REDES NEURAIS TRANSFORMERS E OUTROS MÉTODOS DE DEEP LEARNING APLICADOS NA IDENTIFICAÇÃO PRÉVIA DE TRIP DE UM REATOR NUCLEAR PWR E PREVISÃO DE SEQUÊNCIA DE EVENTOS EM SITUAÇÃO DE TRANSIENTE OPERACIONAL

Bernardo Mendonça Caixeta

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia Nuclear, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia Nuclear.

Orientador: Alan Miranda Monteiro de Lima

Rio de Janeiro

Março de 2025

REDES NEURAIS TRANSFORMERS E OUTROS MÉTODOS DE DEEP LEARNING APLICADOS NA IDENTIFICAÇÃO PRÉVIA DE TRIP DE UM REATOR NUCLEAR PWR E PREVISÃO DE SEQUÊNCIA DE EVENTOS EM SITUAÇÃO DE TRANSIENTE OPERACIONAL

Bernardo Mendonça Caixeta

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA NUCLEAR.

Orientador: Alan Miranda Monteiro de Lima

Aprovada por: Prof. Alan Miranda Monteiro de Lima
Prof. Roberto Schirru
Prof. Victor Henrique Cabral Pinheiro

RIO DE JANEIRO, RJ – BRASIL
MARÇO DE 2025

Caixeta, Bernardo Mendonça

Redes Neurais Transformers e outros Métodos de Deep Learning Aplicados na Identificação Prévia de Trip de um Reator Nuclear PWR e Previsão de Sequência de Eventos em Situação de Transiente Operacional / Bernardo Mendonça Caixeta. – Rio de Janeiro: UFRJ/COPPE, 2025

XIV, 213 p.: il.; 29,7cm

Orientador: Alan Miranda Monteiro de Lima

Dissertação (Mestrado) – UFRJ/COPPE/Programa de Engenharia Nuclear, 2025.

Referência Bibliográfica: p. 200-213

1. Deep Learning. 2. Redes Neurais Transformers. 3. Identificação de Trip. 4. Sequência de Eventos. 5. Reator Nuclear PWR. I. Lima, Alan Miranda Monteiro de II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia Nuclear. III. Título.

“I, a universe of atoms, an atom in the universe.”

Richard Feynman

AGRADECIMENTOS

Agradeço aos meus pais, Maria do Carmo e Marcos Antônio, por todo o carinho e por me apoiar a continuar perseguindo meus sonhos e desejando sempre o melhor para mim, por me incentivar a buscar uma graduação em Engenharia e estar hoje finalizando um Mestrado, por zelar pela minha educação e desenvolvimento pessoal e profissional, por prezar tanto pela minha formação acadêmica e por acreditar no meu potencial. Vocês são a minha inspiração e o principal motivo por eu estar onde estou hoje.

A todo o pessoal do servidor Pesadelo no Discord, em especial aos meus grandes amigos Davi, Dudi, Enzo, João, Lucas, Roberto, Gabriel e Capitaz. Seguindo o meu agradecimento do TCC, por incrível que pareça os anos se passaram e ainda continuamos fechando squad no R6.

Ao meu grande amigo, professor e orientador, professor Alan Miranda, por todos os ensinamentos, risadas, oportunidades e por todo o auxílio no desenvolvimento no conteúdo desta dissertação. Sua paciência e dedicação foram essenciais para a minha formação, e suas orientações, sugestões, críticas e conhecimentos fornecidos foram fundamentais para a realização deste trabalho e para o meu desenvolvimento acadêmico, profissional e pessoal.

Ao professor Roberto Schirru, por todas as oportunidades, ensinamentos, críticas, risadas, orientação, por acreditar no meu potencial e por sempre buscar o melhor para mim. Agradeço o comprometimento, incentivo e suporte constante, que ultrapassaram os limites deste trabalho, mas foram valiosos durante a minha trajetória acadêmica.

A toda a galera do LMP, por todo o suporte, parceria, sugestões e risadas. Agradeço pelas contribuições para o desenvolvimento deste trabalho, por todos os projetos e produções acadêmicas que realizamos juntos e por todo o apoio nessa jornada.

Aos meus grandes amigos que conheci no início da graduação Cauê, Alexandre e Giovanna. Agradeço por todos os momentos que passamos juntos, pelas conversas e risadas nos corredores do CT, por fazerem parte dessa caminhada e por terem sido fundamentais para meu desenvolvimento acadêmico e pessoal.

Aos membros da banca examinadora que estiveram à disposição de avaliar esse trabalho. Agradeço por aceitarem o convite e pelas contribuições.

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

REDES NEURAIS TRANSFORMERS E OUTROS MÉTODOS DE DEEP LEARNING APLICADOS NA IDENTIFICAÇÃO PRÉVIA DE TRIP DE UM REATOR NUCLEAR PWR E PREVISÃO DE SEQUÊNCIA DE EVENTOS EM SITUAÇÃO DE TRANSIENTE OPERACIONAL

Bernardo Mendonça Caixeta

Março/2025

Orientador: Alan Miranda Monteiro de Lima

Programa: Engenharia Nuclear

Dentro do escopo da operação da Central Nuclear Angra 2, uma usina nuclear do tipo PWR, uma das áreas mais críticas para os especialistas envolve a análise da Sequência de Eventos, que corresponde à aquisição e o acompanhamento das variações nos valores de variáveis binárias, exibindo em ordem cronológica as ações tomadas pelos operadores durante a operação. Durante os ciclos de operação, as mudanças no estado dessas variáveis geralmente ocorrem em sequências previstas e controladas pelos operadores. Essas alterações são detectadas, aquiridas, datadas e ordenadas pelo Sistema de Processamento de Alarmes (SPAL), abrangendo os eventos de alarme e chaveamento. No entanto, sequências de eventos inesperadas ou inadvertidas podem ocorrer, podendo resultar em transientes operacionais ou até no desligamento automático do reator (Trip), gerando prejuízos financeiros devido ao tempo de inatividade da usina e potenciais riscos à segurança e à integridade do reator. Diante desse cenário, este estudo propõe o desenvolvimento de uma ferramenta baseada em métodos de *Deep Learning*, capaz de prever uma situação de Trip e os próximos eventos de uma sequência, utilizando sequências de eventos disponíveis. Para validar a eficácia da ferramenta, foram realizados experimentos com um estudo de caso real, utilizando 10 anos de dados do SPAL de Angra 2. Como resultado, a melhor configuração dos modelos de inteligência artificial alcançou 99,94% de precisão média na previsão de Trip e 99,82% na previsão do evento seguinte.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

TRANSFORMER NEURAL NETWORKS AND OTHER DEEP LEARNING METHODS APPLIED TO THE EARLY TRIP IDENTIFICATION IN A PWR NUCLEAR REACTOR AND EVENT SEQUENCE PREDICTION DURING OPERATIONAL TRANSIENTS

Bernardo Mendonça Caixeta

March/2025

Advisor: Alan Miranda Monteiro de Lima

Department: Nuclear Engineering

Within the scope of the operation of the Angra 2 Nuclear Power Plant, a PWR-type plant, one of the most critical areas for specialists involves the analysis of the Sequence of Events, which corresponds to the acquisition and monitoring of changes in the values of binary variables, displaying in chronological order the actions taken by operators during the operation. During operational cycles, changes in the state of these variables generally occur in sequences that are expected and controlled by the operators. These changes are detected, acquired, time-tamped, and ordered by the Alarm Processing System (SPAL), encompassing alarm and switching events. However, unexpected or inadvertent event sequences may occur, which can result in operational transients or even the automatic shutdown of the reactor (Trip). This leads to financial losses due to the plant's downtime and potential risks to the safety and integrity of the reactor. In this context, this study proposes the development of a tool based on Deep Learning methods, capable of predicting a Trip situation and the next events in a sequence, using available event sequences. To validate the effectiveness of the tool, experiments were conducted with a real case study, using 10 years of data from Angra 2's SPAL system. As a result, the best configuration of the artificial intelligence models achieved 99.94% average accuracy in Trip prediction and 99.82% in predicting the next event.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

