAS) é uma língua visual-espacial que apresenta níveis estruturais próprios — fonológico, morfológico, sintático e semântico — e segue regras gramaticais específicas, distintas daquelas da língua portuguesa oral ou escrita. Ela não deve ser confundida com uma simples gesticulação ou com uma transposição direta da língua portuguesa. Trata-se de um sistema linguístico completo e autônomo, com estrutura própria nos níveis fonológico, morfológico, sintático e semântico.

Sua gramática baseia-se em cinco parâmetros visuais que incluem a configuração das mãos, o ponto de articulação no corpo ou no espaço, a direção e o tipo de movimento, a orientação das palmas das mãos e as expressões faciais e corporais.

A configuração de mão refere-se à forma que a mão assume durante o sinal, com variações na posição e nos movimentos dos dedos. O ponto de articulação indica o local do corpo ou do espaço em que o sinal é realizado, como frente ao peito ou próximo ao rosto. O movimento envolve a direção, a trajetória e a repetição do gesto, sendo essencial para distinguir sinais semelhantes. A orientação da palma da mão define para onde ela está voltada durante a execução do sinal, como para cima, para o interlocutor ou para o próprio corpo. Já as expressões faciais e corporais atuam como elementos gramaticais e emocionais, marcando, por exemplo, perguntas, negações ou intensidade, e são indispensáveis para a compreensão completa da mensagem [6].

Dentro da LIBRAS, os sinais podem ser classificados de forma geral em dois tipos principais. O primeiro corresponde aos sinais convencionais, também chamados de sinais lexicais, que representam palavras ou conceitos inteiros e que fazem parte do vocabulário natural da língua. O segundo tipo é a soletração manual, ou datilologia, que consiste na representação de palavras por meio da sequência de letras do alfabeto latino, cada uma

expressa por uma configuração específica da mão, realizada com uma única mão e de forma sequencial.

Essa forma de comunicação é utilizada principalmente nos seguintes contextos: na grafia de nomes próprios, palavras estrangeiras, siglas ou termos técnicos que ainda não possuem um sinal estabelecido na comunidade surda, bem como em situações em que se deseja enfatizar ou esclarecer algo. A Figura 2.1 mostra a configuração da mão que representa cada letra do alfabeto.

Figura 2.1: Configuração de mãos para cada letra do alfabeto
Fonte: Instituto Nacional de Educação de Surdos [7]

A soletração é um componente fundamental da linguagem e, embora menos frequente no discurso cotidiano em comparação com os sinais lexicais, exige alto grau de precisão e atenção tanto por parte de quem executa quanto de quem interpreta os sinais. Como cada letra possui uma configuração distinta de mão, e como a transição entre as letras deve ocorrer de forma fluida e natural, a prática da datilologia envolve coordenação motora refinada e rapidez de percepção visual.

A soletração manual, por sua natureza estruturada e seu vocabulário limitado às letras do alfabeto, oferece um cenário particularmente adequado para o desenvolvimento de sis-

temas automáticos de reconhecimento de sinais. Embora envolva desafios técnicos, como a identificação precisa de gestos realizados em sequência temporal, suas características visuais mais controladas e o uso de apenas uma mão reduzem significativamente a complexidade do problema em comparação com o reconhecimento de sinais lexicais completos. Isso torna a datilologia uma escolha estratégica como ponto de partida para pesquisas na área, permitindo a construção e o treinamento de modelos computacionais capazes de lidar com dados gestuais de forma mais acessível e controlada.

O reconhecimento computacional da LIBRAS como um todo, por outro lado, apresenta dificuldades adicionais, pois requer a interpretação simultânea de múltiplos elementos visuais, como movimentos corporais amplos, expressões faciais e articulação com ambas as mãos. Nesse contexto, focar inicialmente na soletração manual permite não apenas um recorte tecnicamente mais viável, mas também uma base sólida para avanços futuros no reconhecimento de estruturas linguísticas mais complexas.

2.3 Redes Neurais

As Redes Neurais Artificiais (RNAs) [8] compõem uma classe de modelos computacionais inspirados no funcionamento do cérebro humano, particularmente na forma como os neurônios biológicos processam e transmitem informações. Esses modelos são amplamente utilizados em tarefas de reconhecimento de padrões, classificação, previsão e aprendizado a partir de dados. A ideia central por trás das RNAs é reproduzir, em termos matemáticos, o comportamento coletivo de neurônios artificiais organizados em camadas e conectados por meio de pesos ajustáveis.

Um neurônio artificial é uma unidade básica de processamento em redes neurais. Ele recebe um ou mais valores de entrada, cada um multiplicado por um peso associado, realiza a soma ponderada desses valores e, em seguida, aplica uma função de ativação ao resultado. Essa função, que pode ser linear ou não linear (como ReLU, sigmoide ou tangente hiperbólica), determina a saída do neurônio. O objetivo é que o neurônio aprenda, durante o treinamento, a ajustar os pesos de forma a gerar saídas adequadas para os diferentes padrões de entrada, permitindo que a rede como um todo resolva tarefas como classificação, regressão ou reconhecimento de padrões.

As RNAs são compostas por unidades de neurônios artificiais, que estão organizadas em camadas: uma camada de entrada, uma ou mais camadas intermediárias (também chamadas de camadas ocultas) e uma camada de saída. Cada neurônio em uma camada está conectado aos neurônios da camada seguinte por meio de ligações chamadas de pesos sinápticos. Esses pesos determinam a importância que cada entrada tem sobre o processamento do neurônio subsequente. O funcionamento básico de um neurônio artificial

consiste em receber sinais de entrada, aplicar uma ponderação a esses sinais, somá-los e, em seguida, aplicar uma função de ativação para determinar sua saída. Essa função de ativação pode ser linear, sigmoide, tangente hiperbólica, *Rectified Linear Unit* (ReLU), entre outras, e sua escolha influencia diretamente o comportamento do modelo.

Durante o processo de treinamento, a rede neural ajusta iterativamente os pesos sinápticos com o objetivo de minimizar o erro entre a saída prevista e a saída desejada. Isso é feito por meio de algoritmos de otimização, como o gradiente descendente, geralmente associado ao método de retropropagação do erro (*backpropagation*), que permite o ajuste dos pesos de forma eficiente, mesmo em redes com muitas camadas. O aprendizado pode ocorrer de forma supervisionada, quando a rede recebe pares de entrada e saída desejada, ou de forma não supervisionada, quando ela tenta identificar padrões ou estruturas nos dados sem uma resposta explícita.

As redes neurais são particularmente eficazes na resolução de problemas que envolvem reconhecimento de padrões, classificação, regressão e geração de dados [9]. Em especial, problemas com dados altamente não lineares, ruidosos ou com estruturas complexas se beneficiam da capacidade das RNAs de modelar relações implícitas e de alta dimensionalidade. Essa versatilidade as tornou amplamente aplicáveis em áreas como visão computacional, processamento de linguagem natural, sistemas de recomendação, diagnóstico médico e, como no caso do presente trabalho, no reconhecimento automático de gestos.

No caso do reconhecimento de sinais em LIBRAS, as redes neurais são especialmente úteis porque os dados de entrada, imagens ou sequências de quadros de vídeo, contêm variações sutis, sequenciais e não triviais de serem mapeadas diretamente por regras fixas ou modelos estatísticos simples. A capacidade de generalização das RNAs, bem como sua habilidade de aprender representações intermediárias a partir dos dados brutos, tornam-nas ferramentas promissoras para lidar com a variabilidade inerente aos sinais manuais.

A definição da arquitetura da rede neural, incluindo o número de camadas, a quantidade de neurônios por camada, a escolha da função de ativação e o algoritmo de otimização utilizado, é um aspecto fundamental que impacta diretamente o desempenho do modelo. No desenvolvimento deste trabalho, diferentes arquiteturas foram experimentadas com o objetivo de avaliar suas capacidades na tarefa de reconhecimento da soletração em LIBRAS. Entre as abordagens testadas estão o Perceptron Multicamadas (do inglês *Multilayer Perceptron*) (MLP), a rede recorrente de Elman e a arquitetura *Long Short-Term Memory* (LSTM). Cada uma dessas técnicas será detalhada nas seções seguintes, com ênfase em seus princípios de funcionamento e no papel que desempenharam no processo experimental.

Além disso, o processo de treinamento de uma rede neural requer a seleção cuidadosa de parâmetros como a taxa de aprendizado, o número de épocas, o tamanho do lote

(*batch size*) e estratégias de regularização como *dropout* ou *early stopping*, a fim de evitar o sobreajuste (*overfitting*) e garantir boa generalização para novos dados.

Em resumo, as redes neurais artificiais são modelos computacionais inspirados na neurobiologia, capazes de aprender a partir de exemplos, identificar padrões complexos e tomar decisões baseadas em dados de entrada. Sua aplicabilidade no reconhecimento de sinais em LIBRAS advém justamente dessa flexibilidade e robustez frente à variabilidade de formas, movimentos e expressões que caracterizam a língua de sinais [8].

2.3.1 Perceptron Multicamadas (MLP)

O Perceptron Multicamadas (do inglês *Multilayer Perceptron*) (MLP) [10] é um dos modelos mais tradicionais e amplamente utilizados dentro do campo das redes neurais artificiais. Ele representa uma evolução do perceptron simples, proposto por Frank Rosenblatt na década de 1950, superando as limitações deste ao incorporar uma ou mais camadas ocultas entre a camada de entrada e a camada de saída. Essa adição de camadas intermediárias permite que o MLP seja capaz de resolver problemas não linearmente separáveis, como demonstrado formalmente a partir do teorema da aproximação universal.

A arquitetura básica de um MLP consiste em ao menos três camadas: a camada de entrada, que recebe os dados brutos (como vetores numéricos); uma ou mais camadas escondidas, onde ocorre o processamento intermediário por meio de neurônios artificiais interconectados; e a camada de saída, que produz o resultado final da rede, como a classificação ou a predição. Cada neurônio em uma camada é conectado a todos os neurônios da camada seguinte, formando uma rede densa. As conexões entre os neurônios são associadas a pesos sinápticos, que são ajustados durante o processo de treinamento [10]. A Figura 2.2 ilustra a configuração de um neurônio artificial do MLP.

A equação que descreve a saída do neurônio apresentado na Figura 2.2 é expressa pela Equação 2.1. Nessa expressão, y_i representa a saída do neurônio i, enquanto n indica o número total de entradas que o neurônio recebe. O termo w_{ij} corresponde ao peso sináptico associado à entrada j do neurônio i, e x_j representa o valor da j-ésima entrada. Além disso, o termo b_i refere-se ao viés (ou bias) do neurônio, que permite ajustar a função de ativação, deslocando-a para melhor adequação ao problema modelado. Esse viés é essencial para que o neurônio seja capaz de representar funções que não necessariamente passam pela origem.

$$y_i = \varphi\left(\sum_{j=1}^{n} W_{ij}x_j + b_i\right) \tag{2.1}$$

A Figura 2.3 ilustra a configuração das camadas de uma rede neural MLP.

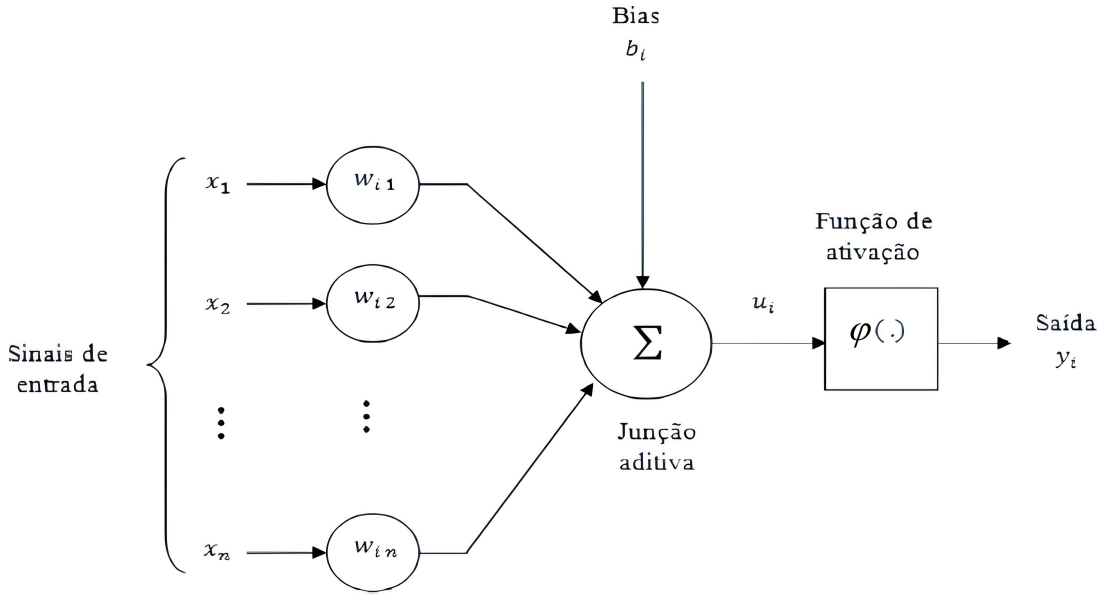

Figura 2.2: Ilustração do funcionamento de um neurônio artificial do MLP
Fonte: Simon Haykin [10]

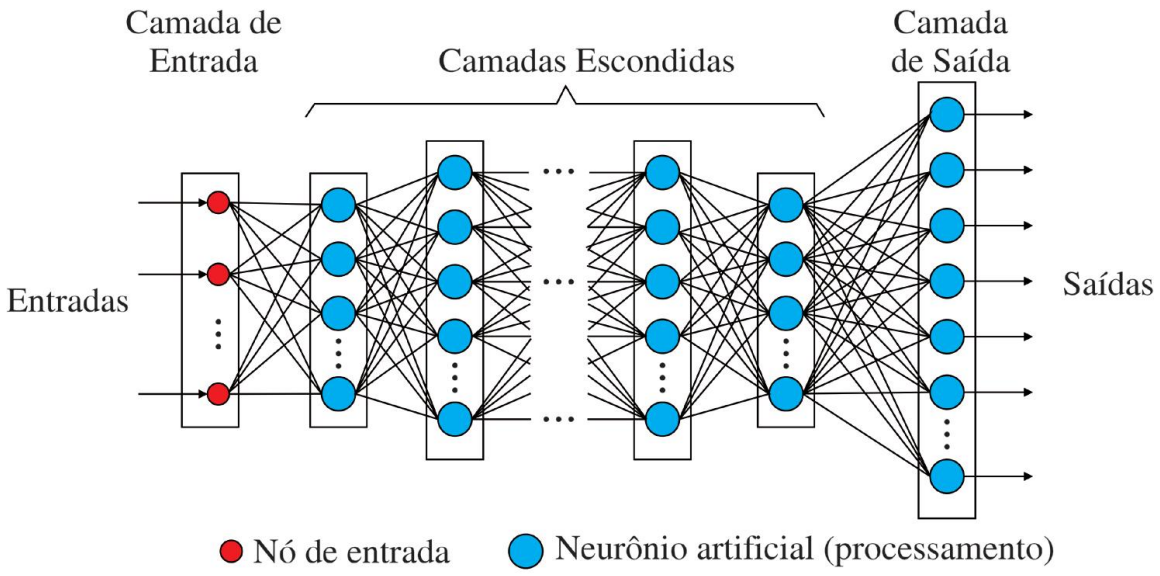

Figura 2.3: Ilustração das camadas de uma rede neural MLP
Fonte: Oscar Gabriel Filho [11]

O treinamento de um MLP pode ser dividido em duas fases principais: a propagação direta (*forward pass*) e a retropropagação do erro (*error backpropagation*). Na propagação direta, os dados de entrada atravessam a rede, camada por camada, passando por somas ponderadas e funções de ativação não lineares, como a ReLU, a sigmoide ou a tangente hiperbólica. O resultado final chega à camada de saída, onde é comparado à resposta

esperada. Em seguida, na fase de retropropagação, calcula-se o erro entre a saída produzida e o valor esperado. Esse erro é propagado de volta pelas camadas da rede, ajustando os pesos de forma a minimizar uma função de custo (como o erro quadrático médio ou a entropia cruzada), por meio de métodos como o Gradiente Descendente Estocástico (do inglês *Stochastic Gradient Descent*) (SGD) ou seus aprimoramentos, como Adam.

Uma das principais vantagens do MLP é sua capacidade de aprender representações internas e não lineares dos dados, o que o torna adequado para tarefas complexas, como classificação de imagens, reconhecimento de padrões e processamento de sinais. No entanto, o treinamento eficaz de um MLP depende de vários fatores, incluindo a quantidade de dados, a escolha da função de ativação, a taxa de aprendizado, o número de camadas e neurônios, e técnicas de regularização, como o *dropout*, que ajudam a evitar o sobreajuste (*overfitting*) [10].

2.3.2 Rede Recorrente de Elman

A rede de Elman, proposta por Jeffrey Elman em 1990 [12], é um tipo específico de Rede Neural Recorrente (do inglês *Recurrent Neural Network*) (RNN) projetada para lidar com dados sequenciais, onde a ordem temporal das informações é relevante. Diferentemente das redes neurais tradicionais do tipo *feedforward*, como o Perceptron Multicamadas (do inglês *Multilayer Perceptron*) (MLP), que tratam cada entrada de forma independente, as redes recorrentes introduzem conexões internas que permitem à rede manter uma memória do que foi processado anteriormente. Essa característica torna as RNNs particularmente eficazes para tarefas como reconhecimento de fala, processamento de linguagem natural e, no caso desta monografia, o reconhecimento de gestos manuais realizados em sequência temporal, como ocorre na soletração em LIBRAS.

A arquitetura da rede de Elman é composta por três camadas principais: a camada de entrada, camada(s) oculta(s) e a camada de saída. O diferencial dessa estrutura está na introdução de uma camada de contexto (ou de estado), que armazena a saída da camada oculta no instante de tempo anterior e a realimenta como entrada adicional na próxima iteração. Assim, a cada novo vetor de entrada apresentado à rede, a camada oculta recebe não apenas esse vetor atual, mas também as ativações da camada oculta da etapa anterior. Essa realimentação confere à rede a capacidade de aprender dependências temporais curtas, capturando relações entre os elementos da sequência. A Figura 2.4 mostra o diagrama de uma rede de Elman.

Matematicamente, o funcionamento da rede de Elman pode ser descrito pelas equações a seguir. Onde a Equação 2.2 define a ativação da camada oculta no tempo t, levando em conta tanto a entrada atual quanto a memória do instante anterior da própria camada oculta. A Equação 2.2 mostra a saída $h_k(t)$ de um neurônio da rede.