ACRÔNIMOS E ABREVIAÇÕES

AE – *AutoEncoder*;

API – *Application Programming Interface*;

AUC – *Area Under Curve*;

BERT - *Bidirectional Encoder Representations from Transformers*;

BPE – *Byte Pair Encoding*;

BPTT – *Backpropagation Through Time*;

CART – *Classification and Regression Trees*;

CEC – *Constant Error Carousel*;

CLM – *Causal Language Model*;

CNAAA – Central Nuclear Almirante Álvaro Alberto;

CNB – *Complement Naive Bayes*;

CNN – *Convolutional Neural Network*;

CUDA – *Compute Unified Device Architecture*;

DL – *Deep Learning*;

EDA – *Exploratory Data Analysis*;

EFH – Engenharia de Fatores Humanos;

FCS – Funções Críticas de Segurança;

FNN – *Feedforward Neural Network;*

FPR – *False Positive Rate*;

GABA - *Gamma-AminoButyric Acid*;

GPU – *Graphics Processing Unit*;

IA – Inteligência Artificial;

LLM – *Large Language Model*;

LSB – *Least Significant Bit*;

LSTM – *Long Short-Term Memory*;

MER – Modelo Entidade Relacionamento;

ML – *Machine Learning*;

MNB – *Complement Naive Bayes*;

NLP – *Natural Language Processing*;

NRC – *Nuclear Regulatory Commission*;

ODBC – *Open Database Connectivity*;

OLS – *Ordinary Least Squares*;

PCA – *Principal Component Analysis*;

PReLU – *Parametric ReLU*;

PWR – *Pressurized Water Reactor;*

RDBMS - *Relational Database Management System*;

ReLU – *Rectified Linear Unit;*

RESA – *Reactor Shutdown Alarm*;

RMSProp - *Root Mean Square Propagation*;

RNA – Rede Neural Artificial;

RNN – *Recurrent Neural Network*;

Seq2Seq – *Sequence-to-Sequence*;

SGD – *Stochastic Gradient Descent*;

SLOG - Sistema de Log de Eventos de Alarme,

SOE – *Sequence of Events*;

SPAL – Sistema de Processamento de Alarmes;

SPDS – *Safety Parameter Display Systems*;

SQL – *Structured Query Language*;

SVM – *Support Vector Machines*;

TMI – *Three Mile Island*;

TUSA – *Turbine Shutdown Alarm*;

t-SNE – *t-Distributed Stochastic Neighbor Embedding*;

CAPÍTULO 1
INTRODUÇÃO

1.1 APRESENTAÇÃO DO PROBLEMA

Com o avanço global rumo a sistemas de energia de baixo carbono, diversas alternativas tecnológicas, tanto já estabelecidas quanto emergentes, estão disponíveis, como usinas hidrelétricas, eólicas, solares, maremotrizes, células a combustível, geotérmicas e nucleares. Entre elas, a energia nuclear e as fontes renováveis se destacam por suas baixas emissões de CO_2 por unidade de energia gerada, além de serem significativamente mais eficientes na redução da poluição do ar em comparação aos combustíveis fósseis.

Em 2024, a indústria nuclear global se encontra em uma encruzilhada, marcada por uma crescente disparidade entre a atenção da mídia, os anúncios políticos e a percepção pública de um lado, e a realidade industrial do outro. Apesar do otimismo em torno da energia nuclear como uma solução para a crise climática, o setor enfrenta desafios para manter as frotas de reatores envelhecidas, acumula atrasos e excessos de custos em projetos de construção e tem dificuldades em conseguir desenvolver novos designs competitivos em tempo hábil. Até julho de 2024, 408 reatores estavam operacionais em 32 países, um a mais que em 2023, dez a menos que em 1989 e 30 abaixo do pico de 2002. A capacidade líquida nominal de geração de eletricidade nuclear atingiu 367,3 GW em meados de 2024, superando o recorde anterior de 367,1 GW em 2006. A participação da energia nuclear na geração bruta global de eletricidade comercial em 2023 foi de 9,15%, o valor mais baixo em quatro décadas e mais de 45% abaixo do pico de 17,5% em 1996. Em compensação, as energias renováveis não-hídricas continuaram a crescer, com um aumento de 13%, atingindo uma participação de 7,5% na energia primária mundial (SCHNEIDER *et al.*, 2024).

O cenário energético global está sendo remodelado pelas ambições climáticas nacionais, continentais e globais. As energias renováveis receberam um impulso significativo, impulsionado por compromissos como o *Global Renewables and Energy*

Efficiency Pledge, lançado na COP28. O investimento global em energias renováveis superou significativamente o investimento em energia nuclear. A Figura 1 ilustra essa disparidade, mostrando que o investimento em energia nuclear permaneceu relativamente constante nos últimos anos, enquanto o investimento em energias renováveis, principalmente em energia eólica e solar, disparou (SCHNEIDER *et al.*, 2024).

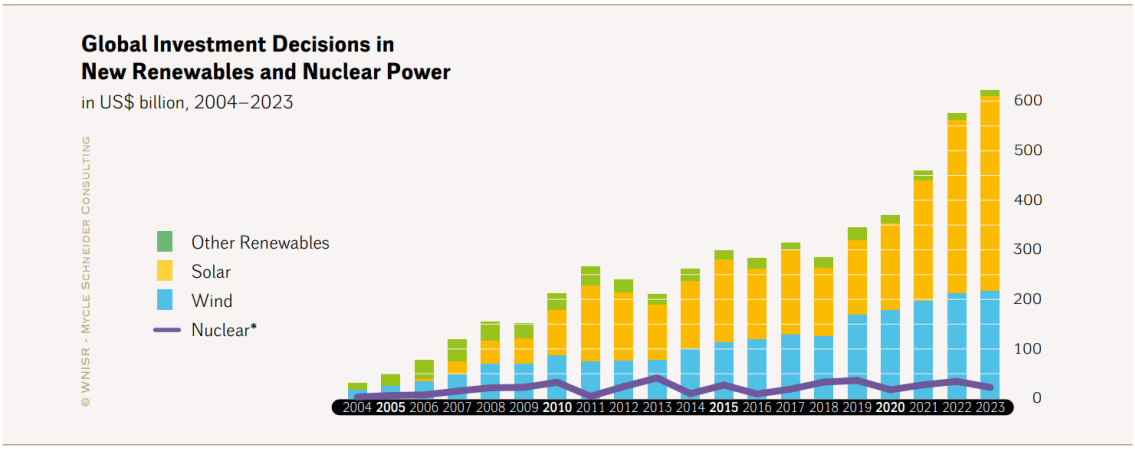

Figura 1. Decisões globais de investimento em energias renováveis e energia nuclear, 2004–2023 (SCHNEIDER *et al.*, 2024)

A indústria nuclear global enfrenta um futuro incerto em 2024. Enquanto a energia nuclear continua sendo uma fonte importante de energia de baixo carbono em muitos países, o setor precisa superar desafios significativos para se manter competitivo em um mercado energético em rápida mudança. O envelhecimento das frotas, os altos custos de construção, a concorrência das energias renováveis e as preocupações geopolíticas representam obstáculos consideráveis. O sucesso futuro da indústria nuclear dependerá de sua capacidade de inovar, reduzir custos, melhorar a segurança e responder às preocupações públicas. No Brasil, o programa nuclear começou a ganhar destaque a partir das décadas de 1970 e 1980, com a construção das usinas de Angra 1 e Angra 2. Apesar do avanço técnico e da alta capacidade instalada, questões como atrasos nos projetos, custos elevados e debates públicos sobre segurança nuclear e impactos ambientais têm limitado o crescimento do setor. Angra 3, por exemplo, representa um caso emblemático desses desafios, com décadas de interrupções em sua construção. Além disso, o Brasil está inserido em um contexto global onde a energia nuclear enfrenta um cenário de incertezas, de forma que, enquanto o setor nuclear tem registrado declínio na capacidade

instalada em várias regiões, fontes renováveis, como solar e eólica, apresentam crescimento exponencial. Essa tendência global levanta questionamentos sobre a viabilidade econômica e estratégica da expansão nuclear, sobretudo em países em desenvolvimento, onde recursos financeiros e tecnológicos são mais limitados (SCHNEIDER *et al.*, 2024).

Entre os cerca de 408 reatores nucleares em operação atualmente no mundo, os do tipo Reator a Água Pressurizada (PWR) são os mais prevalentes, representando aproximadamente 60% da capacidade nuclear instalada. Esses reatores utilizam água leve desmineralizada como moderador e refrigerante e utilizam como combustível o U-235, uma forma levemente enriquecida de urânio físsil. Um exemplo de reator PWR é a usina nuclear Angra 2, localizada na Central Nuclear Almirante Álvaro Alberto (CNAAA), em operação desde 2001, com uma capacidade nominal bruta instalada de 1.350 MWe (ELETRONUCLEAR, 2024).

De forma geral, uma usina nuclear é um complexo sistema composto por diversos subsistemas e componentes, todos desenvolvidos, construídos e operados conforme padrões rigorosos de segurança. O objetivo principal dessas normas é assegurar que, mesmo com o potencial risco causado pela radioatividade presente no núcleo do reator, a chance de um acidente radiológico seja extremamente baixa (IAEA, 2016).

As usinas nucleares dependem de sistemas de monitoramento para garantir a geração de energia elétrica de maneira segura. Esses sistemas precisam integrar informações sobre todas as variáveis essenciais para a operação e seus processos em andamento. No entanto, a grande quantidade de dados disponíveis nesses sistemas pode tornar o diagnóstico de eventos uma tarefa complexa, especialmente quando uma série de alarmes é disparada, sobrecarregando a capacidade cognitiva e uma tomada de decisão assertiva por parte dos operadores. Realizar diagnósticos nessas situações pode se tornar um grande desafio, pois o cérebro humano possui uma capacidade limitada de processar e interpretar grandes volumes de informação em curtos períodos, principalmente em situações em que o tempo para a tomada de decisão se torna um fator crítico. Além disso, fatores como a condição física, psicológica e o estresse dos operadores podem levar a

diagnósticos incorretos e a ações que podem agravar ainda mais a situação da usina e comprometer a integridade do núcleo do reator (SCHIRRU & PEREIRA, 2004).

Durante a operação, um evento anômalo, transientes operacionais, ou um acidente podem levar ao desligamento do reator, de maneira automática realizada pelos sistemas de segurança e monitoração da usina ou manualmente pelo operador, de forma a proteger a integridade física do núcleo e prevenir a liberação de material radioativo para a atmosfera. O desligamento imediato do reator geralmente ocorre através dos mecanismos de: queda das barras de controle do núcleo, para cessar a reação em cadeia de fissão; o aumento da concentração de ácido bórico na água de alimentação, realizada primariamente pelo Sistema de Injeção de Boro e Água Desmineralizada e o Sistema Suplementar de Injeção de Boro; a ativação do Sistema de Remoção de Calor Residual para promover o aumento do fluxo de água desmineralizada no circuito primário e realizar o resfriamento seguro do núcleo (ELETRONUCLEAR, 2001). O evento de desligamento rápido do reator é conhecido como *Trip* ou *Scram*, e o religamento da usina depende de uma análise detalhada do ocorrido, através do estudo da sequência de eventos que levou ao desligamento do reator, assim como da identificação da causa-raiz e evento iniciador. Esse processo é realizado por meio da elaboração de um relatório de evento significante, que inclui a descrição cronológica dos acontecimentos que levaram ao desligamento do reator (AUGUSTO, 2017; NRC, 1983; NRC, 2007).

Nos últimos anos, métodos baseados em Aprendizado de Máquina têm sido amplamente explorados na literatura para a identificação prévia de *Trip* em reatores nucleares e outras aplicações críticas no contexto da operação. Entre os trabalhos mais relevantes, (RANKIN & JIANG, 2016) propuseram um modelo computacional que utiliza um filtro de Kalman para prever, dentro do sistema de desligamento de uma usina nuclear, se os parâmetros de segurança atingirão limites críticos para a ocorrência de desarme do reator. Essa abordagem visa antecipar falhas potenciais e identificar o tempo restante até a ocorrência de *Trip* (*time-to-trip*) do reator, permitindo ações corretivas e preventivas, melhorando assim a segurança e a confiabilidade da operação da usina.

O estudo de (NICOLAU *et al.*, 2017) demonstrou a eficácia de algoritmos de aprendizado simbólico, como Sistemas Especialistas, no suporte ao diagnóstico da causa-

raiz de desligamento da usina e de determinação da emergência, enfatizando a necessidade de processamento em tempo real para auxiliar na operação e inferência lógica de emergências de usinas nucleares com base na análise de sequência de eventos.

O trabalho de (ALVES, 1993) aplicou Redes Neurais Artificiais (RNAs) para a análise de determinadas variáveis-chave de aquisição da usina nuclear Angra 1 com o objetivo de determinar a ocorrência de acidentes que levam ao *Trip* do reator, além da implementação de um Sistema Especialista capaz de tratar ocorrências de eventos não reconhecidos pelo sistema.

Recentemente, (OH *et al*., 2024) utilizaram Redes Neurais Profundas (DNNs) para o desenvolvimento de um sistema de previsão do tempo restante até um possível desarme do reator em usinas nucleares, visando aprimorar a segurança e reduzir erros humanos. O sistema utiliza técnicas de Inteligência Artificial (IA) e IA explicável (XAI), como Autoencoders e *Light Gradient-Boosting Machines*, para diagnosticar estados anormais e prever o tempo até um possível *Trip* do reator. A abordagem XAI fornece explicações sobre os resultados das previsões e destaca as variáveis principais que influenciam o estado da usina. Além disso, o sistema inclui uma interface que comunica os resultados aos operadores, auxiliando na tomada de decisões e na prevenção de falhas inesperadas.

Além disso, a previsão de sequências de eventos, uma das tarefas centrais deste trabalho, também tem sido amplamente investigada em outros domínios. Por exemplo, (VASWANI *et al*., 2017) introduziram os modelos Transformers como uma solução inovadora para a modelagem e geração de sequências de texto, inicialmente aplicada ao Processamento de Linguagem Natural, mas posteriormente adaptada para séries temporais e sistemas industriais. Ao modelar sequências de eventos para a tarefa de predição de novos *tokens*, é possível aplicar Modelos de Linguagem Causais (*Causal Language Model* – CLM), como o GPT (*Generative Pre-trained Transformer*) (RADFORD *et al*., 2018), aplicados na tarefa de geração de texto.

O estudo de (ZHA *et al*., 2021) realiza um comparativo entre redes convolucionais, LSTM e Transformers para analisar dados temporais de simulações para

uma usina nuclear, determinando se uma sequência de eventos pode levar a danos no núcleo do reator. Os resultados demonstraram que o modelo Transformer é capaz de aprender características dos dados sequenciais, obtendo uma precisão de classificação de aproximadamente 99% nos dados de teste.

Seguindo a linha dos modelos Transformer, (SPANGHER *et al*., 2023) aplicaram modelos Transformers, mais especificamente Transformers Mascarados Autorregressivos (*Masked Autoregressive Transformers*), para prever disrupções e a perda de estabilidade de plasma com implicações de danos em *tokamaks*, dispositivos utilizados em pesquisas de fusão nuclear. O modelo apresentou um aumento médio de 5% na métrica de Área Sob a Curva (*Area Under Curve* - AUC) em comparação com métodos existentes, ressaltando a capacidade de aprendizado do modelo Transformer.

Além disso, o trabalho de (KIM *et al*., 2023) propõe o uso de Transformers para extrair diretamente valores observáveis de dados experimentais em física nuclear, eliminando ambiguidades associadas a parâmetros não observáveis em modelos tradicionais. O modelo aprende padrões diretamente dos dados, permitindo previsões mais precisas sem depender de suposições teóricas rígidas e podendo impactar diretamente a forma como se interpreta dados em experimentos de física nuclear, tornando os modelos mais adaptáveis e melhorando a compreensão de fenômenos complexos.