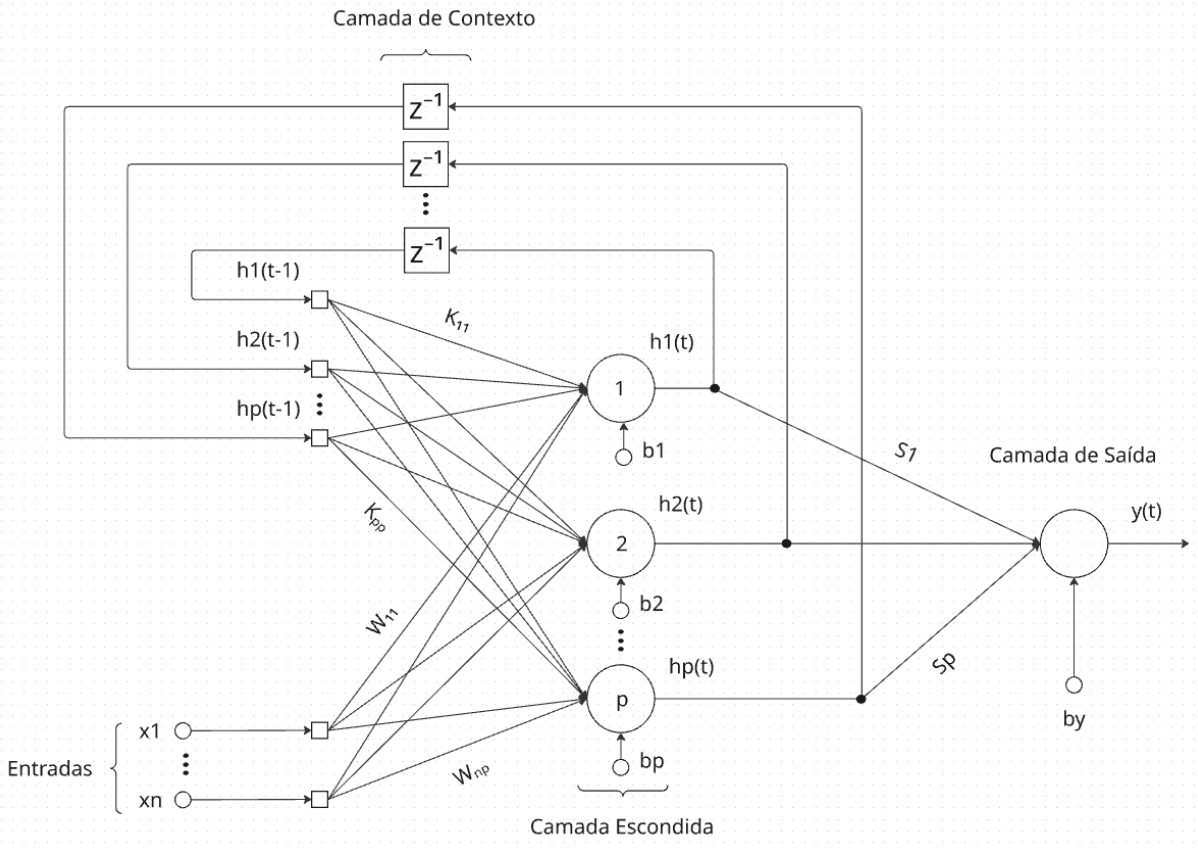

Figura 2.4: Diagrama de uma rede de Elman
Fonte: Próprio autor.

$$h_k(t) = \varphi\left(\sum_{j=1}^{n} W_{jk}x_j + \sum_{j=1}^{p} K_{jk}h_j(t-1) + b_k\right) \tag{2.2}$$

Nessa equação, x_j representa o valor da j-ésima entrada fornecida ao neurônio, enquanto W_{jk} corresponde ao peso associado a essa entrada, que define sua importância relativa no processamento. O termo $h_j(t-1)$ refere-se à j-ésima ativação da camada oculta no instante de tempo anterior, ou seja, a memória trazida da etapa anterior pela camada de contexto. O peso correspondente a essa ativação passada é indicado por K_{jk}, controlando a influência da memória sobre o estado atual do neurônio. Por fim, o termo b_k representa o bias aplicado ao neurônio k, atuando como um deslocamento que ajusta o valor total recebido antes da aplicação da função de ativação $\varphi(\cdot)$. Já a Equação 2.3 mostra a saída $y(t)$ da rede.

$$y(t) = \varphi\left(\sum_{j=1}^{p} S_j h_j(t) + b_y\right) \tag{2.3}$$

Nessa equação, $h_j(t)$ representa a ativação do j-ésimo neurônio da camada oculta no

instante de tempo atual, ou seja, o valor de saída desse neurônio após o processamento do *Time Step* corrente. O termo S_j corresponde ao peso associado a essa saída, determinando a contribuição de cada neurônio da camada oculta para o cálculo da saída final da rede. Além disso, o bias b_y é adicionado à soma ponderada das ativações, funcionando como um ajuste adicional antes da aplicação da função de ativação $\varphi(\cdot)$, que introduz a não-linearidade necessária para que o modelo capture relações complexas nos dados.

Em redes neurais recorrentes, como a rede de Elman, o conceito de *Time Step* (ou etapas temporais) é essencial para o processamento de informações sequenciais. Cada *Time Step* representa um ponto no tempo em que um vetor de entrada é apresentado à rede. Diferentemente de redes *feedforward* tradicionais, que processam entradas de forma independente, as redes recorrentes mantêm um estado interno que é atualizado a cada novo *Time Step*, permitindo que informações sobre entradas passadas influenciem o processamento atual.

Esse estado interno é preservado na chamada camada de contexto, que funciona como uma memória de curto prazo, armazenando as ativações anteriores da camada oculta. Assim, ao processar uma sequência de *Time Step*s, a rede consegue capturar dependências e padrões temporais presentes nos dados, como tendências, ritmos ou transições entre eventos.

No entanto, essa memória acumulada não é infinita: geralmente, as redes são treinadas em janelas delimitadas de *Time Step*s. Após o processamento de um determinado número de *Time Step*s, o estado interno pode ser reinicializado, reiniciando o ciclo de memória da rede. A escolha da quantidade de *Time Step*s utilizados em cada janela de processamento é um parâmetro importante: janelas curtas podem limitar a capacidade da rede de capturar relações de longo prazo, enquanto janelas muito longas podem introduzir dificuldades no treinamento, como o desaparecimento ou explosão do gradiente. Por isso, definir um número adequado de *Time Step*s é crucial para otimizar o aprendizado sequencial em redes recorrentes.

A camada de contexto é atualizada automaticamente a cada iteração, mantendo a memória da sequência processada até o momento. Essa estrutura recorrente, embora simples, é suficiente para permitir que a rede modele padrões temporais em dados sequenciais curtos. Entretanto, a rede de Elman possui limitações, especialmente em tarefas que exigem memórias de longo prazo. Em tais casos, redes mais sofisticadas, como as *Long Short-Term Memory*, que serão tratadas a seguir, tendem a apresentar desempenho superior, pois foram projetadas para mitigar problemas como o desaparecimento ou explosão do gradiente durante o treinamento [12].

2.3.3 Long Short-Term Memory (LSTM)

A arquitetura *Long Short-Term Memory* (LSTM) é um tipo especial de Rede Neural Recorrente (do inglês *Recurrent Neural Network*) (RNN), desenvolvida por Hochreiter e Schmidhuber em 1997 [13], com o objetivo de superar as principais limitações das redes recorrentes tradicionais, como a rede de Elman, sobretudo no que diz respeito à dificuldade em aprender dependências de longo prazo em sequências temporais. As RNNs convencionais, embora eficazes em capturar padrões curtos, sofrem com o problema conhecido como desvanecimento ou explosão do gradiente durante o treinamento por retropropagação através do tempo (*Backpropagation Through Time* (BPTT)), o que compromete seriamente sua capacidade de armazenar e processar informações ao longo de muitas etapas temporais.

O diferencial das LSTMs está em sua estrutura interna, que incorpora um mecanismo de memória explícita controlada por portas (*gates*). Em vez de apenas atualizar o estado oculto a cada *Time Step* com base na entrada atual e no estado anterior, a LSTM mantém um vetor de estado de célula que funciona como uma espécie de "linha de memória", e o atualiza de forma seletiva com a ajuda de três portas principais: a porta de entrada, a porta de esquecimento e a porta de saída. Cada uma dessas portas é composta por neurônios com ativação sigmoide, responsáveis por controlar o fluxo de informações dentro da célula. Abaixo pode ser visto um diagrama dessa estrutura da LSTM no Figura 2.5

Além das três portas já mencionadas, a arquitetura da LSTM também conta com uma quarta operação essencial, frequentemente referida como o vetor candidato ou porta candidata, embora tecnicamente não seja uma porta, mas sim uma transformação intermediária crucial no processo de atualização da célula.

A porta de esquecimento (*forget gate*) recebe como entrada o vetor de entrada no tempo atual e o estado oculto do tempo anterior, aplicando uma função sigmoide para gerar um vetor com valores entre 0 e 1. Cada valor neste vetor indica a proporção de informação do estado de célula anterior que deve ser retida ou descartada. Assim, a porta de esquecimento permite que a LSTM apague seletivamente memórias antigas que já não são relevantes para a tarefa atual.

A porta de entrada (*input gate*) também recebe a entrada atual e o estado oculto anterior, e gera dois vetores — um vetor sigmoide que atua como filtro, decidindo quais valores serão atualizados, e um vetor tangente hiperbólica que produz os chamados valores candidatos, que representam novas informações a serem adicionadas ao estado da célula. A combinação desses dois vetores define quais dados novos devem ser incorporados à memória.

A porta candidata (*cell candidate*) não é uma porta no sentido tradicional, mas representa o vetor de valores candidatos gerado por uma ativação *tanh*, que propõe uma nova

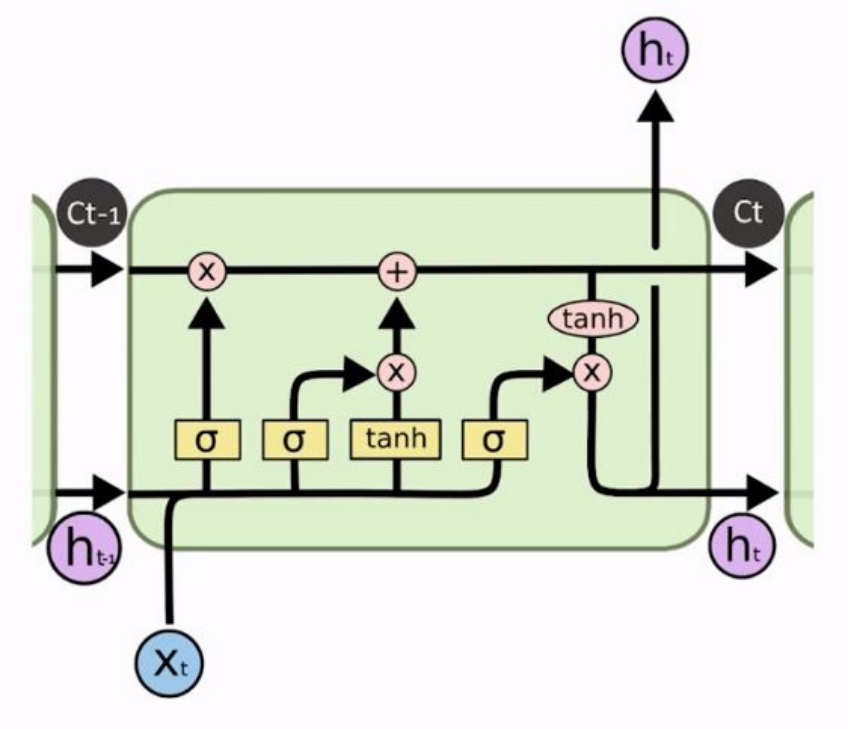

Figura 2.5: Diagrama de um neurônio da rede LSTM
Fonte: Equipe Didática Tech [14]

informação a ser potencialmente armazenada na célula. A filtragem desse vetor ocorre por meio da multiplicação elemento a elemento com a saída da porta de entrada, o que assegura que apenas os dados considerados relevantes sejam efetivamente armazenados.

Já a porta de saída (*output gate*) determina qual parte da informação da célula de memória será utilizada para gerar a saída da LSTM naquele instante. Ela também usa uma função sigmoide para calcular um vetor de ativação com base na entrada atual e no estado oculto anterior, e este vetor é multiplicado pelo estado da célula (após passar por uma função *tanh*) para formar o novo estado oculto, que será propagado para o próximo tempo e utilizado como saída atual da rede.

Essa combinação de operações possibilita que a célula da LSTM selecione de forma dinâmica o que lembrar, o que esquecer e o que expor como saída, mitigando os problemas do desvanecimento do gradiente e permitindo que a rede mantenha informações úteis por longos períodos [13].

2.4 Bibliotecas e Ferramentas Utilizadas

Nesta seção, serão apresentadas e discutidas as principais bibliotecas e tecnologias utilizadas no desenvolvimento do sistema proposto, todas elas integradas à linguagem de

programação Python, escolhida por sua versatilidade, ampla comunidade e ecossistema robusto voltado para aplicações de inteligência artificial e ciência de dados. As ferramentas abordadas incluem o *MediaPipe*, utilizado para a detecção e extração de pontos de referência corporais a partir de vídeos, especialmente os marcos das mãos essenciais para o reconhecimento da datilologia; o OpenCV, que teve papel importante no processamento de vídeo, como a leitura de arquivos, extração de *frames* e conversão de formatos de imagem, integrando-se diretamente às demais etapas do sistema; o *scikit-learn (sklearn)*, uma biblioteca amplamente utilizada para tarefas de aprendizado de máquina, que neste trabalho, foi responsável pela implementação do algoritmo *k-Nearest Neighbors* (k-NN); e o *TensorFlow*, empregado no treinamento e execução das redes neurais artificiais que processam as sequências de gestos. Além dessas, outras bibliotecas e ferramentas auxiliares foram utilizadas para suporte à manipulação de dados e geração de gráficos.

2.4.1 Mediapipe e OpenCV

O *MediaPipe* [15] é uma biblioteca desenvolvida pelo Google voltada para o processamento de mídia multimodal, com foco especial em visão computacional em tempo real. Trata-se de um *framework* poderoso e altamente otimizado para a criação de pipelines que processam e interpretam dados visuais, como vídeos e imagens, utilizando técnicas de aprendizado de máquina. Um dos principais diferenciais do *MediaPipe* é sua capacidade de realizar detecção e rastreamento de estruturas corporais — como mãos, face, corpo inteiro e pose — de maneira rápida e precisa, mesmo em dispositivos com recursos computacionais limitados. Essa característica torna a ferramenta ideal para aplicações interativas, como sistemas de reconhecimento de gestos, que dependem da análise contínua de imagens.

O *Open Source Computer Vision Library* (OpenCV) [16] é uma biblioteca de código aberto amplamente utilizada para aplicações de visão computacional e processamento de imagens. Desenvolvida originalmente pela Intel, ela oferece uma vasta gama de funções otimizadas para captura, manipulação, análise e visualização de imagens e vídeos. Suportando múltiplas linguagens de programação, como Python e C++. Serão detalhadamente explicadas as principais funções das bibliotecas utilizadas ao longo deste trabalho.

Começando pelo OpenCV, que foi utilizada principalmente nas etapas de captura e manipulação de vídeo, servindo como ponte entre os arquivos visuais brutos e o processamento com outras bibliotecas, como o *MediaPipe*. A primeira função empregada foi `cv2.VideoCapture("caminho")`, responsável por abrir o vídeo localizado no caminho especificado e inicializar um objeto que permite a leitura sequencial de seus *frames*. Caso seja passado o valor 0 como argumento, o OpenCV utiliza a câmera padrão do computador em vez de um arquivo de vídeo. Se houver mais de uma câmera conectada ao sistema, é possível acessar as demais utilizando os índices 1, 2 e assim por diante. A partir desse

objeto, é possível utilizar o método `cap.read()`, que realiza a leitura do próximo *frame* disponível no vídeo. Esse método retorna dois valores: o primeiro, denominado *success*, é um valor booleano que indica se a leitura foi bem-sucedida; o segundo, *frame*, é a imagem do *frame* capturada, que será processada nas etapas seguintes.

Após a captura, o *frame* obtido é convertido do formato padrão do OpenCV (BGR — Blue, Green, Red) para o formato RGB, que é exigido pelas funções do MediaPipe. Essa conversão é feita com a função `cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)`. Em seguida, é comum aplicar um redimensionamento à imagem utilizando `cv2.resize(src, dsize, fx, fy)`, seja para uniformizar a entrada do modelo ou para otimizar o desempenho do sistema durante o processamento em tempo real.

Durante a fase de testes e coleta, a visualização dos resultados foi realizada com `cv2.imshow("capture image", frame)`, que exibe o *frame* em uma janela separada, frequentemente com as anotações do *MediaPipe* sobrepostas. Para controlar o fluxo de exibição, é utilizada a função `cv2.waitKey(s)` que aguarda que o usuário pressione uma tecla. Caso o argumento s seja zero, a espera é indefinida; caso seja um número positivo, ele representa o tempo de espera em milissegundos — geralmente, define-se esse valor como 1 para permitir a continuidade do laço de execução sem bloqueios. Ao final da execução do sistema, a função `cv2.destroyAllWindows()` é chamada para garantir que todas as janelas abertas pelo OpenCV sejam fechadas adequadamente. A Figura 2.6 mostra um exemplo de imagem capturada pelo OpenCV, mostra também os pontos gerados pelo *Mediapipe* em cima da imagem para *feedback* visual.

Figura 2.6: Exemplo de imagem capturada com o OpenCV com os pontos desenhados pelo mediapipe

Fonte: Próprio autor.

No contexto deste trabalho, a biblioteca *MediaPipe* foi empregada principalmente para

a detecção das mãos e a extração dos pontos de referência (*landmarks*) necessários para representar as posturas manuais da datilologia em LIBRAS. A função inicial utilizada foi `mp.solutions.hands.Hands(max_num_hands=1)`, que cria uma instância do módulo `Hands`, responsável por realizar a detecção das mãos em imagens e identificar 21 pontos específicos em cada uma delas. O argumento `max_num_hands=1` indica que o sistema deve considerar, no máximo, uma mão por *frame*, o que está de acordo com a natureza da soletração em LIBRAS, feita com uma única mão.

Uma vez criada a instância do detector, utiliza-se o método `.process(RGB_frame)` para processar o *frame* convertido para o formato RGB. Esse método realiza a inferência e tenta identificar a presença de uma ou mais mãos na imagem. Caso a detecção seja bem-sucedida, a função retorna dois conjuntos principais de dados: `multi_hand_landmarks` e `multi_hand_world_landmarks`. Ambos contêm as coordenadas tridimensionais (x, y e z) dos 21 pontos da mão identificados pelo modelo, mas diferem no sistema de referência utilizado.

O primeiro, `multi_hand_landmarks`, representa os pontos em um sistema de coordenadas normalizado, em que os valores de x e y variam de 0 a 1, de acordo com a largura e a altura da imagem, enquanto o eixo z é relativo à profundidade do pulso, considerado como origem. Já `multi_hand_world_landmarks` fornece coordenadas em um sistema métrico tridimensional, cuja origem é estimada no centro geométrico da mão detectada, e cujas medidas estão expressas em metros. Esse segundo retorno é especialmente útil em aplicações que requerem análise mais precisa de posições espaciais ou reconstrução em 3D [17].

Neste trabalho, o vetor de características utilizado no treinamento da rede neural é extraído diretamente de `result.multi_hand_world_landmarks[0]`, que corresponde à primeira (e única) mão detectada no *frame*. A partir desse objeto, percorrem-se os 21 pontos com o acesso `.landmark[i].x`, `.landmark[i].y` e `.landmark[i].z`, extraindo-se assim todas as 63 coordenadas que compõem a entrada do modelo. A Figura 2.7 mostra os 21 pontos capturados na mão pelo *Mediapipe*.

Para fins de visualização durante o desenvolvimento e testes, foi utilizada a função `mp_drawing.draw_landmarks(frame, hand_landmarks, hand_connections)`. Ela desenha, sobre o *frame* original, os pontos identificados na mão e as conexões entre eles, proporcionando um *feedback* visual importante para verificar se a detecção está ocorrendo de maneira correta e precisa. Embora não influencie diretamente os dados utilizados para treinamento, essa visualização auxilia no processo de validação da coleta de dados.

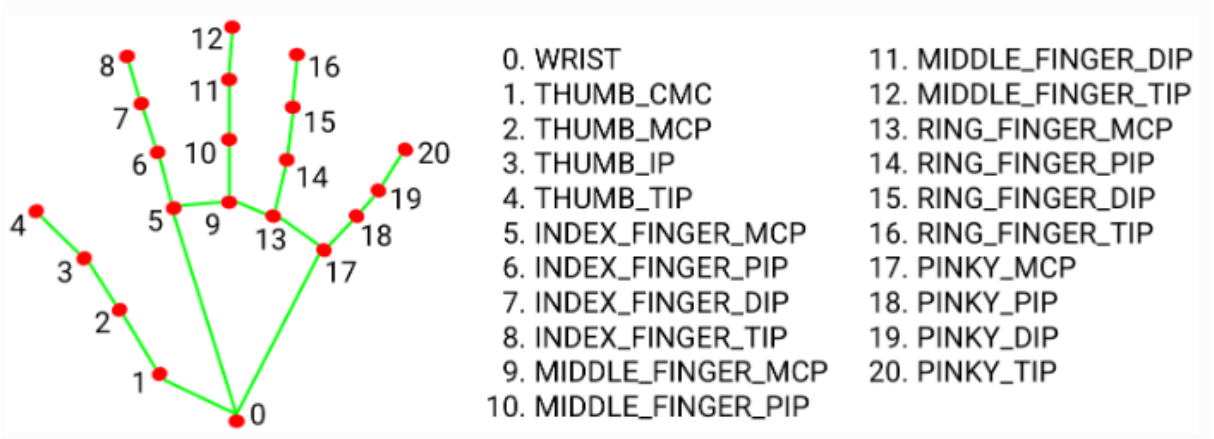

Figura 2.7: Pontos capturados pelo Mediapipe
Fonte: Mediapipe [17]

2.4.2 Scikit-learn (sklearn)

A biblioteca *scikit-learn* (ou *sklearn*) [18] desempenhou um papel importante neste trabalho, sendo utilizada tanto para a implementação do algoritmo *k-Nearest Neighbors* (k-NN) quanto para o suporte à divisão dos dados nas etapas de treinamento, validação e teste do MLP. A função `train_test_split(x, y, test_size)` foi empregada para separar os conjuntos de dados. Essa função recebe como argumentos o vetor x, que corresponde aos dados de entrada — no contexto deste projeto, os vetores de características extraídos dos pontos de referência (*landmarks*) das mãos —, o vetor y, que representa os rótulos associados a cada entrada (neste caso, as letras soletradas em LIBRAS), e o parâmetro `test_size`, que define a proporção dos dados a ser destinada aos subconjuntos de teste e/ou validação.

A função `KNeighborsClassifier(n_neighbors=k)` foi utilizada para a implementação do classificador k-NN, ela cria uma instância do modelo e define o número de vizinhos mais próximos a serem considerados no processo de classificação. Em seguida, o método `knn.fit(x_train, y_train)` é utilizado para treinar o modelo com os dados previamente separados para essa finalidade. Após o treinamento, a etapa de inferência é realizada com `knn.predict(x_test)`, permitindo gerar previsões a partir do conjunto de teste. Por fim, a função `accuracy_score(y_test, y_pred)` é empregada para calcular a acurácia do modelo, comparando os rótulos previstos com os rótulos reais. Essa métrica oferece uma visão quantitativa do desempenho do classificador, sendo útil para comparação com outros métodos, como as redes neurais recorrentes utilizadas neste trabalho.

2.4.3 Tensorflow

A biblioteca *TensorFlow* [19] foi a principal ferramenta utilizada para a construção e treinamento dos modelos baseados em redes neurais, incluindo o Perceptron Multicamadas (do inglês *Multilayer Perceptron*) (MLP), a rede recorrente de Elman (RNN simples) e a rede LSTM. Entre esses, o modelo de Elman é o foco central deste trabalho, e por isso suas funções e estrutura serão exploradas com maior profundidade.

A criação do modelo foi realizada por meio da classe `tf.keras.Sequential()`, que permite construir arquiteturas de rede de forma sequencial, ou seja, camada após camada, em uma pilha linear. Essa classe recebe como argumento uma lista de camadas, cada uma representando uma etapa do processamento dentro da rede. A primeira camada adicionada ao modelo foi `tf.keras.layers.SimpleRNN()`, que implementa uma RNN simples, equivalente à estrutura da rede de Elman. Entre os argumentos dessa camada estão o número de neurônios (ou unidades recorrentes), a função de ativação, que neste trabalho foi definida como "relu" (*Rectified Linear Unit*), adequada para evitar problemas como o desaparecimento do gradiente, o parâmetro `return_sequences=True`, que garante que a rede retorne uma saída para cada passo temporal da sequência de entrada (em vez de apenas a última saída), e o `input_shape`, que define o formato dos dados de entrada. Neste caso, `input_shape=(TIMESTEPS, tx_train.shape[2])` especifica que a entrada será uma sequência temporal, definindo o tamanho do vetor de *Time Step*s, sendo que cada vetor possui `tx_train.shape[2]` características, equivalentes às coordenadas extraídas dos marcos da mão.

A segunda camada adicionada ao modelo foi `tf.keras.layers.TimeDistributed()`, que é utilizada para aplicar uma mesma camada de forma independente a cada passo temporal da saída da RNN. Dentro dessa camada, é inserida uma Dense, ou seja, uma camada totalmente conectada, responsável por gerar a saída do modelo. Essa estrutura é fundamental no contexto da tarefa de classificação por sequência, pois permite que a rede forneça uma previsão em cada instante temporal da entrada — algo desejado, por exemplo, na tarefa de decodificar gestos contínuos em LIBRAS. O uso de `TimeDistributed(Dense(...))` assegura que cada vetor de saída da RNN seja mapeado para um vetor de probabilidades sobre as classes possíveis, facilitando a interpretação dos resultados e a comparação com os rótulos durante o treinamento e a validação. No caso específico deste trabalho, foi utilizada a função `Dense(NUMERO_LETRAS, activation='softmax')`, na qual o parâmetro `NUMERO_LETRAS` representa a quantidade total de classes, ou seja, as diferentes letras que podem ser reconhecidas e o símbolo indicativo de transição, e a função de ativação *softmax* transforma a saída da rede em uma distribuição de probabilidades, garantindo que os valores somem 1 e permitindo interpretar qual letra possui a maior probabilidade de ter sido representada em cada *Time Step*

da sequência.

A etapa seguinte na definição do modelo com o *TensorFlow* é a compilação, realizada por meio da função `model.compile()`. Essa função prepara o modelo para o processo de treinamento, configurando o otimizador, a função de perda e as métricas de avaliação. O primeiro argumento fornecido à função é `optimizer='adam'`, que define o algoritmo responsável pela atualização dos pesos da rede durante o treinamento. O *Adaptive Moment Estimation* (Adam) é amplamente utilizado em problemas de aprendizado profundo devido à sua eficiência computacional, estabilidade e capacidade de adaptação automática da taxa de aprendizado. Além disso, ele combina as vantagens de outros otimizadores, como o *momentum* e o *RMSProp*, dispensando ajustes manuais complexos na maioria dos casos.

O próximo é `loss=SparseCategoricalCrossentropy(from_logits=False)`, que especifica a função de perda a ser utilizada. A função de perda (ou custo) é uma medida que quantifica o quão distante as predições do modelo estão dos valores reais (os rótulos). Neste trabalho, foi escolhida a `SparseCategoricalCrossentropy` por se tratar de um problema de classificação com múltiplas classes mutuamente exclusivas, em que os rótulos são representados por inteiros (por exemplo, a letra "a" é representada pelo valor 0, "b" pelo valor 1, e assim por diante). O argumento `from_logits=False` informa à função que as saídas do modelo já foram normalizadas com uma função *softmax*, o que é o caso na arquitetura proposta; se o *softmax* não tivesse sido aplicado na camada de saída, esse argumento deveria ser ajustado para `True`.

O terceiro argumento, `metrics=[‘accuracy’]`, define quais métricas devem ser monitoradas durante o processo de treinamento e validação. A métrica escolhida neste trabalho foi a acurácia, que mede a proporção de classificações corretas, ou seja, a frequência com que a predição do modelo coincide com o rótulo real. Essa métrica é particularmente útil para problemas de classificação balanceada, como o reconhecimento de letras isoladas na soletração em LIBRAS, permitindo uma avaliação direta do desempenho do modelo durante os ciclos de treinamento.

Com o modelo devidamente compilado, a etapa seguinte consiste no seu treinamento, realizado com a função `model.fit()`. Essa função inicia o processo de ajuste dos pesos da rede neural com base nos dados fornecidos, permitindo que o modelo aprenda a mapear corretamente as entradas para os respectivos rótulos. Os dois primeiros argumentos passados à função são `x_train` e `y_train`, correspondendo, respectivamente, aos vetores de entrada — neste caso, os vetores de características derivados dos marcos das mãos — e aos rótulos associados, representando as letras da soletração.

Em seguida, define-se o número de épocas por meio do argumento `epochs=1`. A opção por realizar uma época por chamada da função se justifica pela estratégia adotada no projeto, em que a validação é executada manualmente a cada ciclo de treinamento,

permitindo controle mais granular sobre o desempenho do modelo e a possibilidade de ajuste fino entre as iterações. Também é especificado o parâmetro `batch_size=32`, que determina o tamanho dos mini-lotes de dados utilizados durante o treinamento. Ao dividir o conjunto de dados em lotes de 32 amostras, o modelo processa cada lote individualmente antes de atualizar os pesos, o que proporciona economia de memória e maior estabilidade na convergência do processo de otimização.

Outro parâmetro essencial nesta implementação é `shuffle=False`. Por padrão, o *TensorFlow* embaralha os dados antes de cada época para evitar que o modelo memorize padrões artificiais na ordem das amostras. No entanto, no presente trabalho, como os dados consistem em sequências temporais associadas a movimentos gestuais, a preservação da ordem original é fundamental para que a rede possa aprender corretamente a dinâmica dos gestos ao longo do tempo. Em razão disso, o embaralhamento foi desativado.

O argumento `validation_data=(tx_validation, ty_validation)` é utilizado para fornecer ao modelo um conjunto separado de dados que será avaliado ao final de cada época, permitindo monitorar a capacidade de generalização da rede em dados não vistos durante o treinamento. Essa prática é essencial para evitar o sobreajuste (*overfitting*) e verificar se o modelo está de fato aprendendo a tarefa desejada. Por fim, o parâmetro `verbose=1` é utilizado apenas para configurar o nível de detalhamento das mensagens exibidas no terminal durante o treinamento. Com esse valor, o TensorFlow apresenta uma barra de progresso com as métricas de desempenho ao longo das épocas, o que facilita o acompanhamento do processo.

Por fim, após o treinamento, é fundamental preservar o modelo treinado para que ele possa ser reutilizado posteriormente sem a necessidade de reexecutar todo o processo de aprendizado. Para isso, utiliza-se a função `model.save('arquivo.keras')`, que salva a estrutura do modelo, os pesos treinados e a configuração de treinamento (incluindo otimizador, função de perda e métricas) em um único arquivo com a extensão `.keras`. Esse procedimento facilita tanto a continuidade dos experimentos quanto a aplicação prática do modelo treinado, permitindo sua reutilização em outros ambientes ou fases do projeto.

Para restaurar o modelo salvo, utiliza-se a função `load_model('arquivo.keras')`, que carrega o conteúdo do arquivo e o armazena em uma variável, restaurando completamente a arquitetura e os pesos do modelo no estado em que foi salvo. Isso permite que o modelo possa ser testado, avaliado ou utilizado para novas predições sem qualquer necessidade de reconfiguração manual.

Com o modelo carregado, é possível realizar uma avaliação quantitativa de seu desempenho utilizando a função `model.evaluate(tx_test, ty_test, verbose=1)`. Essa função aplica o modelo sobre o conjunto de teste previamente separado, calculando as

métricas especificadas na compilação — no caso, a perda (*loss*) e a acurácia — com base nas previsões realizadas e nos rótulos reais. O argumento `verbose=1` garante que os resultados da avaliação sejam exibidos no terminal de forma detalhada, permitindo verificar o desempenho final do modelo em dados não vistos durante o treinamento ou validação. Essa etapa é essencial para validar a generalização da rede e confirmar sua eficácia.

2.4.4 Outras Bibliotecas e Ferramentas

Além das bibliotecas principais discutidas anteriormente, outras ferramentas secundárias foram empregadas ao longo do desenvolvimento do sistema, com o objetivo de dar suporte às tarefas de pré-processamento, visualização e avaliação dos dados e resultados. Embora não sejam o foco central deste trabalho, essas bibliotecas desempenharam papéis relevantes na organização e condução dos experimentos.

A biblioteca *NumPy*, amplamente utilizada na área de ciência de dados, foi empregada principalmente para a manipulação de *arrays* e vetores numéricos. Ela oferece uma estrutura eficiente para o tratamento de grandes volumes de dados em forma de matrizes multidimensionais, com operações vetorizadas de alto desempenho. No contexto deste trabalho, o *NumPy* foi fundamental para organizar os vetores de características (extraídos dos marcos da mão) de forma compatível com a entrada esperada pelas redes neurais implementadas no *TensorFlow*, além de auxiliar em operações básicas de concatenação, normalização e estruturação dos dados.

A biblioteca *Matplotlib*, mais especificamente o módulo `matplotlib.pyplot`, foi utilizada para a geração de gráficos que auxiliam na visualização dos resultados. Com ela, foi possível traçar curvas de perda (*loss*) ao longo das épocas de treinamento, fornecendo uma visão clara sobre a convergência dos modelos. Além disso, o *Matplotlib* também foi usado na criação de histogramas que ilustram a distribuição das letras nos vídeos utilizados, o que contribuiu para a análise do valor dos *Time Step*s.

Foi empregada a sub-biblioteca `sklearn.metrics`, pertencente ao *scikit-learn*, para a avaliação do desempenho dos modelos. Dentre suas funcionalidades, destaca-se o uso da matriz de confusão, que permite visualizar de forma detalhada a quantidade de acertos e erros cometidos pelo modelo em relação a cada classe. Essa matriz é especialmente útil em tarefas de classificação multiclasse, como o reconhecimento de letras na datilologia, pois revela não apenas a acurácia global, mas também os padrões de confusão entre letras semelhantes.

Por fim, foi utilizada a técnica de validação cruzada do tipo *k-fold*, com o intuito de melhorar a robustez das avaliações. Nessa abordagem, os dados são divididos em k subconjuntos (ou "*folds*"), sendo que em cada iteração um desses subconjuntos é utilizado como teste, enquanto os demais são usados para treino e validação. Ao final das

k iterações, os resultados são combinados para fornecer uma estimativa mais confiável da performance do modelo, reduzindo o viés introduzido por uma única divisão dos dados. Essa técnica é especialmente importante quando se trabalha com conjuntos de dados limitados, como é o caso neste projeto.

2.5 Estado da Arte

Nesta seção, será apresentada uma breve revisão do estado da arte relacionado ao reconhecimento de linguagens de sinais com o uso de redes neurais artificiais e outras abordagens de inteligência artificial. O objetivo é contextualizar o desenvolvimento do presente trabalho dentro do panorama atual da pesquisa científica, identificando as principais técnicas utilizadas, os avanços recentes e os desafios ainda em aberto na área. Para isso, foram selecionados alguns artigos e estudos recentes que abordam soluções para a tradução automática de sinais. A seguir, cada um desses trabalhos será comentado de forma concisa, destacando suas metodologias, resultados e contribuições relevantes para o campo.

Lamar *et al.* [20] [21] apresentam um sistema para reconhecimento automático da soletração na *Japanese Sign Language* (JSL) e na *American Sign Language* (ASL) baseado em uma nova estrutura de Rede Neural que combina redes recorrentes e não recorrentes. A proposta sugere o uso uma luva de baixo custo com os dedos coloridos, que facilita a extração de características da postura e movimento da mão. O trabalho reporta um desempenho de 89% em um conjunto de vídeos composto de frames de tamanho 160×120 a 24 quadros por segundo (fps), 42 posturas manuais e 5 movimentos manuais da JSL. Sistema obteve uma acurácia de 94% no conjunto de 26 sinais da ASL.

Kumar *et al.* [22] propõem um método de reconhecimento e tradução da *Indian Sign Language* (ISL) por meio de uma arquitetura baseada em redes recorrentes com atenção e ativação adaptativa, chamada *Self-Attention Long-Short-Term Memory with Shape Autotuning Activation Function* (SALSTM-SAAF). O sistema realiza todas as etapas do processo de reconhecimento: aquisição de dados, pré-processamento, extração de características e classificação. Um diferencial do trabalho é a extração de características com a rede *ResNet-50*, que permite preservar detalhes espaciais complexos dos gestos, e o uso da camada de atenção para melhorar o aprendizado sequencial das redes LSTM. A função de ativação SAAF, por sua vez, se ajusta dinamicamente ao formato dos dados de entrada, melhorando o desempenho da rede em diferentes condições.

A rede SALSTM-SAAF demonstrou desempenho superior quando comparada a outros modelos com a mesma função de ativação. O modelo proposto alcançou acurácia de 99,87%, destacando-se na capacidade de capturar dependências temporais e representar gestos com alto nível de precisão.

Embora o método mostre desempenho promissor, os autores reconhecem que o modelo atual não contempla aspectos de segmentação temporal dos sinais, o que limita sua aplicação em cenários totalmente contínuos ou em tempo real.

Kashikar *et al.* [23] apresentam um modelo de reconhecimento de gestos manuais baseado em Redes Neurais Convolucionais (do inglês *Convolutional Neural Networks*) (CNNs), voltado para melhorar a comunicação de pessoas com deficiência auditiva. O trabalho utiliza um conjunto de dados próprio, composto por 565 imagens que representam 30 sinais diferentes da linguagem de sinais, incluindo variações de gestos, expressões faciais e movimentos corporais.

Os testes demonstraram que o modelo foi capaz de atingir uma acurácia de 93,6%, mostrando boa capacidade de reconhecer os sinais mesmo diante da complexidade dos gestos e suas variações contextuais e individuais. Os autores destacam que o modelo apresentou bom desempenho tanto no conjunto de treino quanto no de validação, o que indica um bom grau de generalização. Embora o conjunto de dados seja relativamente pequeno, o uso de imagens variadas e bem distribuídas por classe contribuiu para o resultado.

Como destaque, o artigo ressalta o potencial de aplicação em áreas como casas inteligentes, educação inclusiva e dispositivos assistivos. No entanto, a proposta não aborda aspectos de detecção temporal ou segmentação contínua de sinais, concentrando-se exclusivamente em gestos estáticos extraídos de imagens. Apesar disso, a pesquisa representa um avanço relevante ao mostrar a viabilidade de soluções acessíveis para reconhecimento de linguagem de sinais usando técnicas simples de CNNs com bons resultados.

Kurada *et al.* [24] apresentam o FHGR-CNN, um modelo baseado em Redes Neurais Convolucionais (do inglês *Convolutional Neural Networks*) voltado para o reconhecimento conjunto de gestos manuais e expressões faciais em linguagens de sinais, com o objetivo de ampliar a expressividade não verbal e a acessibilidade para pessoas com deficiência auditiva. A proposta se diferencia por integrar imagens de rostos e mãos em um sistema multimodal, com Interface Gráfica do Usuário (do inglês *Graphical User Interface*) (GUI) intuitiva e *feedback* visual e sonoro.

A metodologia proposta envolve um pipeline completo com etapas de aquisição de dados, pré-processamento, treinamento e avaliação. O conjunto de dados foi construído pelos autores e é composto por cerca de 350 imagens divididas em 14 classes, incluindo gestos com duas mãos e expressões faciais como "*Smile*", "*Sad*", "*Angry*", "*Dance*", entre outros. As imagens foram normalizadas para 350×350 pixels e passaram por filtros medianos, ajustes de contraste e balanceamento de cores para melhorar a qualidade visual e facilitar a extração de características.

Durante os testes, o modelo alcançou acurácia de 74,38%, evidenciando boa capacidade de identificar corretamente os gestos relevantes. A GUI desenvolvida permite ao usuário carregar imagens para reconhecimento, visualizar as etapas intermediárias de processamento e receber o resultado classificado em tempo real.