O trabalho de (CHEN *et al*., 2023) propõe um modelo de Aprendizado Profundo para prever o tempo restante até o *Trip* em usinas nucleares durante condições anormais. O modelo utiliza redes LSTM para processar séries temporais de dados operacionais e prevê o tempo até a próxima parada do reator. Os resultados experimentais indicaram que o modelo proposto oferece uma ferramenta útil para os operadores de usinas nucleares se prepararem para eventos de parada automática durante condições operacionais anormais.

O estudo de (GRAVES *et al*., 2013) explora o uso de Redes Neurais Recorrentes Profundas (*Deep* RNNs) para aprimorar o reconhecimento automático de fala. A utilização de RNNs com a arquitetura LSTM tem mostrado resultados promissores, especialmente quando treinadas de forma *end-to-end* utilizando métodos como a

Classificação Temporal Conectivista (*Connectionist Temporal Classification* - CTC). No estudo, as RNNs profundas com LSTM alcançaram uma taxa de erro de 17,7% no *benchmark* de reconhecimento de fonemas TIMIT, representando um avanço significativo na área de Processamento de Linguagem Natural (NLP) e no desenvolvimento de modelos de linguagem.

O trabalho de (CHOI & LEE, 2020) propõe um modelo de Aprendizado de Máquina para detectar falhas em sensores durante situações de emergência em usinas nucleares. O modelo utiliza um índice de consistência que avalia a precisão das medições dos sensores, permitindo identificar e localizar rapidamente erros específicos. A pesquisa emprega uma rede neural LSTM para processar dados de parâmetros da planta coletados durante emergências simuladas, com erros artificiais inseridos nos dados. Os resultados indicam que o sistema treinado consegue distinguir eficazmente entre estados de erro e estados normais dos sensores, contribuindo para a prevenção de diagnósticos incorretos e melhorando a segurança operacional

Esses estudos fornecem uma base sólida para a utilização de métodos de Aprendizado Profundo em sistemas nucleares, demonstrando sua eficácia em prever situações críticas com alta exatidão. A proposta deste trabalho busca avançar nesse campo, adaptando e combinando técnicas de Aprendizado Profundo, incluindo Redes Neurais LSTM e Transformer, para realizar a identificação prévia de *Trip* e prever eventos subsequentes em situações de transiente operacional, contribuindo para o aumento da segurança e eficiência na operação de usinas nucleares.

1.2 OBJETIVOS

A operação segura e eficiente de reatores nucleares PWR, como Angra 2, depende de sistemas de monitoramento e controle capazes de lidar com a complexidade dos processos envolvidos. Eventos de transientes operacionais ou *Trip* representam desafios críticos, tanto pela interrupção na geração de energia elétrica quanto pelos potenciais riscos à segurança e integridade do reator. Diante desse cenário, este trabalho tem como objetivo principal desenvolver uma ferramenta baseada em Aprendizado Profundo para a identificação prévia de eventos de *Trip* e a previsão de sequências de eventos em cenários

de transiente operacional, contribuindo para a mitigação de riscos e para a tomada de decisão mais ágil e informada.

Para atingir esse objetivo, é necessário, inicialmente, consolidar uma base de dados a partir dos registros históricos do Sistema de Processamento de Alarmes (SPAL) da usina Angra 2, abrangendo 10 anos de operação. Esse conjunto de dados será preparado por meio de etapas de curadoria e análise exploratória de dados, a fim de identificar padrões e variáveis relevantes que possam fornecer indícios antecipados de situações críticas. Esse processo segue abordagens semelhantes às utilizadas por (NICOLAU *et al.*, 2017), (AUGUSTO, 2017) e (RANKIN & JIANG, 2016), que destacaram a importância da qualidade dos dados na modelagem de sistemas complexos.

Com a base de dados preparada, serão implementados e comparados modelos preditivos e de classificação baseados em Redes Neurais *Long Short-Term Memory* (LSTM) e Transformers, assim como algoritmos clássicos de Aprendizado de Máquina amplamente citados na literatura, como Ridge (HOERL & KENNARD, 1970; HILT & SEEGRIST, 1977), Floresta Aleatória (HO, 1995; BREIMAN, 2001), *Stochastic Gradient Descent* (SGD) (FERGUSON, 1982; BOTTOU & BOUSQUET, 2011), *Naive Bayes* (HAND & YU, 2001; CARUANA & NICULESCU-MIZIL, 2006) e *k-Nearest Neighbors* (kNN) (FIX & HODGES, 1951; COVER & HART, 1967; FIX & HODGES, 1989), consistindo em um *benchmark* capaz de verificar a capacidade de aprendizado e generalização de cada modelo em diferentes situações de operação.

O conjunto de modelos será utilizado tanto para identificar situações de iminência de *Trip*, com exatidão e antecedência suficientes para a tomada de decisões corretivas e preventivas, quanto para prever o próximo evento em uma sequência de eventos operacional, sendo utilizadas somente arquiteturas de redes LSTM e Transformer para este último caso. A aplicação de LSTMs segue trabalhos como o de (CHEN *et al*., 2022), (CHEN *et al*., 2023) e (CHOI & LEE, 2020), que demonstraram a capacidade dessas redes em capturar padrões de longo prazo em sistemas industriais e na predição de situações críticas em sistemas nucleares. Por outro lado, a arquitetura Transformer (VASWANI *et al.*, 2017), tem se mostrado eficaz em modelar dependências complexas em dados sequenciais ou que apresentam correlação temporal e será adaptada para os desafios

específicos do contexto nuclear descrito, em linha com o trabalho recente de (OH *et al*., 2024). Nesse contexto, a aplicação de redes Transformer nos problemas de identificação prévia de *Trip* e previsão de sequências de eventos em instalações nucleares é inédita na literatura, sendo primeiramente explorada neste trabalho.

Além do desenvolvimento dos modelos, este estudo busca avaliar comparativamente a eficácia dos modelos computacionais em termos de métricas como precisão, *Recall*, *F1-score* e a taxa de falsos positivos (*False Positive Rate* – FPR), permitindo uma análise mais assertiva da capacidade de inferência de cada modelo para os diferentes cenários operacionais. Esse tipo de análise é essencial para determinar a abordagem mais adequada ao cenário operacional de Angra 2, considerando as exigências de confiabilidade e tempo de resposta. Trabalhos como os de (GRAVES *et al*., 2013; HOCHREITER & SCHMIDHUBER, 1997) destacaram a importância de ajustes específicos para maximizar o desempenho de modelos em tarefas semelhantes.

Finalmente, a ferramenta proposta será validada por meio de um estudo de caso utilizando dados reais da usina Angra 2, com foco na aplicação prática e na avaliação de resultados em cenários reais de operação. Espera-se que os modelos desenvolvidos contribuam para a identificação de padrões críticos, reduzam o número de falsos positivos e forneçam previsões precisas para a sequência de eventos, fortalecendo a segurança e eficiência das operações nucleares. Além disso, este trabalho visa ampliar o conhecimento científico sobre o uso de Aprendizado Profundo em sistemas complexos, fornecendo uma base metodológica que poderá ser aplicada a outros setores críticos.

CAPÍTULO 2
FUNDAMENTAÇÃO TEÓRICA

2.1 REVISÃO BIBLIOGRÁFICA DE TÉCNICAS DE INTELIGÊNCIA ARTIFICIAL APLICADAS NA ANÁLISE DE SEQUÊNCIA DE EVENTOS E PREDIÇÃO DE TRIP EM USINAS NUCLEARES

(OH *et al*., 2022) propuseram um modelo de Inteligência Artificial Explicável baseado em *Light Gradient-Boosting Machine* (LightGBM) e *Shapley Additive Explanation* (SHAP) para prever o tempo restante antes de um *Trip* do reator e identificar as causas da degradação das condições do núcleo através de uma explicação detalhada sobre as variáveis que mais influenciaram a previsão do modelo, auxiliando os operadores a tomarem decisões de mitigação antecipadamente. A capacidade de interpretação dos resultados permite que operadores compreendam melhor as causas das anomalias, facilitando a implementação de medidas corretivas e representando um passo importante para a redução de erros humanos e a melhoria da segurança operacional.

(JANG *et al*., 2022) investigaram um modelo de arquitetura cognitiva denominada ADS-IDAC, que simula a tomada de decisão humana para automatizar respostas a situações de risco, se assemelhando a um Sistema Especialista e demais modelos simbólicos de conhecimento. O estudo incorpora um modelo cognitivo para simular situações de operação normal e acidentes, permitindo que os modelos tomem decisões baseadas em algoritmos de Inteligência Artificial. O trabalho identificou um potencial para automação de tarefas de alto risco em usinas nucleares, reduzindo a exposição humana a ambientes perigosos e possibilitando respostas mais rápidas a incidentes.

(CHEN *et al*., 2022) desenvolveram um modelo de previsão baseado em Redes Neurais LSTM para prever automaticamente o tempo até um *Scram* (parada automática do reator). O modelo utiliza dados de monitoramento em tempo real e gera uma contagem regressiva para os operadores, promovendo um melhor suporte à tomada de decisões, permitindo que operadores implementem contramedidas antes que o *Trip* ocorra ou evitando desligamentos desnecessários.

(ZHANG *et al*., 2023b) compararam modelos GRU (*Gated Recurrent Units*) e LSTM na previsão de fluxos de resfriamento durante acidentes severos em pequenas usinas nucleares, além da utilização da técnica SHAP para explicar a tomada de decisão dos modelos. Como resultado, os autores verificaram uma melhor compreensão da confiabilidade da IA em cenários críticos, aumentando a confiança dos operadores nesses modelos para prevenção de falhas nucleares, fornecendo transparência ao destacar quais variáveis tiveram maior influência na previsão.

(KOO *et al*., 2020) aplicaram modelos LSTM e GRU para identificar eventos iniciais em acidentes nucleares, além de classificar e detectar sinais anômalos em tempo real a partir de sinais de monitoramento, permitindo uma intervenção mais rápida para evitar escalonamentos de falhas. Os resultados do trabalho identificaram uma melhor identificação precoce de falhas críticas, reduzindo o tempo de resposta a acidentes.

(ZHA *et al*., 2021) utilizaram redes neurais convolucionais, LSTM e Transformers para classificar sequências de eventos em usinas nucleares. O modelo foi treinado com 10.000 simulações de acidente de *Blackout* usando o *software* RELAP5-3D, alcançando 99% de precisão na previsão de falhas críticas e danos ao núcleo do reator. A partir deste trabalho, os autores verificaram um potencial de substituir métodos tradicionais de análise de eventos, fornecendo previsões mais rápidas e precisas de danos ao núcleo do reator, ressaltando que a alta capacidade dos Transformers de processar grandes quantidades de dados sequenciais os torna uma alternativa poderosa para substituir métodos tradicionais de análise de segurança.

(YOO *et al*., 2018) desenvolveram um sistema de suporte inteligente para auxiliar operadores de usinas nucleares na tomada de decisões durante acidentes severos, assim como permitir avaliar a integridade e a vida útil dos instrumentos utilizados na operação. Através de simulações de acidentes, o sistema emprega modelos de Inteligência Artificial, como *Support Vector Classification* (SVC) para identificar eventos iniciadores e *Support Vector Regression* (SVR) para estimar o tempo crítico antes de falhas graves, além de diagnosticar acidentes logo após o *Trip* do reator. Os resultados indicam que o sistema possibilita antecipar acidentes, avaliar a integridade dos instrumentos e prever condições críticas, contribuindo para maior segurança e eficiência na operação de usinas nucleares.

2.2 SEQUÊNCIA DE EVENTOS

Após o acidente de *Three Mile Island* (TMI) em 1979, a importância dos fatores humanos em situações críticas, onde a velocidade de resposta e aspectos psicológicos influenciam diretamente o julgamento e a tomada de decisões, ganhou destaque. Em resposta a esse incidente, a Comissão Reguladora Nuclear (NRC) estabeleceu novas diretrizes rigorosas, exigindo mudanças fundamentais nos projetos das usinas nucleares (LEWIS *et al.*, 1979; NRC, 1983), com o objetivo de melhorar a performance dos operadores durante cenários de emergência, como por exemplo a inclusão de computadores para auxiliar na tomada de decisão. Desde então, a Engenharia de Fatores Humanos (EFH) tem concentrado seus esforços nesse campo, visando mitigar riscos associados a falhas humanas em cenários críticos e de alta complexidade, que geralmente envolvem um alto volume de indicadores, alarmes e sinais a serem analisados em um curto espaço de tempo.

Uma das principais iniciativas introduzidas pela NRC foi a otimização das interfaces homem-máquina nas salas de controle, por meio do desenvolvimento de sistemas computacionais de apoio à operação e à tomada de decisões, conhecidos como *Safety Parameter Display Systems* (SPDS). Esses sistemas foram criados para facilitar a identificação de eventos críticos, organizando e processando de maneira rápida as informações operacionais da planta, oferecendo aos operadores uma visão consolidada dos parâmetros mais relevantes da operação, sendo seu principal objetivo fornecer suporte ao operador durante emergências, garantindo a segurança operacional da planta (JOYCE & LAPINSKY, 1983; SCHIRRU & PEREIRA, 2004). Com isso, foram definidas seis Funções Críticas de Segurança (FCS), encarregadas de monitorar aspectos essenciais para determinar o status de segurança do reator: controle da reatividade, resfriamento do núcleo, remoção de calor do sistema primário, integridade do sistema de resfriamento, controle da radioatividade e a integridade da contenção. Essas funções correspondem às informações mínimas que devem ser mostradas aos operadores para verificar o status da planta e se qualquer medida deve ser tomada (SCHIRRU & PEREIRA, 2004).

Entre as funcionalidades principais do SPDS, destaca-se a avaliação contínua do estado de segurança da planta nuclear, por meio de monitoramento em tempo real das disposições lógicas das FCS. Além disso, o sistema prioriza as ações que devem ser

executadas pelos operadores em condições de emergência, orientando a tomada de decisões críticas. Outro aspecto relevante é a indicação dos procedimentos operacionais de emergência necessários para conduzir a planta a uma condição segura, assegurando que as diretrizes apropriadas sejam seguidas de maneira eficiente. O SPDS também disponibiliza elementos auxiliares para apoiar os operadores na execução das tarefas prescritas nos procedimentos de emergência, permitindo simultaneamente o monitoramento em tempo real do impacto dessas ações sobre os sistemas e parâmetros de segurança da planta, promovendo uma resposta coordenada e eficaz durante cenários críticos. Dessa forma, a utilização de sistemas de monitoração se posiciona como uma ferramenta indispensável para a gestão de emergências e auxílio à tomada de decisão em instalações nucleares (CULLBERT *et al.*, 1991; SCHIRRU & PEREIRA, 2004).

Dentro desse cenário, em 2002 foi implantado o Sistema Integrado de Computadores de Angra 2 (SICA / S2A2) na usina nuclear Angra 2, localizada na Central Nuclear Almirante Álvaro Alberto (CNAAA). Esse sistema foi desenvolvido para realizar a monitoração em tempo real dos parâmetros de segurança da planta, garantindo o acompanhamento contínuo das condições operacionais da usina (SCHIRRU & PEREIRA, 2004). Dentre os subsistemas presentes na rede de computadores SICA, três se destacam sobre as funções de realizar a detecção, aquisição e armazenamento dos eventos realizados durante a operação:

- Sequência de Eventos (SOE): responsável pela aquisição e acompanhamento das variações nos valores de variáveis binárias, i.e., ativação/desativação de alarmes e chaveamentos.
- Sistema de Processamento de Alarmes (SPAL): responsável por exibir, de forma cronológica, os eventos de alarme, permitindo que os operadores reconheçam e gerenciem essas ocorrências. Além disso, o sistema oferece a funcionalidade de monitorar valores de grupos de sinais analógicos, assim como acompanhar os estados de grupos de sinais binários da usina, proporcionando um controle abrangente e preciso dos parâmetros operacionais.

- Sistema de Log de Eventos de Alarme (SLOG): responsável pelo registro da sequência de eventos em uma impressora para agilizar a consulta de alarmes e chaveamentos pela sala de controle, assim como apresentar a sequência de eventos no terminal do servidor de eventos.