Apesar do desempenho promissor, o modelo ainda apresenta limitações no volume de dados utilizados e na diversidade dos sinais. Como proposta futura, os autores sugerem expandir o vocabulário de sinais, explorar classificações *multilabel* e aplicar aprendizado por conjuntos (*ensemble learning*), além de testar o sistema em cenários mais variados e contínuos.

Senthil Pandi *et al.* [25] propõem um modelo de detecção de linguagem de sinais utilizando redes *Long Short-Term Memory* (LSTM), voltado ao reconhecimento de gestos em ambientes reais, inclusive com variações complexas. A arquitetura foi desenvolvida com foco em sequências temporais, buscando superar limitações de abordagens com CNNs em vídeos, como dificuldade em lidar com variações de movimento e sequência. A proposta foi avaliada utilizando dados extraídos com *MediaPipe Holistic*, que fornece pontos-chave de mãos, rosto e corpo em tempo real.

O modelo proposto realiza a detecção de sinais por meio de um pipeline que compreende: captura de vídeo, extração de *keypoints* com *MediaPipe*, formatação em *arrays NumPy*, segmentação de sequências de 30 *frames*, treinamento com rede LSTM e saída com função *softmax* para classificação. Foram aplicadas técnicas de *dropout* para prevenir *overfitting*, e a divisão dos dados foi feita em 78% para treino e 22% para validação. Além disso, os autores utilizaram validação cruzada com 10 *folds* para garantir maior robustez dos resultados.

O conjunto de dados utilizado contém vídeos com média de 2 segundos, resolução de 1920×1080 e taxa de 28 quadros por segundo. Os dados foram capturados em ambientes naturais, sem o uso de sensores adicionais, luvas ou cenários controlados, aumentando a aplicabilidade prática da solução. A LSTM foi treinada para reconhecer gestos isolados dinâmicos, e a comparação com redes CNN bidimensionais mostrou desempenho significativamente superior: a LSTM alcançou acurácia de 97,8%.

A proposta também integra um módulo para conversão de sinais em linguagem natural e fala, utilizando uma base de dados de imagens de sinais escrita e dicionários visuais. Esse sistema transforma o gesto detectado em uma frase simples em inglês, que por fim é convertida em áudio.

Em termos de inovação, o destaque do artigo está na aplicação do LSTM em um contexto multimodal com extração de pontos-chave e no uso de um fluxo totalmente baseado em vídeo e visão computacional, sem sensores físicos adicionais. Como proposta futura, os autores indicam a expansão do vocabulário, testes em ambientes mais variados

e a aplicação de algoritmos evolutivos e *ensemble* para otimizar a detecção.

Cesar Augusto de Carvalho [26] desenvolveu um sistema portátil para traduzir as letras do alfabeto manual de LIBRAS em voz, utilizando uma luva equipada com sensores inerciais *Inertial Measurement Unit* (IMU). O sistema consiste em cinco sensores acoplados aos dedos, cada um com acelerômetro e giroscópio, conectados a um microcontrolador ESP32 que transmite os dados por Bluetooth para um *smartphone*. O *smartphone* realiza o processamento dos dados e classifica os sinais com algoritmos de aprendizado de máquina.

Três tipos de classificadores foram testados: Perceptron Multicamadas (do inglês *Multilayer Perceptron*) (MLP), *k-Nearest Neighbors* (k-NN) e Redes de Função de Base Radial (do inglês *Radial Basis Function Network*) (RBFN). O RBFN apresentou o melhor desempenho em tempo real, com acurácia de 99,84%, enquanto o MLP chegou a 99,93% de acurácia, porém foi considerado menos eficiente para tempo real por não lidar bem com entradas desconhecidas. O k-NN alcançou 99,69%.

O sistema foi planejado para ser de baixo custo e portátil, possibilitando a tradução da soletração manual para o português falado. O autor sugere que o sistema pode ser ampliado para reconhecer outros sinais além do alfabeto, além de otimizações para tornálo ainda mais eficiente e adaptável a diferentes usuários.

Abdullah Zaiter e Lucas Schiavini [27] desenvolveram um sistema inteligente para reconhecimento de sinais em Língua Brasileira de Sinais (LIBRAS) usando uma luva instrumentada equipada com sensores inerciais. O objetivo foi criar uma ferramenta de baixo custo para traduzir sinais manuais em texto, facilitando a comunicação entre pessoas surdas e ouvintes. A luva possui sensores IMU (acelerômetro e giroscópio) em cada dedo, conectados a um microcontrolador ESP32 que transmite os dados por Bluetooth para um aplicativo móvel desenvolvido em Flutter.

O sistema captura informações de movimento e posição dos dedos, que são processadas por redes neurais recorrentes, especialmente LSTM e *Gated Recurrent Unit* (GRU), treinadas para classificar sinais específicos da LIBRAS. Para o treinamento, os autores criaram um banco de dados próprio contendo registros de 10 sinais básicos, coletados a diferentes frequências de amostragem, o que permitiu analisar o impacto da frequência na acurácia. O melhor desempenho foi obtido com redes LSTM profundas a 300 Hz, alcançando acurácia média de 92,7% para os sinais propostos.

Além da luva e do aplicativo, o projeto inclui um módulo de subamostragem de dados para simular diferentes taxas de captura e otimizar a transmissão entre o hardware e o aplicativo. Um dos diferenciais do trabalho é a análise detalhada da relação entre a taxa de amostragem dos sensores e o desempenho das redes neurais, bem como a integração completa entre hardware, software embarcado e aplicação para *smartphones*.

Embora o sistema não reconheça expressões faciais ou sinais com duas mãos, os autores sugerem como trabalho futuro a ampliação do vocabulário de sinais, a integração de novos sensores e o aperfeiçoamento da arquitetura das redes para aplicações em tempo real.

Lipisha Chaudhary *et al.* [28] apresentam o *SignNet-II*, um modelo baseado em *Transformers* para tradução bidirecional entre linguagem de sinais (em vídeo) e linguagem falada (em texto), ou seja, tanto do vídeo de sinais para texto quanto do texto para sequência de gestos. A proposta busca resolver o problema da falta de modelos capazes de realizar essa tradução completa com precisão, especialmente em sinais contínuos. O diferencial do trabalho é o uso da arquitetura *Transformer* adaptada para sequências multimodais, com mecanismo de atenção e codificação posicional tanto para vídeo quanto para texto.

O sistema é composto por dois módulos principais. Um tradutor de sinais para texto, que usa um codificador *Transformer* para processar os *frames* extraídos do vídeo e um decodificador *Transformer* para gerar a frase correspondente. E um gerador de sinais a partir de texto, que faz o processo inverso: recebe texto como entrada e gera a sequência de poses (*keypoints*) correspondente aos sinais, como se fosse um vídeo de gestos.

Para representar os sinais, os autores usam dados do conjunto *RWTH-PHOENIX-Weather 2014T*, que fornece anotações em vídeo, *glossas* (palavras-chave dos sinais) e transcrições. Cada vídeo é processado com *OpenPose* para extrair os pontos-chave do corpo, e os dados são normalizados e divididos em segmentos temporais. Além disso, o modelo utiliza atenção cruzada entre sinais e texto, permitindo que ele aprenda as relações entre movimentos específicos e palavras correspondentes.

Nos testes, o *SignNet-II* obteve *BLEU-4* score de 21,4 de 100, mostrando desempenho superior tanto em tradução de sinais para texto quanto no caminho reverso. Outro destaque é que o modelo é totalmente baseado em aprendizado de máquina supervisionado, sem necessidade de dicionários manuais ou glossários intermediários.

Como trabalho futuro, os autores planejam adicionar características visuais mais ricas (como aparência facial e expressões) e melhorar o alinhamento semântico entre sinal e texto em contextos mais variados.

Capítulo 3

Metodologia Proposta

O sistema proposto realiza a identificação de uma palavra a partir de um vídeo em que uma pessoa soletra sinais em LIBRAS. Inicialmente, os dados de cada *frame* do vídeo são extraídos por meio da biblioteca *MediaPipe*. Em seguida, essa sequência de dados é estruturada e enviada a um classificador que utiliza rede recorrente, responsável por identificar a letra correspondente em cada *frame*. Por fim, aplica-se um pós-processamento sobre a sequência de letras reconhecidas para reconstruir a palavra representada no vídeo.

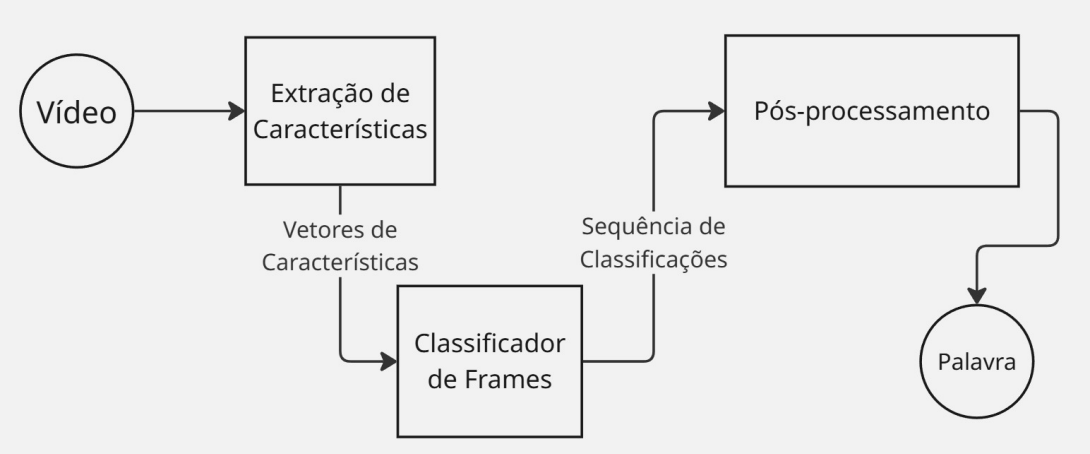

Figura 3.1: Fluxo geral do sistema de reconhecimento de palavras em Libras
Fonte: Próprio autor.

A Figura 3.1 apresenta um panorama das principais etapas que compõem o sistema desenvolvido. Inicialmente, realiza-se a extração de características, na qual as coordenadas 3D espaciais da mão são capturadas quadro a quadro com o auxílio da biblioteca MediaPipe. Em seguida, ocorre a classificação de *frames*, etapa em que um modelo RNN ou LSTM é empregado para associar cada *frame* a uma letra, considerando o contexto temporal. Para melhorar a robustez do reconhecimento, aplica-se uma etapa de pós-processamento composto de 3 etapas: i) filtragem de ruídos, responsável por eliminar

classificações isoladas e identificar corretamente as transições entre letras. Na sequência, a etapa de ii) correção de repetições visa suprimir repetições indesejadas de letras, ajustando a saída de acordo com padrões linguísticos do português. Por fim, a etapa de iii) correção linguística com uma modelo de linguagem LLM refina o resultado, sugerindo a palavra existente mais provável.

Considerando o uso real da LIBRAS no cotidiano da comunidade surda, observa-se que a soletração não é uma prática comum durante a comunicação fluida. Entretanto, ela é frequentemente utilizada em situações específicas[1], como ao expressar nomes próprios, especialmente de pessoas ou lugares. Diante disso, optou-se por focar os testes do projeto exclusivamente em nomes próprios de pessoas e localidades. Essa escolha também influencia diretamente na etapa de pós-processamento, visto que a correção linguística foi adaptada para sugerir palavras válidas dentro desse contexto.

3.1 Extração de Características

Esta etapa consiste na extração dos características referentes aos *frames* do vídeo fornecido ao sistema. Para isso, foi utilizada a biblioteca *MediaPipe*, especificamente o modelo *Hand Landmarks*, que detecta e rastreia pontos-chave da mão em tempo real. Ao processar um *frame* no qual a mão do usuário está visível, o modelo captura 21 pontos de referência (*landmarks*) da mão. Cada ponto fornece três coordenadas espaciais — X, Y e Z — totalizando 63 valores por *frame*. Esses valores são então organizados em um vetor unidimensional que representa a pose da mão naquele instante, servindo como entrada para o classificador.

O sistema foi configurado para considerar apenas uma mão por *frame*, mesmo quando múltiplas mãos são detectadas. Essa restrição garante que a quantidade de dados extraídos permaneça consistente e compatível com o modelo de classificação utilizado. Além disso, caso a biblioteca não consiga identificar nenhuma mão em determinado *frame*, esse quadro é descartado e não gera vetor de características, evitando ruídos na sequência de entrada.

Dessa forma, para cada *frame* válido, é gerado um vetor com a seguinte estrutura:

$$\mathbf{v} = \{X_1, Y_1, Z_1,\ X_2, Y_2, Z_2,\ \ldots,\ X_{20}, Y_{20}, Z_{20},\ X_{21}, Y_{21}, Z_{21}\}$$

Em que X_n, Y_n e Z_n representam as coordenadas espaciais do n-ésimo ponto de referência capturado pelo *MediaPipe*, para $n = 1, 2, \ldots, 21$.

[1] `https://www.handtalk.me/br/blog/alfabeto-em-libras-o-que-e-para-que-serve-e-importancia/`

Ao final do processo de extração, é formada uma sequência temporal de vetores — um para cada *frame* válido — que será utilizada como entrada para o classificador responsável por identificar as letras soletradas no vídeo.

3.2 Classificador de Frames

A sequência de vetores extraídos é então enviada para o classificador de *frames*, responsável por identificar a letra representada em cada instante do vídeo. Esse classificador é baseado em um modelo de Rede Neural Recorrente (do inglês *Recurrent Neural Network*) (RNN), que considera não apenas a entrada atual, mas também o contexto das entradas anteriores para realizar a predição. Neste trabalho, foram realizados testes utilizando tanto a Rede de Elman quanto a LSTM (*Long Short-Term Memory*).

Em ambos os casos, o processamento ocorre da seguinte forma: a sequência completa de vetores gerada a partir do vídeo é dividida em subsequências de comprimento fixo, denominadas *Time Step* (TS). Esse parâmetro define quantos vetores consecutivos serão utilizados em conjunto para que o modelo explore o contexto temporal durante a classificação.

Ou seja, sendo k o valor de TS, cada janela com k vetores sequenciais é passada para o modelo, que utiliza as informações recorrentes dentro dessa janela para gerar a classificação de cada vetor individual. O modelo retorna, então, uma sequência de rótulos correspondente aos vetores processados, indicando a letra que cada *frame* representa.

Caso a quantidade total de *frames* não seja divisível pelo número de TS, é aplicado um *padding* ao final da sequência, adicionando vetores nulos (com todos os valores zerados) para completar a última janela, garantindo que todas as entradas tenham o mesmo comprimento.

Uma representação do funcionamento do classificador pode ser vista na Figura 3.2.

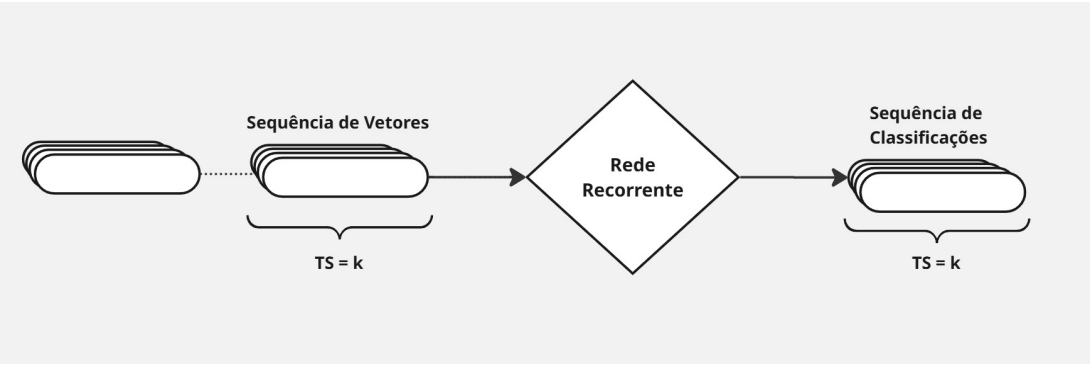

Figura 3.2: Diagrama do Classificador de Frames
Fonte: Próprio autor.

Essa abordagem com janelas de tamanho fixo permite ao modelo aprender padrões temporais locais e torna o treinamento mais eficiente, ao mesmo tempo em que garante compatibilidade com os requisitos de entrada das RNNs, que esperam sequências com tamanhos definidos.

As classificações geradas pelo classificador correspondem às letras do alfabeto latino, com exceção da letra *ç* , além de uma classe especial representada pelo símbolo $*$, utilizada para indicar que nenhum sinal correspondente a uma letra foi identificado naquele *frame*.

3.3 Pós-processamento

A etapa de pós-processamento tem como objetivo reconstruir, a partir da sequência de classificações geradas pelo classificador de *frames*, a palavra soletrada no vídeo. Esse processo é dividido em três partes: Filtragem de Ruídos, Correção de Repetições Excessivas e Correção Linguística com LLM.

3.3.1 Filtragem de Ruídos

Nesta primeira etapa, busca-se eliminar ruídos presentes na sequência bruta de classificações, como a presença do símbolo $*$ (utilizado para representar ausência de letra) e classificações pontuais incorretas, que surgem em *frames* isolados e não refletem uma letra representada de forma consistente ao longo do tempo. Além disso, sequências de letras suficientemente grandes são consideradas como execuções das letras durante o tempo e, portanto, são consideradas no produto final dessa etapa.

A filtragem segue o seguinte raciocínio:

1. Apenas sequências consecutivas de uma mesma letra com comprimento mínimo de L classificações são consideradas válidas. Caso o comprimento seja maior ou igual a L, a sequência é reduzida a apenas uma letra. Caso o contrário, a sequência é então descartada por ser considerada instável.

2. Sequências do símbolo $*$ com comprimento igual ou superior a L são interpretadas como transições entre letras. Caso o comprimento seja menor que L, essa sequência é ignorada, não sendo considerada uma transição entre letras. Essas transições ajudam a delimitar o fim de uma letra e o início de outra.

O L representa o tamanho da sequência mínima necessária para ser considerada uma letra ou uma transição. Tamanhos menores que L são considerados como ruídos e, portanto, serão desconsiderados. Foi determinado, de forma arbitrária, o valor de 4 para L.

A lógica baseia-se na observação de que, ao soletrar em LIBRAS, o usuário tende a manter cada sinal (letra) por um intervalo de tempo. Entre um sinal e outro, ocorre uma transição gestual, que muitas vezes não é reconhecida como nenhuma letra, sendo classificada como $*$. Com isso, a sequência temporal típica apresenta um padrão composto por três segmentos: uma sequência de classificações consistentes para uma letra, uma sequência de transição, e outra sequência consistente para a letra seguinte.

Esse comportamento pode ser observado na Figura 3.3.

Figura 3.3: Transição da letra a para a letra b
Fonte: Próprio autor.

Considerando a Figura 3.3 e assumindo que o classificador cometeu alguns erros pontuais, uma possível sequência de classificações gerada para esse vídeo seria:

$$\mathbf{s} = [\ a,\ a,\ a,\ a,\ *,\ a,\ a,\ *,\ *,\ *,\ *,\ *,\ *,\ *,\ b,\ b,\ b,\ b,\ r,\ b,\ b\ ]$$

Ao aplicar a etapa de filtragem de ruídos, os seguintes ajustes são realizados:

- A letra r, que aparece dentro da sequência de b, e o $*$, que aparece dentro da sequência de a, são ignorados, pois não formam uma sequência suficientemente longa; - A sequência de $*$ é interpretada como uma transição; - As sequências principais de a e b são reduzidas a apenas uma aparição. Como resultado, obtém-se a palavra:

$$\mathbf{r} = ab$$

Essa filtragem é fundamental para eliminar variações incorretas e obter uma sequência mais representativa das letras realmente soletradas.