Esses sistemas operam em conjunto para realizar a detecção, aquisição, datação e ordenação dos eventos de mudança de variáveis binárias, e são utilizados para manter o registro de histórico cronológico de todas as ações realizadas pelos operadores na usina, assim como permitir a consulta e emitir relatórios de sequência de eventos pela operação.

De maneira geral, o SOE, SPAL e SLOG realizam a coleta e organização de dados cruciais durante eventos críticos, como transientes operacionais e desligamentos automáticos e manuais do reator e da turbina, e são ferramentas fundamentais para garantir a segurança operacional e a tomada de decisão informada pelos operadores. Os eventos são registrados de maneira a indicar de maneira clara e objetiva ao especialista informações críticas como: código do instrumento; data, hora e milissegundo da aquisição; descrição do alarme ou chaveamento; se a variável binária está variando do estado 0 → 1 ou 1 → 0, dentre outros. Durante a sequência de eventos ou uma avalanche de alarmes, é possível analisar a presença de variáveis específicas, a ordem em que os eventos ocorreram, e identificar comportamentos anômalos, aspectos essenciais para entender o que levou ao evento crítico e como evitá-lo no futuro.

Uma das funções mais importantes das sequências de eventos é a capacidade de identificar padrões e antecipar falhas antes que elas se concretizem. Em um sistema tão crítico quanto uma usina nuclear, a previsão de eventos críticos não se limita apenas à detecção de falhas em tempo real, mas também ao processo de manutenção preditiva. A análise de sequências de eventos passados pode revelar padrões que indicam a iminência de falhas e permitir que intervenções sejam feitas antes que a falha ocorra, reduzindo o risco e melhorando a segurança operacional. Por exemplo, ao estudar sequências de eventos passadas, é possível identificar a ordem dos eventos que precedem um *Trip*, permitindo que os operadores ou sistemas automatizados detectem sinais de alerta mais cedo. Além disso, a modelagem preditiva pode oferecer uma visão mais clara de falhas

sistemáticas que podem surgir em determinada sequência, ajudando a melhorar a estratégia de monitoramento e as decisões operacionais.

A crescente aplicação de Inteligência Artificial e Aprendizado Profundo no monitoramento e análise de dados operacionais em usinas nucleares está transformando a forma como eventos críticos são analisados, onde tecnologias como Redes Neurais Artificiais (RNAs) e modelos preditivos têm mostrado grande potencial para identificar padrões de falha, prever eventos futuros e analisar a progressão de acidentes (SANTOS *et al.*, 2021; DESTERRO *et al.*, 2023; NICOLAU *et al.*, 2023). A integração dessas tecnologias no contexto da sequência de eventos permite não só a detecção em tempo real de falhas iminentes, mas também a previsão de falhas antes que elas ocorram, oferecendo aos operadores tempo suficiente para tomar medidas corretivas.

2.3 REDES NEURAIS

2.3.1 REDE NEURAL BIOLÓGICA

O sistema nervoso humano pode ser concebido como uma rede altamente complexa e distribuída composta por bilhões de neurônios, as unidades de processamento de informações fundamentais do cérebro, que se comunicam através de sinapses, responsáveis por transmitir sinais elétricos e químicos entre as células nervosas. Cada neurônio recebe sinais de entrada por seus dendritos, processa essas informações em seu corpo celular e transmite a resposta para outros neurônios através de um axônio. Este processo de comunicação neuronal é o alicerce da percepção, tomada de decisão e memória, funções essenciais para o comportamento e aprendizado humano (HOPFIELD, 1982). De maneira geral, o sistema nervoso humano pode ser estruturado em três estágios principais, conforme demonstrado na Figura 2. No núcleo desse sistema reside o cérebro, simbolizado pela Rede Neural, cuja função primordial é processar de maneira ininterrupta as informações recebidas, percebê-las e tomar decisões adequadas. O diagrama da Figura 2 também apresenta dois fluxos distintos de setas: as setas orientadas da esquerda para a direita (representadas em preto) ilustram a transmissão linear de sinais informacionais ao longo do sistema, refletindo a passagem da informação através de cada estágio. Em contrapartida, as setas direcionadas da direita para a esquerda (representadas em azul) simbolizam o mecanismo de *feedback* do sistema, ou seja, a retropropagação das

respostas, sinalizando a reação a estímulos em cada estágio. Nesse contexto, os receptores desempenham o papel de converter estímulos do corpo ou do ambiente em impulsos elétricos, que são enviados para a Rede Neural (cérebro). Os efetores, por sua vez, traduzem os impulsos gerados pela Rede Neural em respostas concretas, configurando as saídas discerníveis do sistema (HAYKIN, 1999; HAYKIN, 2009).

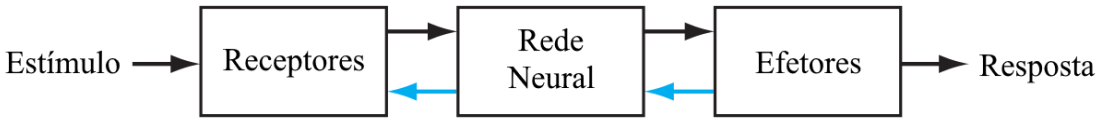

Figura 2. Representação em blocos do sistema nervoso (HAYKIN, 2009)

Sinapses, ou terminações nervosas, são unidades estruturais e funcionais elementares que mediam as interações entre os neurônios, sendo o tipo mais comum a sinapse química, que funciona da seguinte maneira: um processo pré-sináptico libera uma substância transmissora que se difunde através da junção sináptica entre os neurônios e, em seguida, age sobre um processo pós-sináptico. Assim, a sinapse converte um sinal elétrico pré-sináptico em um sinal químico e, posteriormente, de volta em um sinal elétrico pós-sináptico (STERRATT *et al.*, 2011). No cérebro adulto, a capacidade do cérebro de se adaptar pode ser explicada por dois mecanismos: a criação de novas conexões sinápticas entre neurônios e a modificação de sinapses existentes. Esses processos envolvem os axônios, que atuam como linhas de transmissão, e dendritos, correspondentes às zonas receptivas, constituem dois tipos de filamentos celulares diferenciados com base em suas características morfológicas. Um axônio tem uma superfície mais lisa, menos ramificações e maior comprimento, enquanto um dendrito possui uma superfície irregular e mais ramificações (HAYKIN, 1999; HAYKIN, 2009).

A maioria dos neurônios se comunica por meio de breves pulsos de voltagem, conhecidos como potenciais de ação ou espículas, de forma que esses pulsos se originam perto do corpo celular do neurônio e viajam por toda a extensão do neurônio com velocidade e intensidade constantes, sendo esse método de comunicação baseado nas propriedades físicas dos axônios. Os axônios por sua vez são estruturas longas e finas, com alta resistência elétrica e capacitância significativa distribuída ao longo de seu comprimento. Dessa forma, essa configuração permite que os axônios sejam modelados

como linhas de transmissão do tipo resistor-capacitor (circuito RC), e a análise desse mecanismo de transmissão revela que, quando uma voltagem é aplicada em uma extremidade do axônio, ela decai exponencialmente com a distância, diminuindo a níveis insignificantes ao chegar à outra extremidade (HAYKIN, 1999; HAYKIN, 2009), permitindo a regulagem da intensidade da informação transmitida.

As sinapses são fundamentais para a transmissão de impulsos nervosos entre neurônios, desempenhando um papel crucial ao permitir uma comunicação rápida e direta por meio da criação de circuitos elétricos. Além disso, a sinapse atua como um ponto de conexão onde ocorrem tanto a transmissão quanto o processamento de informações, sendo um meio vital de comunicação entre os neurônios. Em sua essência, os neurônios são especializados em enviar sinais para células-alvo específicas, de forma que as sinapses são o mecanismo através do qual isso é realizado. Na sinapse química, a atividade elétrica no neurônio pré-sináptico é convertida (por meio da ativação de canais de cálcio controlados por voltagem) na liberação de um composto químico chamado neurotransmissor, que se liga a receptores localizados na membrana plasmática da célula pós-sináptica. Esse neurotransmissor pode desencadear uma resposta elétrica ou ativar uma via de mensageiros secundários, que podem excitar ou inibir o neurônio pós-sináptico (HAYKIN, 2009; STERRATT *et al.*, 2011).