3.3.2 Correção de Repetições Excessivas

Mesmo após a etapa de filtragem de ruídos, ainda podem ocorrer repetições de letras que não fazem sentido dentro das regras ortográficas da língua portuguesa. O exemplo a seguir

mostra uma sequência que foi classificada a partir da execução da letra b no decorrer do tempo:

$$\mathbf{s} = [\ b,\ b,\ b,\ b,\ *,\ *,\ *,\ *,\ b,\ b,\ b,\ b,\ b,\ *,\ *,\ *,\ *,\ b,\ b,\ b,\ b,\ b]$$

Após a etapa de filtragem de ruídos ficaria:

$$\mathbf{r} = bbb$$

Como houve um ruído maior nessa classificação, acabou que o que deveria ser apenas um b, resultou numa repetição que não corresponde a realidade.

Para definir uma regra de correção adequada, foi considerada a estrutura ortográfica da língua portuguesa. De forma geral, não existem casos válidos de repetição de uma mesma letra por mais de duas vezes consecutivas em palavras do português. Assim, qualquer ocorrência de três ou mais letras idênticas em sequência é, na prática, fruto de erro de reconhecimento ou flutuação na detecção do sinal.

Dessa forma, a regra aplicada nesta etapa limita todas as letras, sem exceção, a no máximo duas ocorrências consecutivas. Assim, qualquer sequência com mais de duas letras repetidas é reduzida automaticamente a uma repetição dupla.

A sequência de três bs vinda da etapa anterior, seria então corrigida para:

$$\mathbf{r} = bb$$

Com essa abordagem, evita-se que ruídos no reconhecimento causem palavras artificialmente prolongadas, garantindo que a saída se mantenha coerente com as possibilidades reais da ortografia da língua portuguesa. Essa etapa contribui para que a sequência final seja mais próxima de uma forma lexicalmente válida, facilitando a etapa seguinte de correção linguística e interpretação da palavra.

3.3.3 Correção Linguística com LLM

Por fim, mesmo após as etapas anteriores de filtragem e correção, a sequência de letras resultante ainda pode conter erros — como letras faltando ou trocadas. Para aumentar a chance de identificar corretamente a palavra soletrada, aplica-se uma etapa de correção linguística utilizando uma LLM.

Nesta etapa, a sequência de letras é enviada a um modelo de linguagem natural, especificamente o *ChatGPT*, com o objetivo de inferir qual palavra da língua portuguesa mais se aproxima da sequência fornecida, mesmo que ela esteja corrompida ou incompleta.

A escolha por utilizar uma LLM como o *ChatGPT* se justifica por sua alta capacidade de compreensão textual e contextualização linguística. Modelos dessa natureza são treinados com grandes volumes de texto e são capazes de realizar correções, sugestões e inferências com base em padrões linguísticos comuns, mesmo quando a entrada contém erros.

O comando utilizado que será apresentado a seguir foi montado tendo em vista que os vídeos para testes foram feitos com nomes próprios de pessoas e de lugares. Este é o comando:

"Considerando a língua portuguesa, que nome próprio válido, de pessoa ou lugar, mais se aproxima da sequência '*s*', que contém letras erradas e repetidas. Na sua resposta, traga somente o nome próprio, dessa forma: nome."

Com esse comando, o modelo tenta encontrar o nome próprio mais provável que corresponda à sequência fornecida, considerando variações e erros plausíveis. Essa etapa final funciona como um refinamento semântico que aumenta a robustez do sistema em situações reais, onde ruídos na classificação podem comprometer a reconstrução exata da palavra, além de eliminar a necessidade do uso de um dicionário previamente escolhido, com todos os nomes próprios existentes.

Capítulo 4

Resultados Obtidos

Neste capítulo, são apresentados os principais procedimentos realizados e os resultados obtidos ao longo do desenvolvimento do sistema proposto, conforme descrito no capítulo de Metodologia. O código-fonte do projeto pode ser acessado no github (link). Os experimentos foram organizados de acordo com as três etapas centrais do sistema: Extração de Dados, Classificação de *Frames* e Pós-processamento.

Na etapa de Extração de Características, serão descritos os procedimentos de gravação dos vídeos utilizados para treinamento e testes, e a análise da classificação manual feita a partir dos *frames* desses vídeos que foi utilizada para justificar a definição do valor do parâmetro *Time Step* (TS).

Na etapa do Classificador de *Frames*, serão apresentados os resultados dos treinamentos realizados com as RNNs de Elman e LSTM, incluindo gráficos de desempenho e análises comparativas entre os modelos.

Por fim, na etapa de Pós-processamento, serão demonstrados os resultados finais do sistema completo, utilizando gravações feitas por três participantes — 2 não fluentes em LIBRAS e um fluente — que soletraram 26 palavras para avaliar a capacidade do sistema em reconstruir corretamente as palavras representadas.

4.1 Extração de Características

4.1.1 Preparação dos Vídeos Utilizados

A etapa inicial do projeto envolveu a gravação dos vídeos que seriam utilizados tanto para o treinamento das redes neurais do classificador quanto para a avaliação do sistema em funcionamento.

Para os vídeos foram utilizadas as câmeras dos celulares pessoais de cada participante. Os vídeos do participante 1 tinham tamanho de 1920×1080 pixeis amostrados a 60fps,

os do participante 2 tinham tamanho 1920×1080 a 30fps e as do participante 3 tinham tamanho 848×480 a 60fps. Não foi feito nenhum tipo de redução de tamanho das imagens para fazer a extração dos dados.

É importante destacar que a dominância manual do indivíduo, ou seja, ser destro ou canhoto, não influencia diretamente o desempenho do sistema. O fator determinante para garantir a consistência dos resultados é que os vídeos utilizados no treinamento e aqueles utilizados nos testes apresentem sinais realizados com a mesma mão. Caso seja necessário utilizar vídeos com a mão oposta, é possível contornar essa variação por meio de um espelhamento da imagem, uma vez que, na LIBRAS, todos os sinais de soletração podem ser interpretados corretamente mesmo quando espelhados, respeitando-se a simetria das configurações manuais.

Foram gravados dois tipos principais de vídeos:

- vídeos com a datilologia completa do alfabeto em LIBRAS;
- vídeos em que o participante soletrava palavras específicas (nomes próprios).

O primeiro tipo de vídeo teve como objetivo fornecer dados para o treinamento das redes neurais recorrentes. A hipótese assumida foi que, mesmo com dados limitados, apenas a repetição dos sinais de cada letra em diferentes vídeos poderia ser suficiente para que o classificador aprendesse padrões consistentes e generalizáveis. Para cada participante, foram gravados pelo menos 10 vídeos completos do alfabeto, totalizando diversas instâncias de cada letra em diferentes condições de tempo e posicionamento da mão. Na Figura 4.1 podem ser vistas imagens de um dos vídeos feitos do alfabeto. Pode ser visto também nos seguintes vídeos onde os participantes 1 (`link`) e 2 (`link`) fazem os sinais do alfabeto em LIBRAS.

O segundo tipo de vídeo foi preparado para a fase de testes do sistema completo. Para isso, foi definida uma lista com 26 palavras, todas sendo nomes próprios de pessoas ou lugares. Cada palavra foi escolhida de forma que sua inicial correspondesse a uma das letras do alfabeto, garantindo assim que todas as letras fossem contempladas nos testes. A lista foi elaborada com atenção à diversidade de fonemas, comprimentos e tipos de nomes, e está apresentada a seguir:

- Amanda
- Beatriz
- Camila
- Dublin
- Eduardo
- Fernando
- Gabriela
- Helsinque
- Itália
- Jacarta
- Kyoto
- Luxemburgo
- Mariana
- Nairobi
- Oslo

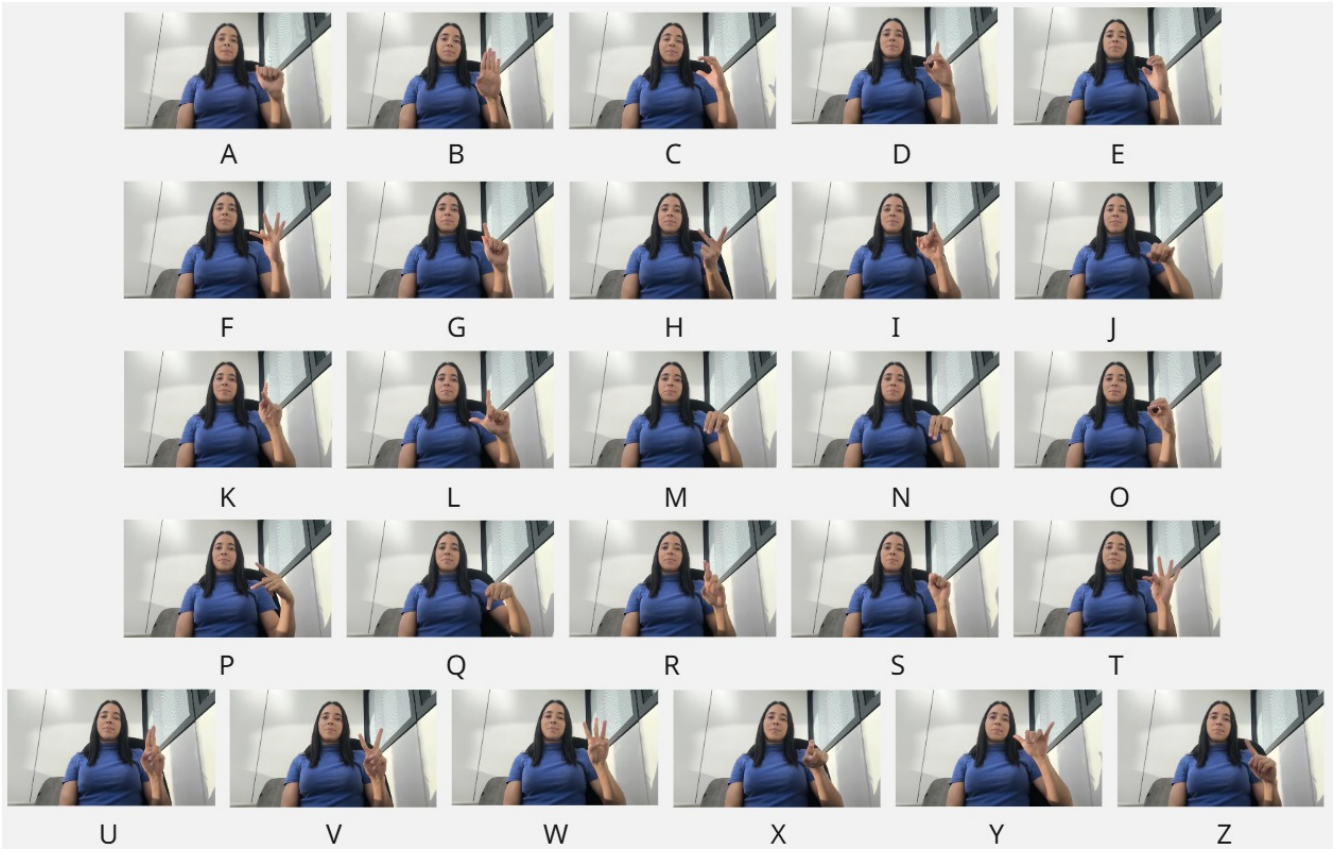

Figura 4.1: Alfabeto feito pela participante 3.
Fonte: Próprio autor.

- Portugal
- Qatar
- Raphael
- Samuel
- Tokyo
- Uruguai
- Vancouver
- William
- Xangai
- Yuri
- Zimbabwe

Durante a gravação, foram adotados alguns cuidados para garantir a qualidade dos dados capturados. Era essencial que apenas uma das mãos do participante estivesse visível no vídeo, pois a biblioteca *MediaPipe* tende a apresentar inconsistências quando múltiplas mãos são detectadas simultaneamente. Além disso, observou-se que o funcionamento do modelo se manteve estável mesmo sob variações de ambiente e iluminação, desde que a mão permanecesse visível e bem delimitada durante toda a execução dos sinais.

Na Figura 4.2 pode ser visto a diversidade de ambientes e iluminações que foram gravados os vídeos de cada participante.

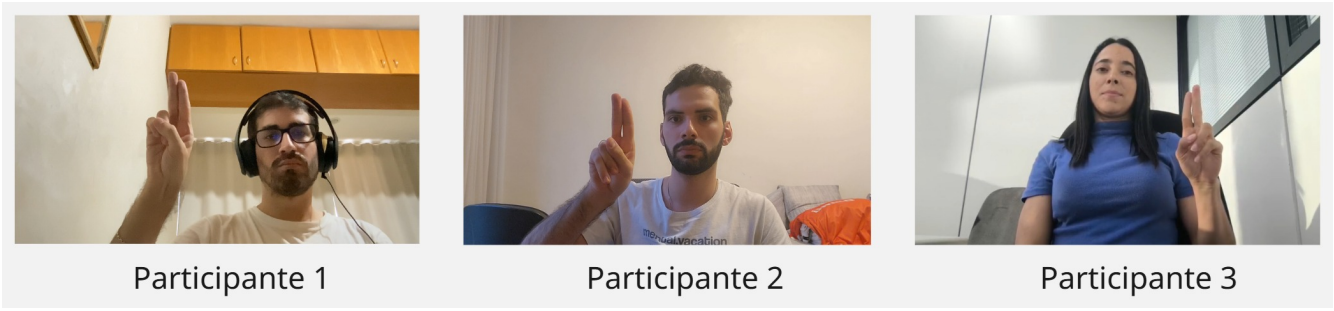

Figura 4.2: Diversidade de ambientes de gravação de cada participante.
Fonte: Próprio autor.

4.1.2 Análise das Classificações Iniciais

Como as RNNs utilizadas são redes supervisionadas, foi necessário realizar a rotulagem manual de todos os *frames* dos vídeos de treinamento. Para cada vídeo, foi gerado um vetor de rótulos com comprimento igual à quantidade de *frames* em que uma mão foi detectada pela biblioteca *MediaPipe*.

Esses vetores representam a sequência ideal de classificações que o classificador deve aprender. A partir disso, foi feita uma análise para verificar quantos *frames*, em média, eram utilizados por cada participante para representar cada letra do alfabeto em LIBRAS, bem como o valor mínimo e máximo. Essa análise permite entender a duração temporal dos gestos e orientar a configuração dos parâmetros da rede.

Os histogramas a seguir (Figura 4.3, Figura 4.4, Figura 4.5) mostram a distribuição da quantidade de *frames* utilizados para representar cada letra por três participantes, participantes 1 e 2 que não são fluentes em LIBRAS e o participante 3 que é fluente.

Com base nesses dados, observou-se que a menor duração registrada entre todas as letras foram de 24, 16 e 13 *frames*, para os participantes 1, 2 e 3, respectivamente. Ou seja, as letras com menores tempos de execução de cada participante exigiu pelo menos 24, 16 e 13 *frames*.

Dessa forma, adotou-se a seguinte estratégia para a escolha do valor de *Time Step* (TS): para cada participante será atribuído o valor que corresponde a metade do valor de 24, 16 e 13. Foi feito assim, partindo da hipótese de que, ao dividir a janela de execução mínima de uma letra pela metade, a rede teria acesso suficiente ao contexto gestual para identificar corretamente a letra representada. Essa escolha permite que a rede processe partes sobrepostas do gesto completo, aprendendo padrões temporais relevantes mesmo em execuções rápidas.

Fazendo dessa forma, busca-se equilibrar o uso de contexto temporal com a generalização dos padrões de movimento, otimizando a performance da rede na tarefa de classificação de letras em sequência.

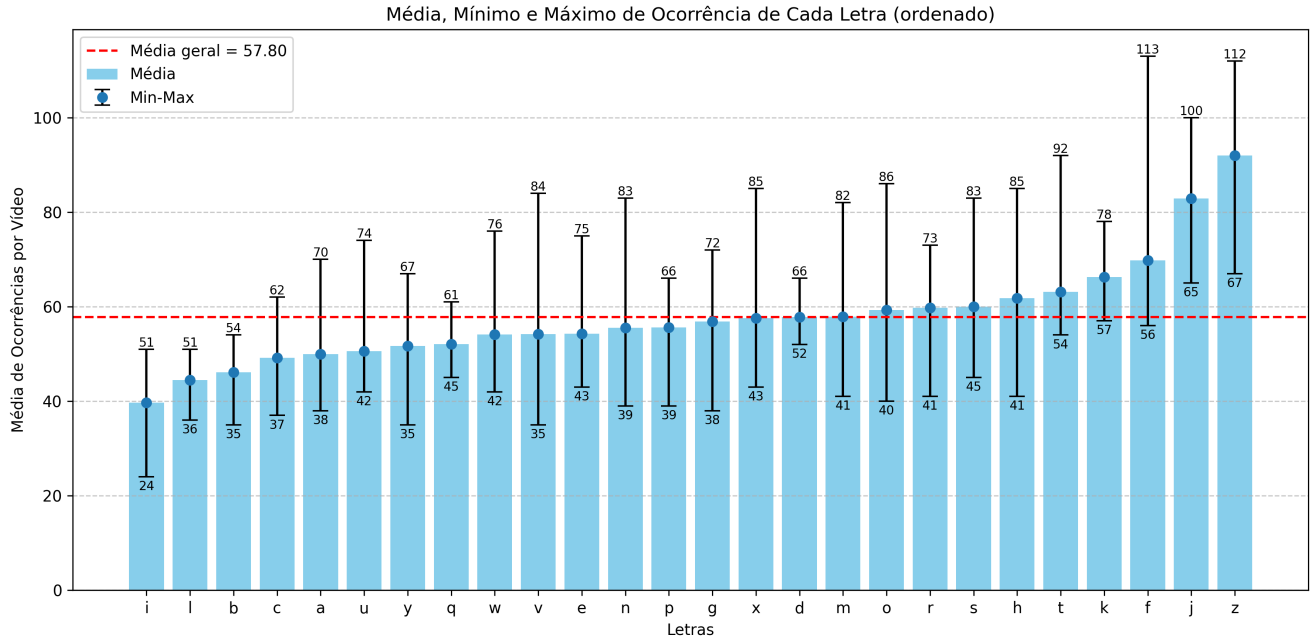

Figura 4.3: Distribuição da quantidade de *frames* utilizados para representar cada letra do alfabeto - Participante 1

Fonte: Próprio autor.

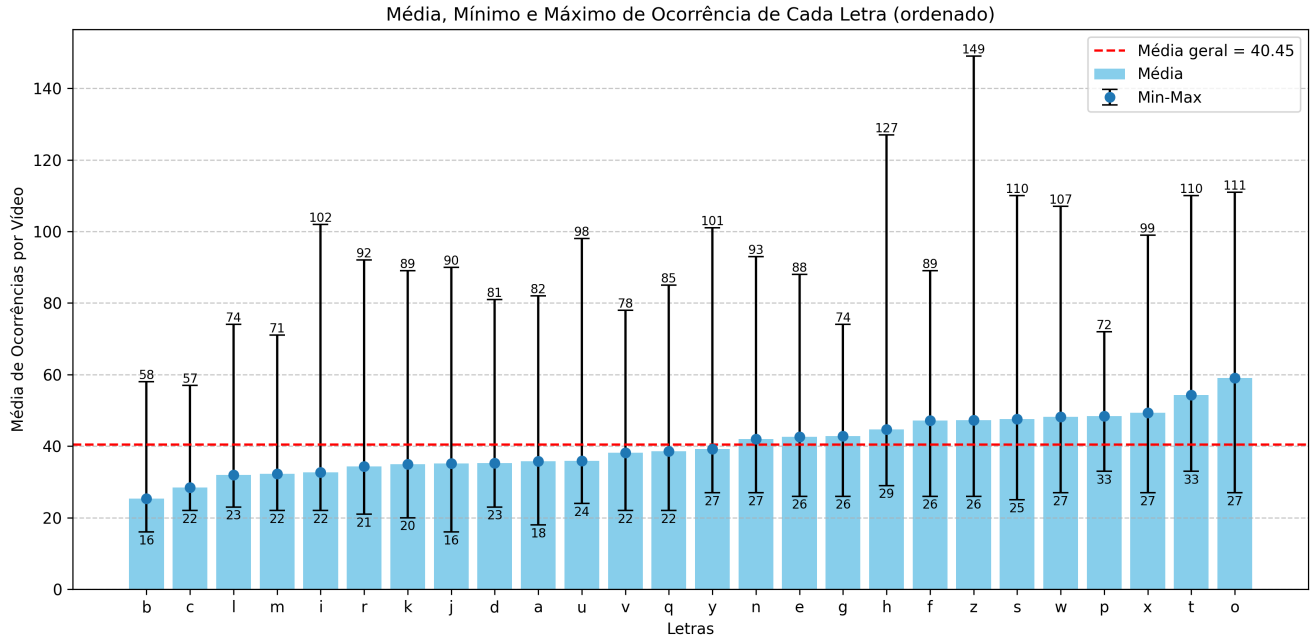

Figura 4.4: Distribuição da quantidade de *frames* utilizados para representar cada letra do alfabeto - Participante 2

Fonte: Próprio autor.

Com isso, os valores de TS determinados para cada participante foi, **12** (participante 1), **8** (participante 2) e **6** (participante 3).

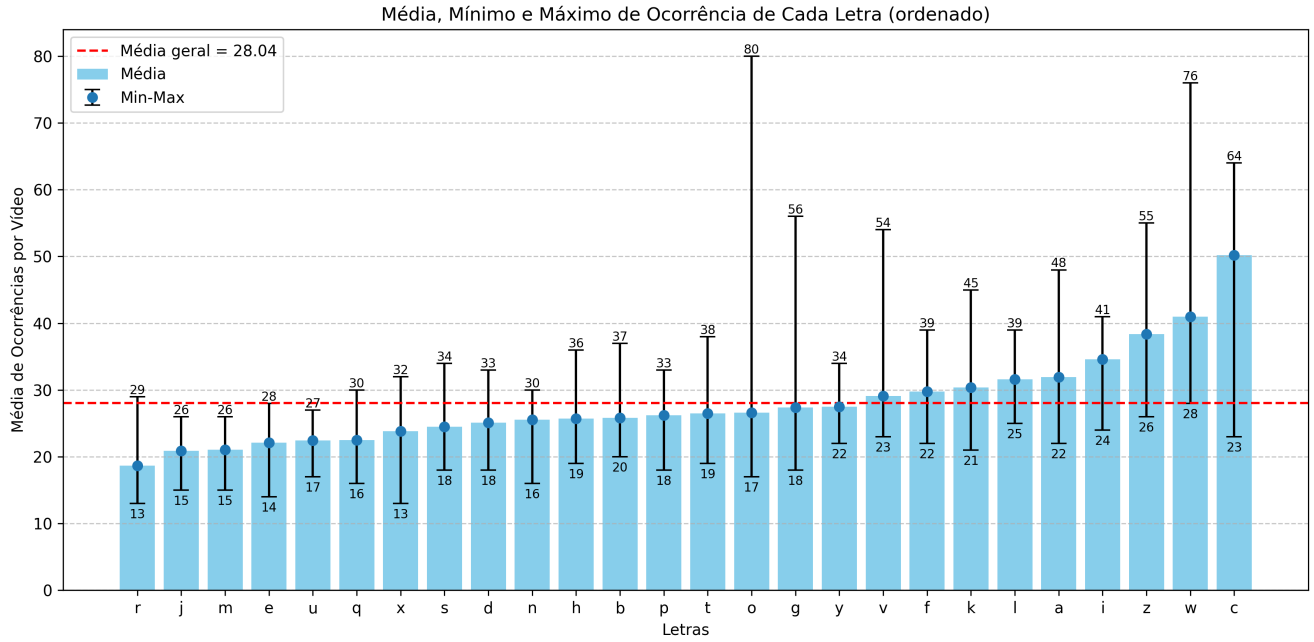

Figura 4.5: Distribuição da quantidade de *frames* utilizados para representar cada letra do alfabeto - Participante 3

Fonte: Próprio autor.

A definição do valor da variável *Time Step* foi feita neste trabalho com base em características específicas de um único participante, considerando a média de duração dos sinais por vídeo. No entanto, ao generalizar o treinamento da rede de Elman para incluir dados de múltiplos usuários, que possuem diferentes velocidades, estilos e variações de execução, torna-se necessário adotar métodos mais robustos para essa definição. Uma alternativa seria utilizar a mediana ou percentil de duração dos sinais em toda a base de dados, de modo a minimizar o impacto de *outliers* e variações extremas.

4.2 Classificador de Frames

Nesta seção são descritos os procedimentos de treinamento realizados para definir os principais parâmetros da versão final do classificador de *frames*. Entre esses parâmetros estão o número de neurônios da camada recorrente e o tipo de rede (Elman ou LSTM). Além disso, são apresentados resultados quantitativos de desempenho que embasaram essas decisões.

Para todos os treinamentos feitos, os dados de entrada utilizados foram os vetores extraídos dos vídeos do alfabeto gravados pelos participantes. Esses vetores foram divididos em conjuntos de treino, validação e teste, de forma a garantir avaliações consistentes dos modelos.

O tipo de dado utilizado nos vetores extraídos das *frames*, tanto dos vídeos de treinamento quanto dos vídeos de teste, foi o `multi_hand_world_landmarks` da biblioteca *MediaPipe Hands*. Essa escolha foi definida com base em testes exploratórios realizados com imagens estáticas, utilizando algoritmos como k-NN e um MLP de camada única para reconhecer letras, comparando especificamente o uso do `multi_hand_landmarks` e o `multi_hand_world_landmarks`. Os resultados mostraram melhor desempenho na identificação das letras quando se utilizou o `multi_hand_world_landmarks`, justificando sua adoção nesta pesquisa.

4.2.1 Treinamento dos Modelos

Os treinamentos das RNNs seguiram os seguintes critérios:

- Inicialmente, adotou-se o valor de *Time Step* (TS) igual a 10 em todos os testes preliminares, uma vez que o ajuste desse parâmetro não era o foco neste estágio;
- Cada rodada de treinamento consistia em apenas uma *epoch* por vez. Após cada *epoch*, o modelo era validado com o conjunto de validação para monitorar o valor de *loss*. Se o *loss* diminuísse em relação à rodada anterior, o modelo era salvo. Caso contrário, o treinamento prosseguia até completar 1000 *epochs* consecutivas sem nova redução do *loss*;
- Para garantir uma comparação justa, o mesmo particionamento de dados foi usado em todos os experimentos;
- Os únicos parâmetros variáveis entre os experimentos foram: o tipo de rede (Elman ou LSTM), o número de neurônios na camada recorrente e a origem dos dados (participante 1 ou 2).

4.2.2 Resultados Quantitativos

As Tabela 4.1 e a Tabela 4.2 apresentam os resultados de cada treinamento. Cada linha corresponde a um modelo treinado, indicando o tipo de rede, o número de neurônios, o menor *loss* obtido durante a validação e a acurácia alcançada no conjunto de teste. A Tabela 4.1 mostra os resultados para o participante 1, enquanto a Tabela 4.2 mostra os do participante 2.

A partir da análise das acurácias apresentadas na tabela, definiu-se o tipo de rede e a quantidade de neurônios mais adequados para o sistema. Olhando para as acurácias das redes de Elman, para o participante 1 (Tabela 4.1) a maior acurácia foi alcançada com 100 neurônios atingindo 0,8582. Já para o participante 2 (Tabela 4.2), o melhor resultado também foi obtido com 100 neurônios, registrando uma acurácia de 0,9244.

Tabela 4.1: Resultados dos Treinamentos dos Modelos - Participante 1.

	Elman		**LSTM**	
Nº Neurônios	**Menor *loss***	**Acurácia Teste**	**Menor *loss***	**Acurácia Teste**
10	0,5738	0,8195	0,5283	0,7897
25	0,4547	0,8525	0,4921	0,8128
50	0,4782	0,8207	0,3851	0,8224
100	0,4041	**0,8582**	0,3561	0,8352
150	0,4404	0,8489	0,3717	0,7879
200	0,4254	0,8426	0,3031	0,8215
300	0,4374	0,8296	0,3127	0,8280
400	0,4545	0,8310	0,2799	**0,8647**
500	0,4466	0,8323	0,3295	0,8377
750	0,4204	0,8389	0,2596	0,8568
1000	0,4589	0,8327	0,3213	0,8327

Tabela 4.2: Resultados dos Treinamentos dos Modelos - Participante 2.

	Elman		**LSTM**	
Nº Neurônios	**Menor *loss***	**Acurácia Teste**	**Menor *loss***	**Acurácia Teste**
10	0,4838	0,9029	0,4554	**0,9021**
25	0,3284	0,9101	0,5629	0,8826
50	0,3339	0,9109	0,4623	0,8779
100	0,2980	**0,9244**	0,3576	0,8677
150	0,2852	0,9168	0,3665	0,8626
200	0,3169	0,9181	0,3931	0,8664
300	0,3245	0,9000	0,4306	0,8719
400	0,3357	0,9000	0,4728	0,8638
500	0,3691	0,8882	0,4406	0,8928
750	0,3807	0,8761	0,4501	0,8783
1000	0,3699	0,8735	0,5196	0,8540

Já para as redes LSTM, a melhor rede do participante 1 foi de 400 neurônios com 0,8647 de acurácia. E pro participante 2 foi a rede de 10 neurônios com acurácia de 0,9021.

4.2.3 Definição dos Parâmetros Finais

Observando a Tabela 4.3, verifica-se que, em média, as redes de Elman superaram as redes LSTM em acurácia para ambos os participantes. Assim, optou-se por utilizar a arquitetura de Elman como base do classificador final.

Outro fator que reforça essa escolha é que o problema em questão envolve um conjunto de dados pequeno, proveniente de apenas um participante por vez. Nesse contexto restrito, redes mais simples, como a Elman, têm maior capacidade de generalização do que LSTMs

Tabela 4.3: Médias calculadas das acurácias das redes obtidas.

Dados	Tipo Rede	Média Acurácias
Participante 1	Elman	0,8327
Participante 1	LSTM	0,8280
Participante 2	Elman	0,9029
Participante 2	LSTM	0,8719

mais complexas, que podem superajustar facilmente aos dados de treino, prejudicando o desempenho em novos exemplos.

Além disso, ao analisar as tabelas (Tabela 4.1 e Tabela 4.2), nota-se que, para ambas as bases, os modelos de Elman com 100 neurônios alcançaram as maiores acurácias, consolidando a escolha desse valor para a rede final.

4.2.4 Validação dos Modelos Finais

Portanto, os principais hiperparâmetros definidos foram: TS de cada participante sendo igual a 12, 8 e 6 (como justificado na seção anterior), rede do tipo Elman e 100 neurônios na camada recorrente. Com esses parâmetros, foram conduzidos novos treinamentos usando validação cruzada *k-fold*, resultando em três modelos finais: um para o participante 1 (acurácia de 88,47%), outro para o participante 2 (acurácia de 90,93%), e por fim, um para o participante 3 (acurácia de 88,34%).

Os modelos foram treinados em dois computadores com configurações distintas. O modelo do participante 1 foi treinado no PC 1, equipado com processador Ryzen 5 3500U com 4 cores e 2,1GHz de frequência de clock, placa de vídeo NVIDIA GeForce GTX 1050 Mobile, 16 GiB de memória RAM, SSD de 512 GiB e sistema operacional Windows 11, apresentando um tempo médio de compilação de aproximadamente 16 minutos e 43 segundos. Já os modelos dos participantes 2 e 3 foram treinados no PC 2, com processador AMD Ryzen 7 3750H com 4 cores e 2,3GHz de frequência de clock, NVIDIA GeForce GTX 1650, 8 GiB de RAM, SSD de 128 GiB mais HD de 1 TiB, também com Windows 11. O tempo médio de treinamento para o participante 2 foi de 14 minutos e 50 segundos, enquanto para o participante 3 foi maior, chegando a cerca de 20 minutos e 20 segundos, devido à maior quantidade de dados utilizados no treinamento (17 vídeos de cada letra do alfabeto, contra 10 vídeos para os participantes 1 e 2).

Para evidenciar que o treinamento atingiu o ponto de generalização ideal, todos os valores de *loss* ao longo das *epochs* foram registrados, incluindo as rodadas adicionais. A tendência de aumento do *loss* após atingir o mínimo demonstra que o modelo ultrapassou o ponto ótimo, caracterizando o início do sobreajuste. Assim, o ponto mínimo reflete o

melhor equilíbrio entre ajuste e generalização. Os gráficos da Figura 4.6, da Figura 4.7 e da Figura 4.8 ilustram esse comportamento do *loss* para os modelos finais dos participantes.

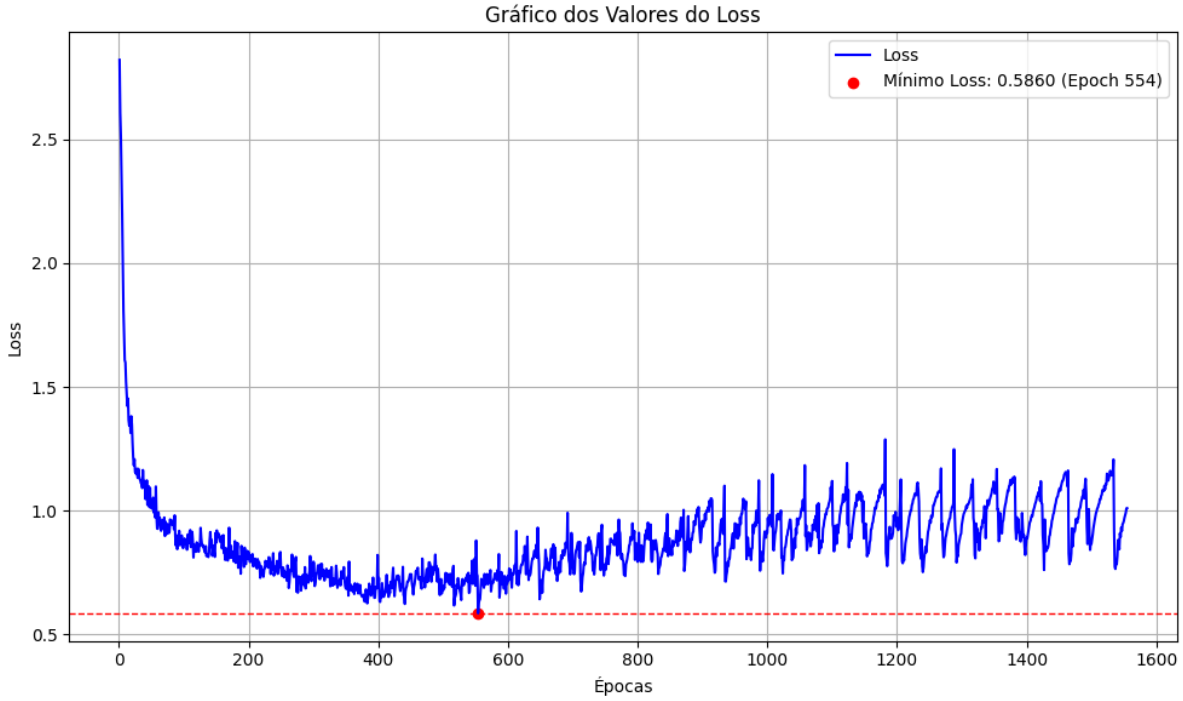

Figura 4.6: Gráfico de treinamento da Rede de Elman do Participante 1
Fonte: Próprio autor.

Por fim, foram geradas matrizes de confusão para os modelos finais, utilizando o conjunto de teste. Essas matrizes (Figura 4.9, Figura 4.11 e Figura 4.13) permitem analisar detalhadamente os acertos e erros de classificação, evidenciando quais letras são mais propensas a confusões.

Avaliando a matriz do participante 1, Figura 4.9, observa-se que, de forma mais significativa, a letra j foi confundida com a letra i. Na Figura 4.10 observa-se que o início da execução da letra j é bem parecido com a própria letra i, justificando essa confusão.

Na rede do participante 2, Figura 4.11, nota-se mais expressamente que as letras r são confundidas com u e as letras u são confundidas com v. Essas confusões podem ser melhor visualizadas pelas semelhanças dos sinais na Figura 4.12.

Por fim, na matriz do participante 3 (fluente em LIBRAS), Figura 4.13, percebeu-se uma maior confusão entre várias letras. A hipótese disso é que, como fluente, o participante fez mais variações na execução dos sinais, o que fez a rede se confundir mais do que as redes dos outros participantes que, por não terem tanta prática, variavam menos na execução dos sinais.

Além desses casos, verificou-se que em todos os participantes houve um número considerável de *frames* de letras confundidas com transições ($*$) e vice-versa. Essa ocorrência

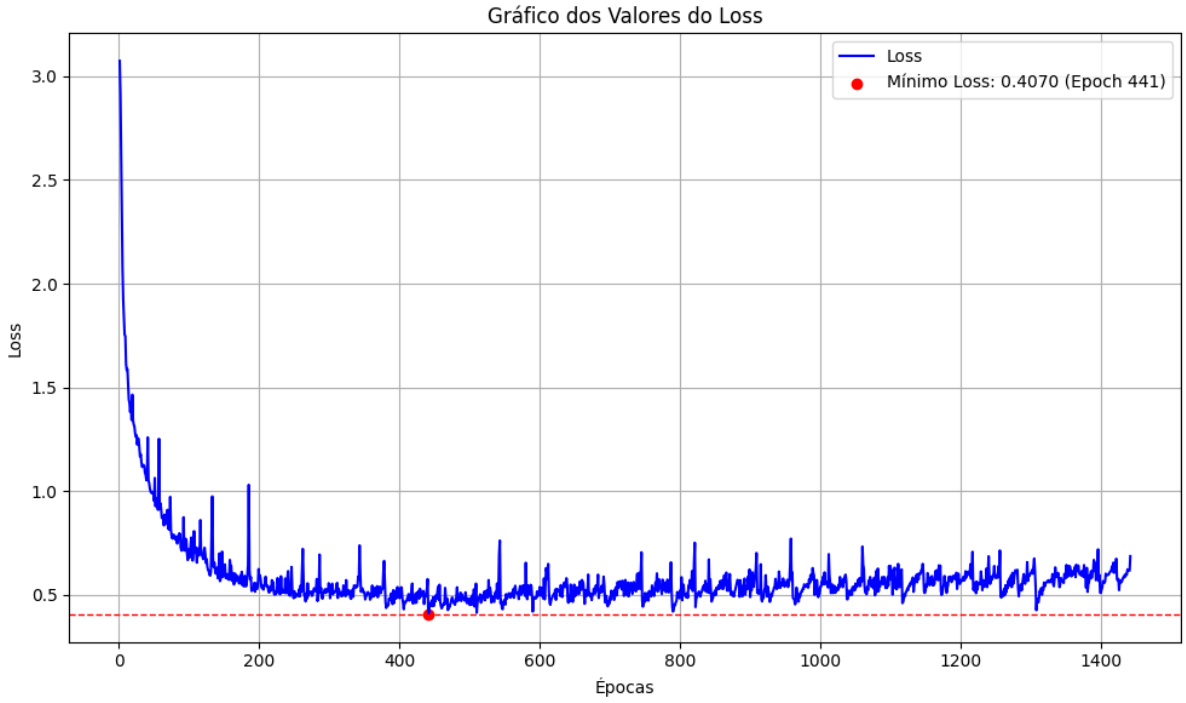

Figura 4.7: Gráfico de treinamento da Rede de Elman do Participante 2
Fonte: Próprio autor.