A plasticidade sináptica, que corresponde à capacidade de modificar a força das conexões entre os neurônios, é uma característica chave do cérebro humano, que permite o aprendizado e a adaptação a diferentes estímulos. Quando uma conexão entre dois neurônios é estimulada repetidamente, a sinapse se fortalece, facilitando a comunicação entre eles — um mecanismo fundamental para o armazenamento de memórias e o aprendizado contínuo. Esse processo é comumente conhecido como aprendizado sináptico e está diretamente relacionado ao conceito de memória de longo prazo (HOPFIELD, 1982; HAYKIN, 2009).

Em redes neurais biológicas, as informações temporais também desempenham um papel importante. O cérebro humano é altamente eficiente no processamento de sequências de eventos, onde o contexto temporal e a ordem dos estímulos influenciam as decisões e respostas. Esse tipo de processamento temporal é um componente fundamental

do aprendizado e pode ser observado em várias funções cognitivas, como a memória de trabalho e a tomada de decisão dinâmica (HOPFIELD, 1982; HAYKIN, 2009).

Nesse contexto, a Figura 3 mostra a estrutura básica de uma célula piramidal, que corresponde a um tipo de neurônio excitador encontrado principalmente no córtex cerebral, sendo responsável por funções cognitivas complexas, como memória e aprendizagem. Essas células possuem uma morfologia característica, com um corpo celular em forma de pirâmide, de onde se estendem várias projeções (dendritos e axônio) (HAYKIN, 2009).

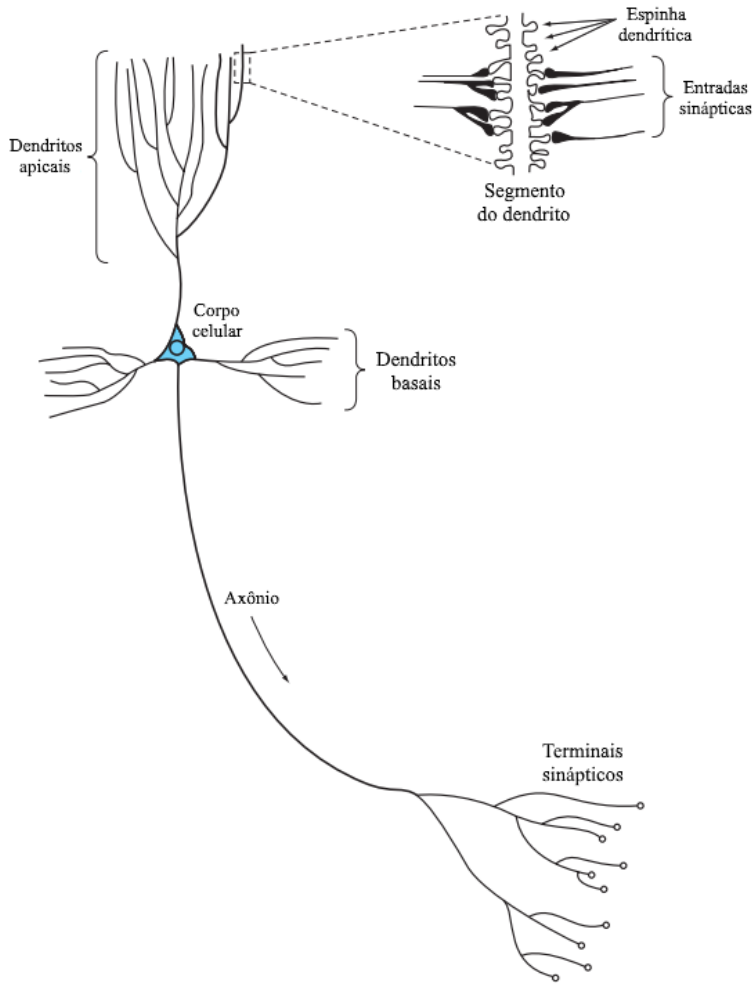

Figura 3. Representação de uma célula piramidal (HAYKIN, 2009)

As células piramidais são compostas por diversas estruturas importantes que estão relacionadas principalmente com a retenção de informação e propagação de sinais e estímulos nervosos, como:

- O corpo celular, que abriga o núcleo da célula. Nele ocorrem os processos metabólicos essenciais para a sobrevivência e funcionamento da célula, além de ser o local onde os sinais recebidos pelos dendritos são integrados antes de serem transmitidos ao longo do axônio (HOPFIELD, 1982; HAYKIN, 2009).
- As células piramidais possuem dois tipos principais de dendritos: os basais e os apicais. Os dendritos basais se estendem lateralmente e horizontalmente a partir da base do corpo celular, sendo responsáveis por receber sinais de neurônios vizinhos em curtas distâncias. Eles apresentam uma densa arborização, com diversas ramificações, aumentando a superfície de contato com outras células. O dendrito apical, por sua vez, se projeta verticalmente a partir do topo do corpo celular, muitas vezes alcançando camadas mais externas do córtex. Sua estrutura é mais alongada e arborizada, com várias ramificações na extremidade, permitindo que faça sinapses com neurônios localizados em regiões mais distantes do cérebro (HAYKIN, 2009).
- Os dendritos são cobertos por pequenas protuberâncias chamadas espinhas dendríticas, que formam sinapses com os terminais dos axônios de outros neurônios e desempenham um papel crucial na plasticidade sináptica, fundamental para processos como a aprendizagem e a memória. A capacidade das espinhas dendríticas de remodelarem suas conexões é uma característica central da adaptação e do armazenamento de informações no cérebro (HAYKIN, 2009).
- O axônio é responsável por transmitir o impulso elétrico do corpo celular para outros neurônios. Geralmente, cada célula piramidal possui apenas um axônio, que pode se ramificar ao longo de seu trajeto, formando múltiplas conexões. O local onde o axônio se origina é crucial para a geração do potencial de ação, o sinal elétrico que se propaga ao longo do axônio até os terminais sinápticos (HAYKIN, 2009).
- As sinapses nas células piramidais podem ser excitatórias, usando neurotransmissores como o glutamato, ou inibitórias, utilizando neurotransmissores como o GABA (mensageiro químico que transmite informações de um neurônio para outro, regulando o sistema nervoso). As sinapses excitatórias ocorrem principalmente nas espinhas dendríticas, enquanto

as sinapses inibitórias frequentemente se formam no corpo celular ou nos dendritos mais próximos a ele (HAYKIN, 2009).

Dessa forma, podem ser citadas as seguintes características do cérebro humano e suas estruturas fundamentais que serviram como a motivação biológica para a criação e desenvolvimento das Redes Neurais Artificiais:

1. Os dendritos, que correspondem aos prolongamentos ramificados dos neurônios, recebem estímulos e informações nervosas, e conduzem os impulsos nervosos em direção ao corpo celular do neurônio (HAYKIN, 2009).
2. O corpo celular integra esses sinais para gerar um impulso de entrada. Quando a soma dos sinais atinge um valor limite, o neurônio dispara, e o sinal é transmitido ao longo do axônio para outros neurônios (HAYKIN, 2009).
3. A intensidade do sinal transmitida depende da força das conexões sinápticas. Essas conexões podem ser inibitórias, diminuindo a intensidade do sinal, ou excitatórias, aumentando sua intensidade (HAYKIN, 2009).

O estudo das redes neurais biológicas inspirou diretamente o desenvolvimento das RNAs, especialmente no campo do Aprendizado de Máquina e, posteriormente, o Aprendizado Profundo. Assim como o cérebro humano ajusta as conexões sinápticas para aprender a partir de novas experiências, as RNAs ajustam os pesos sinápticos das conexões entre os neurônios artificiais durante o processo de treinamento. Esse processo de aprendizado, que envolve a adaptação dos pesos para minimizar uma função de erro, é essencial para a construção de modelos preditivos para tarefas comumente utilizadas no ramo da Inteligência Artificial, como regressão ou classificação (STERRATT *et al.*, 2011; SCHMIDHUBER, 2022).