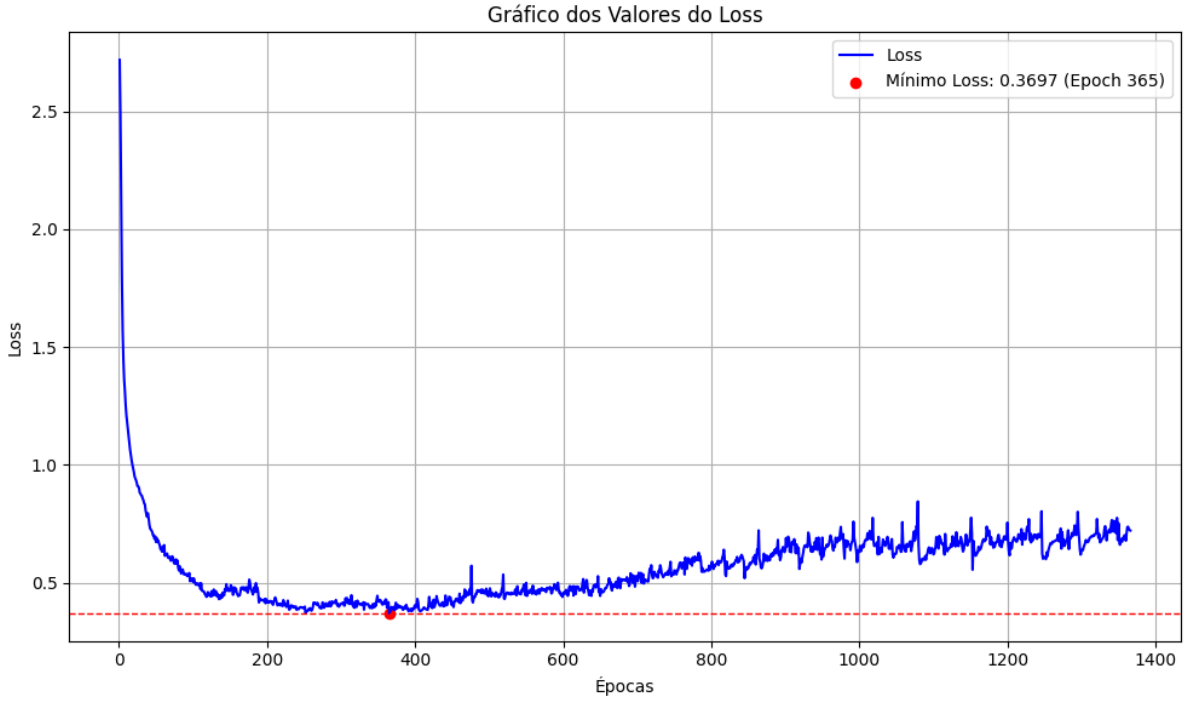

Figura 4.8: Gráfico de treinamento da Rede de Elman do Participante 3 - fluente
Fonte: Próprio autor.

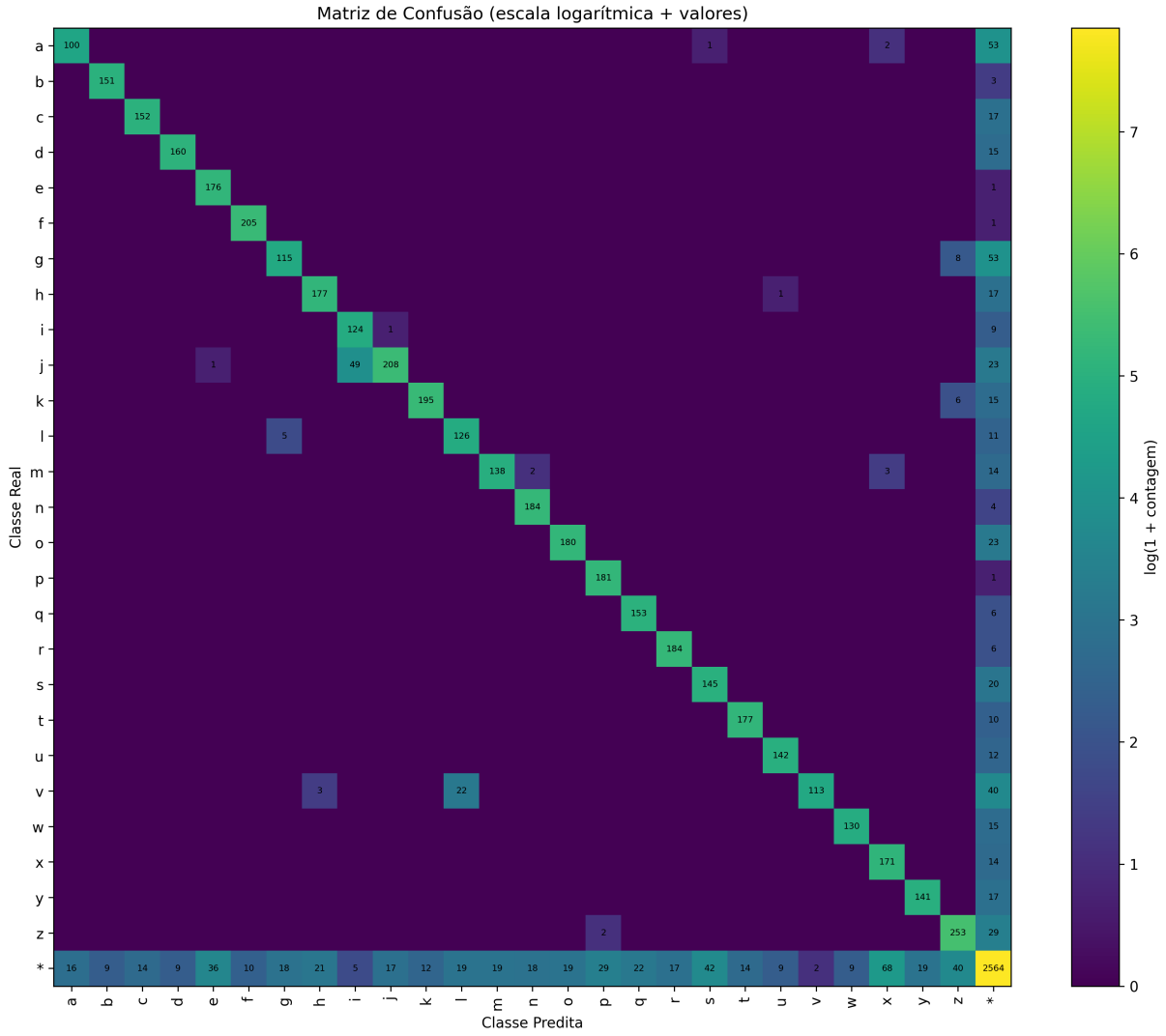

Figura 4.9: Matriz de Confusão do modelo do Participante 1
Fonte: Próprio autor.

é plausível, pois nos pontos de transição entre letra e $*$ (ou de $*$ para letra) existe uma indefinição natural, dado que a segmentação é baseada em janelas temporais, o que pode gerar sobreposição entre os estados.

4.3 Pós-processamento

Nesta seção são apresentados os resultados finais do sistema, já integrando todas as etapas de pós-processamento. Os testes foram realizados a partir dos vídeos gravados pelos participantes 1 e 2, utilizando a lista de palavras específicas apresentada na seção de preparação de dados.

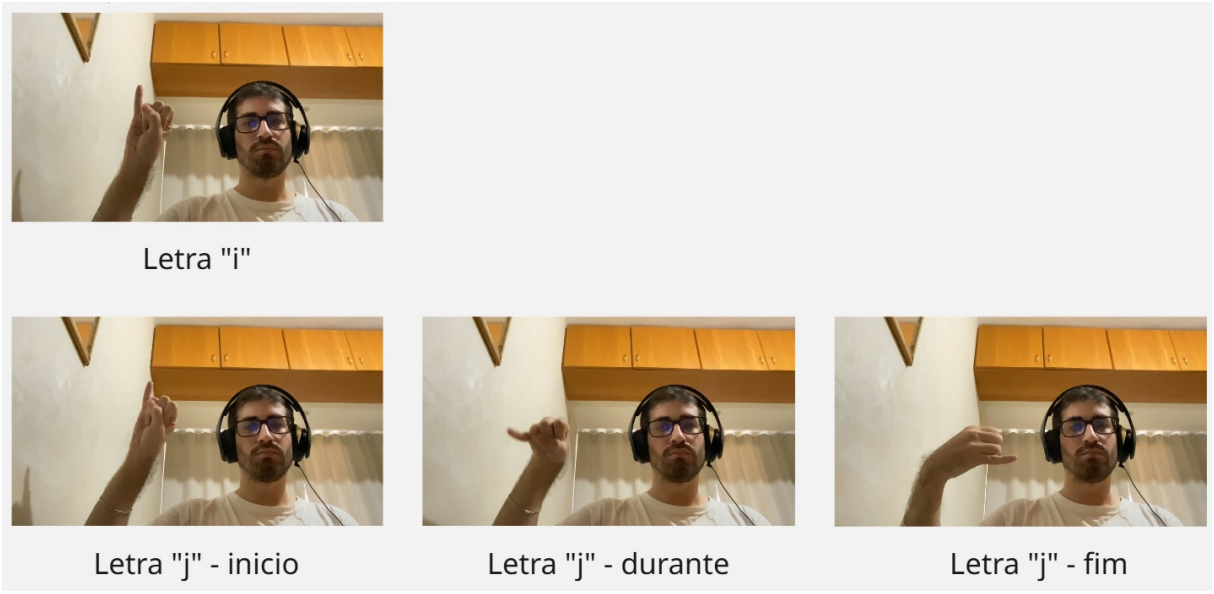

Figura 4.10: Execução das letras *i* e *j* do Participante 1.
Fonte: Próprio autor.

4.3.1 Resultados Finais com Usuários

Após a aplicação de todas as etapas de pós-processamento nos vetores de letras previstos pela rede, os resultados foram repassados para modelos de linguagem de grande porte (LLMs), com o objetivo de inferir qual palavra havia sido soletrada em cada vídeo. Para essa etapa, foram testados três modelos distintos: *LLaMA 3*, *Gemini* e *ChatGPT*.

O modelo *LLaMA 3* foi executado localmente, mas essa abordagem apresentou limitações práticas significativas, especialmente relacionadas ao consumo de memória *RAM*, que é elevado em IAs locais. Como consequência, foi necessário optar por uma versão mais leve do modelo, com um número reduzido de parâmetros, o que acabou comprometendo a precisão das respostas obtidas. Em seguida, foi testado o *Gemini*, utilizando a API gratuita disponibilizada pelo Google, especificamente com o modelo *Gemini 2.5 Flash*, que é otimizado para velocidade e custo, mas não representa o modelo mais avançado da plataforma. Os resultados obtidos com o *Gemini* foram medianos, sem grandes destaques em termos de precisão.

Por fim, o *ChatGPT* foi testado na sua versão mais recente, por meio do acesso gratuito diário disponibilizado pela plataforma. Este modelo apresentou desempenho superior em relação aos anteriores, fornecendo respostas significativamente mais precisas e promissoras na tarefa de identificação das palavras soletradas. Esses resultados indicam que, apesar das limitações observadas em modelos locais ou gratuitos, o uso de LLMs mais avançados e atualizados pode ser um aliado importante no aprimoramento do sistema de tradução desenvolvido.

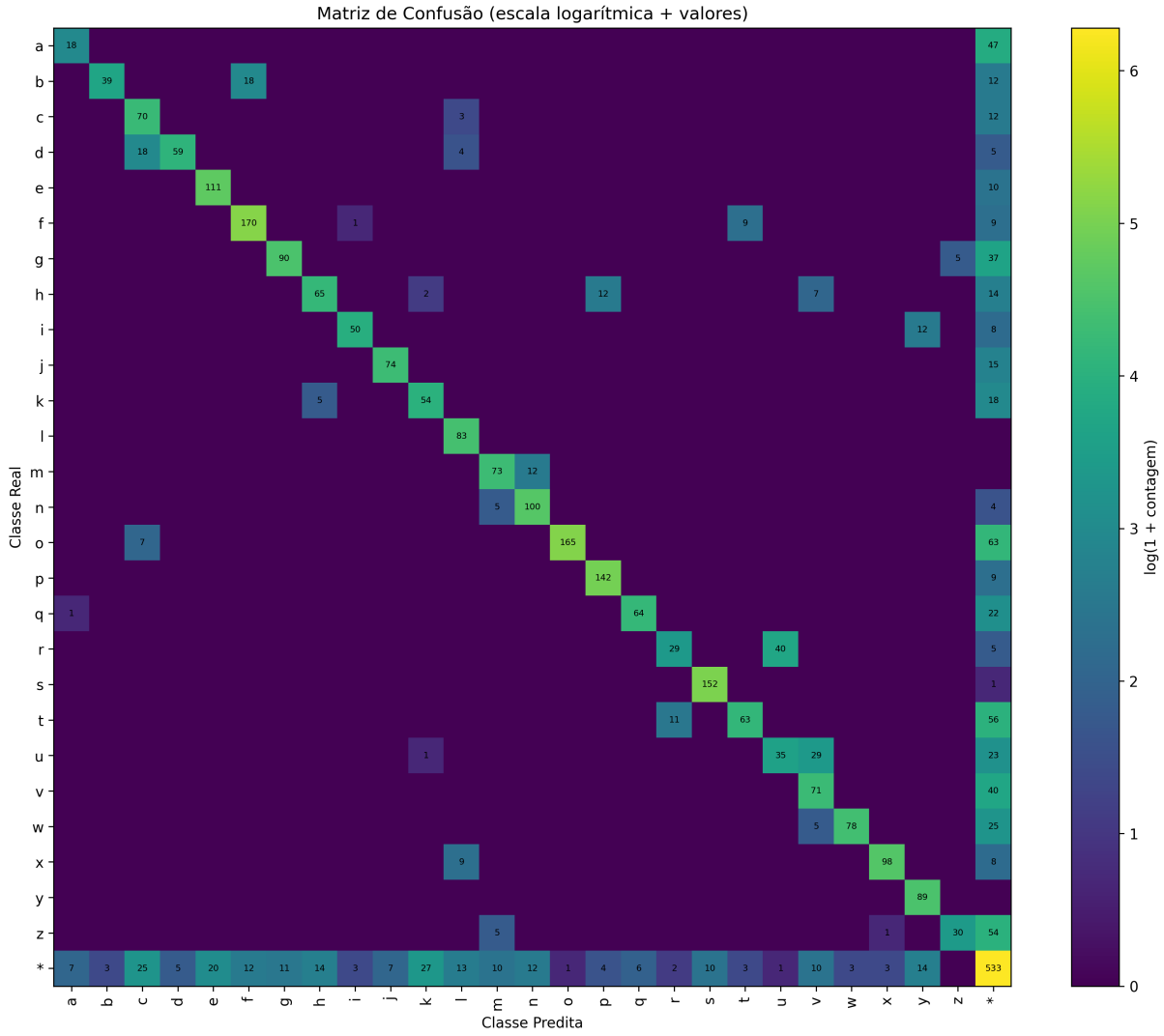

Figura 4.11: Matriz de Confusão do modelo do Participante 2
Fonte: Próprio autor.

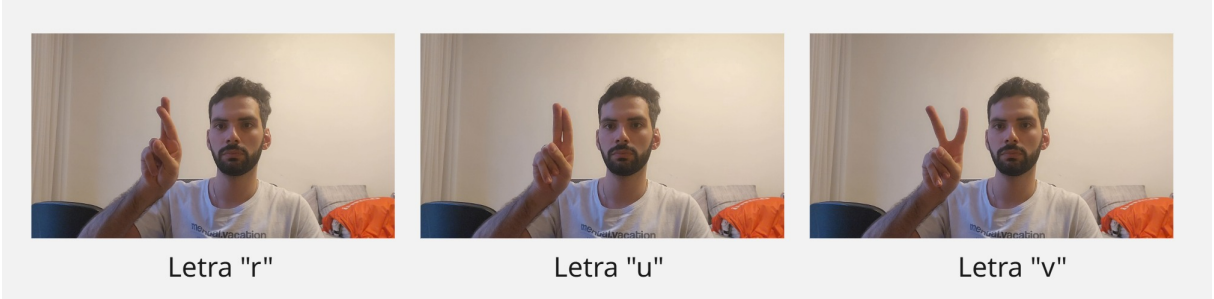

Figura 4.12: Execução das letras r, u e v do Participante 2.
Fonte: Próprio autor.

Mostrando a execução do pós-processamento completo utilizando um teste real, tem-se

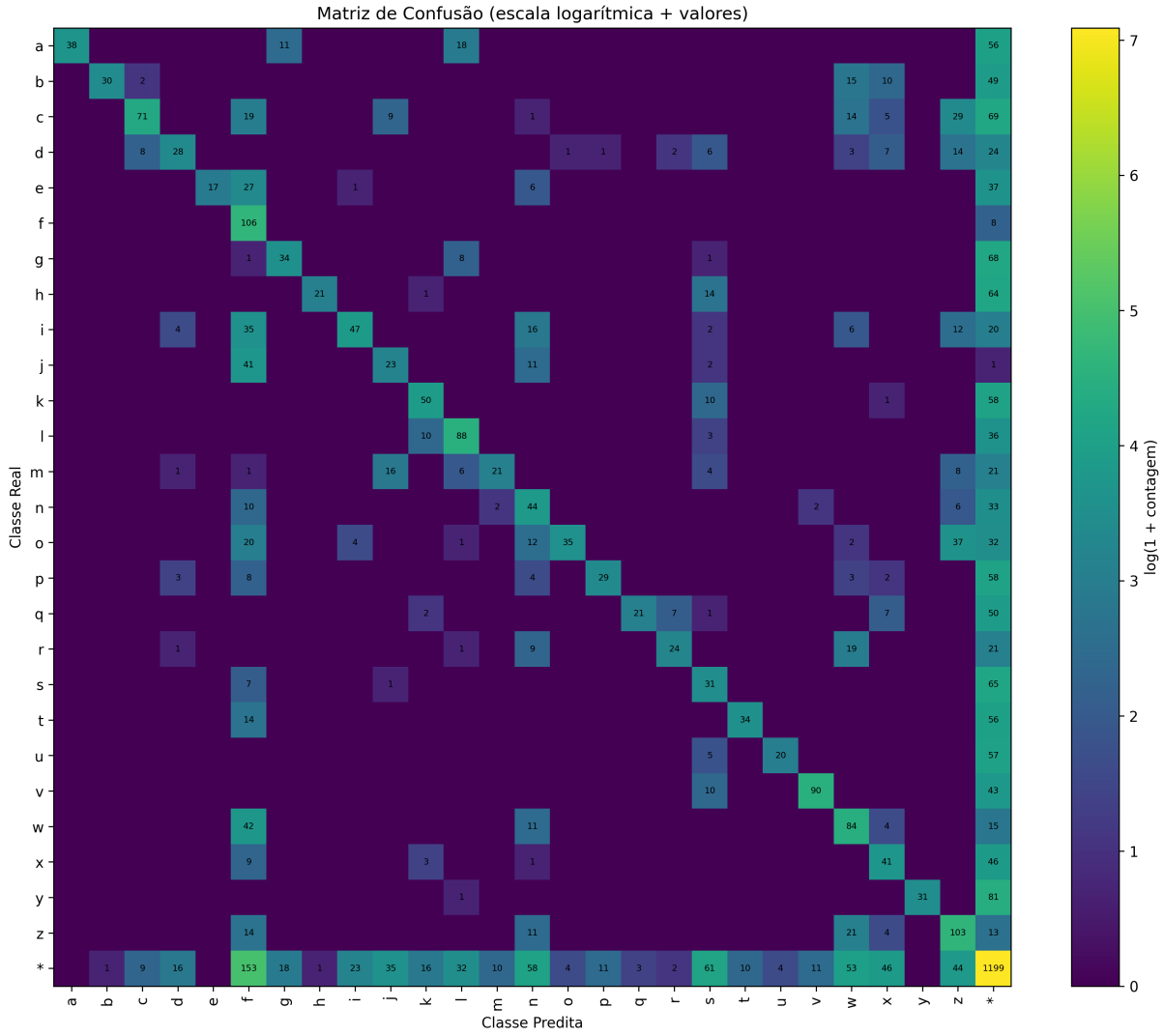

Figura 4.13: Matriz de Confusão do modelo do Participante 3 - Fluente
Fonte: Próprio autor.

a soletração da palavra “luxemburgo” feita pelo participante 2 (`vídeo`). Após a classificação das *frames* obteve-se a seguinte sequência:

`"*z**gggg*lllllll*lllll*******hhuuuuuuuu*****xxxxzzzzzxxxxxxxxxx***`
`****eeeeeeee*e**********mmmmmmmmmmmm*********bbbbbbbwww**********uuuu`
`uuuuuuuuuuuurrrr*uuurrrruuuu**********zzzz**zggggg**ggggg*ooooooooooo`
`ooooooooooo".`

Passando pelas etapas de filtragem e correção de erros, a sequência resultante foi “glluxzxemburruzggo”. Por fim, após a etapa de correção com a LLM, obteve-se “luxemburgo”.

As Tabela 4.4, Tabela 4.5 e Tabela 4.6 mostram os resultados obtidos para cada participante, comparando a forma da palavra gerada pelo sistema imediatamente após as

etapas de Filtragem de Ruídos e Correção de Repetições Excessivas (sem uso de LLM) com o resultado final após a aplicação da etapa de correção linguística, que utilizou o *ChatGPT* como modelo de linguagem.

Em termos de acurácia, a rede do participante 1 obteve 84,61% de acerto das palavras, considerando que "italy", "rafael" e "toquio" são acertos. A rede do participante 2 obteve 76,92% de acerto, considerando "tóquio" como acerto. E a rede do participante 3 obteve 61,53% de acerto. Essas considerações foram feitas pois a LLM acertou semanticamente as palavras, apesar de não ter sido literal.

Esses resultados evidenciam como o pós-processamento, em especial a integração com uma LLM, foi essencial para transformar sequências de letras ruidosas em palavras reais que se aproximam da forma esperada. Observa-se que, em muitos casos, mesmo quando a saída inicial apresentava caracteres adicionais, duplicações ou substituições, a LLM conseguiu inferir a palavra correta ou uma variação próxima.

Tabela 4.4: Resultados Finais - Participante 1.

Palavra Esperada	**Sem correção com LLM**	**Após correção com LLM**
Amanda	axmaaxnnxdaa	**amanda**
Beatriz	beaatfttururururuiizkzz	**beatriz**
Camila	caxmxjil	**camila**
Dublin	zdubljjxmm	dublim
Eduardo	edaaurrdcoo	**eduardo**
Fernando	nnjaanxddo	**fernando**
Gabriela	gaabrrielaa	**gabriela**
Helsinque	hheelsinqxrre	helenice
Itália	ytaaly	italy
Jacarta	jcaaurtaa	**jacarta**
Kyoto	**kyoto**	**kyoto**
Luxemburgo	luuxeemtbbuvrrdzglco	**luxemburgo**
Mariana	mmaarrjaaxnaa	**mariana**
Nairobi	naaiijrobiji	**nairobi**
Oslo	cossglo	coslo
Portugal	pcorrtvuggal	**portugal**
Qatar	qaataaurrur	**qatar**
Raphael	urraazphael	rafael
Samuel	ssaamxrrel	**samuel**
Tokyo	ftozkyo	toquio
Uruguai	uuruuggkuaaii	**uruguai**
Vancouver	vvaanccoruuveur	**vancouver**
William	jwjllsijmm	**william**
Xangai	xaanlgaaii	angola
Yuri	yrurj	**yuri**
Zimbabwe	zxjmxtbbaabwe	**zimbabwe**

Tabela 4.5: Resultados Finais - Participante 2.

Palavra Esperada	**Sem correção com LLM**	**Após correção com LLM**
Amanda	ammandda	**Amanda**
Beatriz	batriissz	**Beatriz**
Camila	ocamnssla	**Camila**
Dublin	dvbbliin	**Dublin**
Eduardo	edvvarruudo	**Eduardo**
Fernando	ferrnamndo	**Fernando**
Gabriela	gabusiiela	**Gabriela**
Helsinque	vhhelsiisinqe	**Helsinque**
Itália	**italia**	**Itália**
Jacarta	ijacarta	**jacarta**
Kyoto	khkyofto	**Kyoto**
Luxemburgo	glluxzxemburruzggo	**luxemburgo**
Mariana	mariiaana	**mariana**
Nairobi	mnairobii	marilbio
Oslo	oislo	Isalo
Portugal	pourtuvgal	**portugal**
Qatar	natar	natal
Raphael	**raphael**	**raphael**
Samuel	sammel	**samuel**
Tokyo	okyo	tóquio
Uruguai	vvruvvggxuvvaia	uruguaia
Vancouver	vamncouvuver	**vancouver**
William	wiyliamm	**william**
Xangai	zzxanzzai	Xanaçaí
Yuri	iyuui	Iuri
Zimbabwe	zaimmebabwe	**zimbabwe**

A partir desses testes, nota-se que o sistema, aliado ao pós-processamento linguístico, demonstra potencial para interpretar palavras datilografadas em LIBRAS mesmo quando o classificador de *frames* comete pequenas falhas de reconhecimento.

Também fica evidente a importância das etapas de filtragem e correção de repetições para reduzir o ruído antes de passar o texto para a LLM, tornando a tarefa de correção mais viável e precisa.

Os exemplos apresentados indicam que, apesar de algumas saídas não coincidirem exatamente com a palavra esperada, o contexto fornecido pela sequência de sinais permite que o modelo de linguagem recupere a intenção do usuário na maioria dos casos, validando a estratégia proposta para a etapa de pós-processamento.

Aplicação do sistema em tempo real mostrou-se viável. Tirando a etapa de correção com LLM, que é a etapa mais lenta do processo, as classificações de frames, a filtragem de ruídos e a correção de repetições são etapas que podem ser executadas em tempo real.

Tabela 4.6: Resultados Finais - Participante 3.

Palavra Esperada	**Sem correção com LLM**	**Após correção com LLM**
Amanda	aadda	**Amanda**
Beatriz	baatrizz	**Beatriz**
Camila	aamil	Amil
Dublin	ddblliin	**Dublin**
Eduardo	edaarrdoo	**Eduardo**
Fernando	ftfranndoo	**Fernando**
Gabriela	ggbrioeya	**Gabriela**
Helsinque	khhllzzizn	Kilzinho
Itália	izzglii	Iguazu
Jacarta	jjartta	Júlia
Kyoto	kyffc	Kyffhäuser
Luxemburgo	llxmmwbrro	alximarorro
Mariana	maarriyanaa	**mariana**
Nairobi	nirrcobi	nicóbio
Oslo	ooslo	**oslo**
Portugal	portuul	**portugal**
Qatar	qqataar	**qatar**
Raphael	raapkhaal	**Raphael**
Samuel	aammk	Amank
Tokyo	tkyoo	**tokyo**
Uruguai	uurrgguuaaii	**uruguai**
Vancouver	vaoccuuvvrr	**Vancouver**
William	wiliyymm	**william**
Xangai	xanni	janni
Yuri	yyrri	**Yuri**
Zimbabwe	zzimmbbww	ximbabu

Um aspecto importante para essa aplicação é que, para identificar a transição entre uma palavra e outra, seria preciso que o usuário retirasse a mão do campo de visão da câmera. Seguindo esse procedimento, torna-se possível implementar uma versão do sistema que opere em tempo real.

Por fim, os resultados obtidos evidenciam que este sistema representa uma contribuição para a área de reconhecimento e tradução de datilologia em LIBRAS para o português escrito, demonstrando que a integração de redes recorrentes, filtragem de ruído, correção ortográfica e modelos de linguagem pode contornar limitações individuais de cada etapa. Assim, o trabalho comprova a viabilidade técnica dessa abordagem, podendo auxiliar em estudos futuros na pesquisa de soluções voltadas à tradução automática de sinais.

Esses resultados, embora inferiores aos alcançados por propostas mais sofisticadas do estado da arte, como as de Pandi *et al.* [25] e Zaiter e Schiavini [27], demonstram um desempenho satisfatório considerando a simplicidade da abordagem adotada, a ausência

de sensores físicos e a utilização de dados reais com variação individual.

Pandi *et al.* [25] propuseram um sistema baseado em redes LSTM treinadas com dados extraídos via *MediaPipe Holistic*, utilizando pontos-chave do corpo, mãos e rosto, e alcançaram 97,8% de acurácia. O modelo deles, voltado para reconhecimento de gestos em ambientes reais, conta com segmentação temporal precisa, validação cruzada com 10 *folds*, e um pipeline completo de reconhecimento e conversão dos sinais para texto e fala. Nesse trabalho, são utilizados gestos completos e que possuem uma dependência temporal maior, por isso o uso de LSTM, obtendo assim resultados melhores que os encontrados neste trabalho, que utilizou menos dados e trabalhou com dependências temporais menores.

Já Zaiter e Schiavini (2020) desenvolveram um sistema com uma luva instrumentada equipada com sensores inerciais (IMUs), treinando redes LSTM e GRU sobre dados captados diretamente do movimento dos dedos. O sistema alcançou 92,7% de acurácia em um conjunto restrito de sinais e com coleta altamente controlada. Apesar do alto desempenho, a solução deles depende de hardware dedicado, coleta com sensores físicos e não contempla sinais com duas mãos ou expressões faciais. Em contrapartida, o sistema proposto neste trabalho utiliza apenas vídeo como fonte de dados e uma arquitetura recorrente mais simples.

Essas comparações destacam que, embora o desempenho obtido neste trabalho seja numericamente inferior, ele foi alcançado com uma estrutura técnica mais simples, dados mais variados e sem o uso de sensores adicionais. Isso reforça a viabilidade da abordagem baseada em rede de Elman com pós-processamento por LLM, especialmente em cenários de recursos limitados ou para fins educacionais.

Com o avanço contínuo das LLMs, é plausível imaginar que, no futuro, essas arquiteturas sejam capazes de realizar, de forma integrada e autônoma, o reconhecimento e a interpretação de sinais em LIBRAS diretamente a partir de vídeos. Embora esse cenário ainda dependa de avanços substanciais em áreas como aprendizado multimodal, captura de movimento e representação semântica de sinais, os resultados obtidos neste trabalho indicam que arquiteturas mais simples, combinadas com estratégias adequadas de pré e pós-processamento, ainda possuem relevância prática e podem atuar como soluções viáveis enquanto tais sistemas integrados não estiverem amplamente disponíveis.

Capítulo 5

Conclusão

Este trabalho teve como objetivo desenvolver um sistema de tradução da soletração em LIBRAS utilizando redes neurais recorrentes, com foco específico na arquitetura de Elman. O sistema foi implementado na linguagem de programação *Python*, com apoio das bibliotecas OpenCV, *MediaPipe* e *TensorFlow*, que foram fundamentais para o processamento de vídeo, a extração dos pontos de referência da mão e o treinamento dos modelos de aprendizado de máquina, respectivamente.

Foram conduzidos três treinamentos independentes, um para cada participante, sendo que todos gravaram diversos vídeos em que executavam integralmente o alfabeto manual em LIBRAS. Esses vídeos foram utilizados para treinar, separadamente, três modelos baseados na rede de Elman, implementados com o *TensorFlow*. Após o processo de treinamento, cada participante gravou um novo conjunto de vídeos, desta vez soletrando palavras completas. Esses vídeos serviram como base para os testes. Em cada *frame*, o sistema classificava o gesto correspondente a uma letra do alfabeto ou atribuía o símbolo especial de asterisco, utilizado para marcar a transição entre letras. As sequências de letras reconhecidas eram então processadas para gerar uma *string* mais limpa e legível, que era finalmente passada como entrada para uma LLM, responsável por inferir a palavra correta com base na sequência fornecida.

Os resultados obtidos foram promissores: a LLM conseguiu obter uma acurácia de 84,61% para o primeiro participante, 76,92% para o segundo e 61,53% para o terceiro, que é fluente em LIBRAS. Apesar dos resultados, é importante destacar que o sistema foi treinado exclusivamente com dados de uma única pessoa e validado com vídeos gravados por essa mesma pessoa. Isso limita a capacidade de generalização do modelo, tornando-o ainda inadequado para uso prático em ambientes diversos ou com diferentes usuários.

5.1 Trabalhos Futuros

Como possíveis desdobramentos deste trabalho, destaca-se, em primeiro lugar, a necessidade de generalização do modelo. Para isso, recomenda-se a coleta de um conjunto de dados mais amplo e diversificado, contemplando participantes distintos — preferencialmente pessoas com alto grau de fluência em LIBRAS — a fim de garantir uma representação mais fiel da variedade de estilos e formas de execução dos sinais. Com um volume maior de dados e maior variabilidade, torna-se viável a adoção de arquiteturas mais sofisticadas, como redes do tipo LSTM, que são capazes de capturar dependências temporais mais longas e complexas, possibilitando o reconhecimento mais robusto de sequências gestuais realizadas por diferentes usuários.

Além disso, uma direção promissora consiste em expandir o escopo do sistema para além da soletração manual, abordando sinais correspondentes a palavras completas, que são muito mais utilizados na comunicação cotidiana da comunidade surda. Essa transição aumentaria consideravelmente o potencial de aplicabilidade prática da ferramenta desenvolvida. Um caminho natural seria iniciar com sinais simples, compostos apenas por movimentos das mãos, e, progressivamente, incluir sinais mais complexos que envolvam outras partes do corpo, como cabeça, tronco e expressões faciais, elementos fundamentais para a expressividade e gramática da LIBRAS.

Referências

[1] Censo de 2019 de indicadores de saúde em território nacional. 2019. 1

[2] Almir Cristiano. O que é libras? `https://www.libras.com.br/o-que-e-libras`, May 2017. Online; Atualizado em 19/03/2020. Acesso em 01/07/2025. 4

[3] Robert Ruben. Sign language: Its history and contribution to the understanding of the biological nature of language. *Acta oto-laryngologica*, 125:464–7, 06 2005. 4, 5

[4] Daniel Neves. Língua brasileira de sinais (libras). https://mundoeducacao.uol.com.br/educacao/lingua-brasileira-de-sinais-libras.htm, s.d. Online; Acesso: 01-07-2025. 5

[5] Agência Senado. Obrigatoriedade da oferta de libras na educação básica passa na cdh. https://www12.senado.leg.br/noticias/materias/2021/08/30/obrigatoriedade-da-oferta-de-libras-na-educacao-basica-passa-na-cdh, August 2021. Online; Acesso: 01-07-2025. 6

[6] Almir Cristiano. Os cinco parâmetros da libras. `https://www.libras.com.br/os-cinco-parametros-da-libras`, 2018. Online; Atualizado em 19 mar. 2020; Acesso: 01-07-2025. 6

[7] Instituto Nacional de Educação de Surdos. Alfabeto manual e configuração de mãos, 2024. Acessado em: 30 jun. 2025. 7

[8] Marco Armenta e Pierre-Marc Jodoin. The representation theory of neural networks. *Mathematics*, 9(24), 2021. 8, 10

[9] Ian Goodfellow, Yoshua Bengio, e Aaron Courville. *Deep Learning*. MIT Press, 2016. 9

[10] Simon Haykin. *Redes neurais: princípios e práticas*. Bookman, 2 edition, 2001. 10, 11, 12

[11] Oscar Gabriel Filho. *Inteligência artificial e aprendizagem de máquina: aspectos teóricos e aplicações*. Blucher, 1 edition, 2023. 11

[12] Jeffrey L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990. 12, 14

[13] Sepp Hochreiter e Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997. 15, 16

[14] Equipe Didática Tech. Como funciona a lstm – long short term memory. https://didatica.tech/lstm-long-short-term-memory/, 2021. Online; Acesso em 03-07-2025. 16

[15] Camillo Lugaresi, Jiuqiang Tang, Minh Nash, Chris McGuire, Won Lee, Chuo-Ling Chang, Michael Yong, Jean-Christophe Hudon, Vignesh Mazzocchi, John Grigsby, et al. Mediapipe: A framework for building perception pipelines. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2278–2286. ACM, 2019. 17

[16] Gary Bradski. The opencv library, 2000. Oline; Acesso em 01-07-2025. 17

[17] MediaPipe. Mediapipe hands – high-fidelity hand and finger tracking solution. https://mediapipe.readthedocs.io/en/latest/solutions/hands.html. Online; Acesso: 01-07-2025. 19, 20

[18] Fabian Pedregosa, Gael Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, Edouard Duchesnay, e Gilles Louppe. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12, 10 2011. 20

[19] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016. 21

[20] Marcus Vinicius Lamar, Md Shoaib Bhuiyan, e Akira Iwata. Hand gesture recognition using t-combnet: A new neural network model. *IEICE transactions on information and systems*, 83(11):1986–1995, 2000. 25

[21] Md Shoaib Bhuiyan, MV Lamar, e Akira Iwata. Hand gesture recognition using morphological principal component analysis and an improved combnet-ii. In *IEEE SMC'99 Conference Proceedings. 1999 IEEE International Conference on Systems, Man, and Cybernetics (Cat. No. 99CH37028)*. IEEE, 2003. 25

[22] Rakesh Kumar, Sarvesh Kumar Dwivedi, M. R. Maurya, e Amrit K. Sinha. Sign language recognition and translation using self-attention long-short-term memory with shape autotuning activation function. In *2024 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS)*, pages 1–6. IEEE, 2024. 25

[23] Vaibhav Kashikar, Pranjali Raut, Sakshi Bhusari, Supriya Jadhav, e Prof. Smita Kulkarni. Hand gesture recognition using improved convolutional neural networks for specially abled. In *2024 International Conference on Smart Technologies and Systems for Next Generation Computing (ICSTSN)*, pages 1–5. IEEE, 2024. 26

[24] Ramachandra Rao Kurada, Vaishalini Vunnam, Reshma Vasanthawada, Lakshmi Saranya Thondapu, N. Silpa, e V.V.R. Maheswara Rao. Face and hand gesture recognition for sign languages to support non-verbal expressions using convolutional neural

network. In *2024 10th International Conference on Advanced Computing and Communication Systems (ICACCS)*, pages 1140–1146. IEEE, 2024. 26

[25] Senthil Pandi S, Kumar P, Sutha J, e Shanmugam P. A novel approach for sign language detection using learning algorithms. In *2024 International Conference on Computational Intelligence for Green and Sustainable Technologies (ICCIGST)*, pages 1–6. IEEE, 2024. 27, 54, 55

[26] Cesar Augusto de Carvalho. Um sistema portátil de tradução de posturas do alfabeto manual de libras em voz utilizando luva instrumentalizada com sensores imu. Monografia de Graduação, Universidade de Brasília, 2019. 102 páginas. 28

[27] Abdullah Zaiter e Lucas dos Santos Schiavini. Sistema inteligente para captura e interpretação de dados temporais da língua brasileira de sinais a partir de uma luva instrumentada. Trabalho de Conclusão de Curso, Universidade de Brasília, 2020. 102 páginas. 28, 54

[28] Lipisha Chaudhary, Tejaswini Ananthanarayana, Enjamamul Hoq, e Ifeoma Nwogu. Signnet ii: A transformer-based two-way sign language translation model. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(11):12896–12907, 2023. 29