As RNAs têm sido aplicadas com grande sucesso para resolver problemas complexos, como a identificação de padrões e a previsão de falhas. Em particular, as redes neurais do tipo *feedforward*, que apresentam camadas de neurônios dispostas de forma hierárquica, e as redes recorrentes (RNNs), que mantêm uma “memória” de informações passadas, são fortemente inspiradas nos mecanismos biológicos de processamento e armazenamento de informações temporais (HAYKIN, 2009).

A Neurociência Computacional estuda os processos de aprendizado e memória no cérebro e os traduz em modelos matemáticos que podem ser aplicados em algoritmos de aprendizado de máquina (HAYKIN, 2009). Os modelos de memória de longo prazo, como as Redes LSTM (do inglês, *Long Short-Term Memory*), são diretamente inspirados no funcionamento do cérebro humano, particularmente nos mecanismos de retenção de informações temporais, sendo eficazes no processamento de sequências de dados temporais, o que os torna extremamente úteis para a previsão de sequências de eventos e falhas, como em sistemas de monitoramento de reatores nucleares (CHEN *et al.*, 2021; ZHANG *et al.*, 2023a; CHAN *et al.*, 2023; CHOI & LEE, 2024). Além disso, a capacidade de retenção das informações relevantes ao longo do tempo dos modelos LSTM permite que as RNAs aprendam de forma eficiente a partir de sequências temporais, atenuando a perda do contexto de eventos passados (HAYKIN, 2009). Esse aspecto é crucial para a análise de falhas em sistemas complexos, como os encontrados em usinas nucleares e plantas industriais, onde a sequência e a temporalidade dos eventos podem ser determinantes para a antecipação de falhas e a segurança operacional (NICOLAU *et al.*, 2017; CHOI & LEE, 2024).

Assim como o cérebro processa informações de forma dinâmica e ajusta suas respostas com base em novas entradas, as RNAs ajustam seus pesos sinápticos para se adaptar a novos dados. A capacidade de aprender de forma eficiente com dados sequenciais torna as Redes Neurais modelos amplamente utilizados para prever falhas em sistemas industriais, como reatores nucleares, onde as sequências de eventos críticos podem ser antecipadas antes que resultem em falhas graves (SANTOS *et al.*, 2021).

O advento do mecanismo de atenção (BAHDANAU *et al.*, 2014) e, posteriormente, de modelos baseados em sua utilização, como as Redes Neurais Transformers (VASWANI *et al.*, 2017) para lidar com tarefas de processamento de sequências, como tradução automática e análise de linguagem natural, é inspirado em princípios biológicos fundamentais de como o cérebro processa e organiza informações ao longo do tempo. O cérebro humano não processa informações de maneira linear ou sequencial, mas sim usa mecanismos de atenção para focar em partes específicas de um estímulo enquanto ignora outras. Essa atenção seletiva permite que o cérebro se concentre

em detalhes importantes em uma sequência de estímulos e determine sua relevância dentro do contexto geral. O mecanismo de atenção nas redes Transformers é inspirado por esse processo biológico de seleção e foco em partes relevantes da entrada, permitindo que o Transformer pondere diferentes partes da entrada de maneira não sequencial, ao contrário dos modelos recorrentes como as LSTM, que processam a sequência de forma linear, uma etapa após a outra (VASWANI *et al.*, 2017). Essa capacidade de focar nas partes mais relevantes de uma sequência de eventos e ajustar a importância de cada uma delas de acordo com o contexto foi uma adaptação crucial para melhorar o desempenho em tarefas como previsão de falhas em sistemas temporais complexos.

Em um modelo Transformer, a informação é processada em paralelo, ao invés de ser passada de forma sequencial como nas RNNs e LSTMs. Isso melhora a eficiência computacional e permite que o modelo aprenda dependências de longo prazo em sequências de eventos de forma mais eficaz. Assim como no cérebro, onde diferentes áreas são responsáveis por processar diferentes tipos de informações em paralelo (por exemplo, visão, audição e memória), as redes Transformers conseguem processar múltiplas dependências de entrada simultaneamente, o que as torna extremamente eficientes para modelar sequências de eventos complexas, como as que ocorrem em sistemas de monitoramento de reatores nucleares.

2.3.2 REDES NEURAIS ARTIFICIAIS

O estudo e desenvolvimento de Redes Neurais Artificiais (RNAs) tem sido impulsionado desde o início pelo reconhecimento de que o cérebro humano realiza cálculos de uma maneira completamente diferente dos computadores digitais convencionais. O cérebro é um sistema de processamento de informações altamente complexo, não linear e paralelo (HAYKIN, 2009) que possui a capacidade de organizar seus componentes estruturais, conhecidos como neurônios, para executar certos cálculos e inferências (como reconhecimento de padrões, percepção e controle motor) de maneira mais rápida que qualquer modelo computacional existente atualmente. Como exemplo, a visão humana, é uma tarefa de processamento de informações, onde a função do sistema visual é fornecer uma representação do ambiente ao nosso redor e, mais importante, fornecer as informações necessárias para interagir com o ambiente.

De maneira geral, uma Rede Neural é um processador massivo distribuído paralelamente, composto por unidades de processamento simples denominadas neurônios, que possuem uma propensão natural para armazenar conhecimento baseado na experiência e torná-lo disponível para uso. Ela se assemelha ao cérebro em dois aspectos:

1. O conhecimento é adquirido pela rede a partir do seu ambiente por meio de um processo de aprendizado.

2. As forças de conexão entre os neurônios, conhecidas como pesos sinápticos, são usadas para armazenar o conhecimento adquirido.

O procedimento utilizado para realizar o processo de aprendizado é chamado de algoritmo de aprendizado, cuja função é modificar os pesos sinápticos da rede de maneira a correlacionar a informação de entrada com os dados observados na saída. No entanto, também é possível que uma Rede Neural modifique sua própria topologia, i.e., estrutura, motivada pelo fato de que neurônios no cérebro humano podem morrer e novas conexões sinápticas podem crescer e se intensificar (HAYKIN, 2009). Com isso, o processo de otimização dos pesos sinápticos de uma rede neural é o cerne do aprendizado da rede. É através da modificação desses pesos que a rede se torna capaz de realizar tarefas complexas, como reconhecimento de padrões, previsão de séries temporais e controle de sistemas dinâmicos. Esse problema de otimização tem como objetivo encontrar o conjunto de pesos sinápticos que minimiza uma função de custo, responsável por mensurar o erro entre a saída da rede e a resposta desejada. Essa função de custo define uma superfície de erro no espaço de pesos da rede, e o processo de otimização consiste em encontrar o ponto de mínimo nessa superfície, que corresponde ao conjunto de pesos que produz o menor erro, gerando a saída ótima para o problema.

Um exemplo de algoritmo de otimização é a Descida de Gradiente (do inglês, *Gradient Descent*), utilizado para encontrar um ponto de mínimo de uma função diferenciável, geralmente com base na intensidade da variação dos erros de predição em função dos pesos neuronais. Quanto maior o gradiente, mais íngreme é a inclinação da reta tangente na superfície de erro, consequentemente indicando uma maior variação nos pesos sinápticos, enquanto uma menor inclinação indica uma menor variação dos pesos.

Uma inclinação nula indica que o modelo atingiu um ponto de mínimo, podendo ser um mínimo local ou global da função de custo. Em termos matemáticos, o método da Descida de Gradiente atualiza os pesos da rede neural através da equação (1):

$$\Delta w_{ij} = \eta * \frac{\partial E}{\partial w_{ij}} \tag{1}$$

Onde: Δw_{ij} é o incremento nos pesos da conexão entre o nó i e o nó j

η é a taxa de aprendizado (número real positivo)

$\frac{\partial E}{\partial w_{ij}}$ é o gradiente associado com o peso entre a conexão

Na equação (1) é possível observar a presença de um termo η, que corresponde à taxa de aprendizado da rede. Esse termo é definido como um hiperparâmetro dentro do escopo da arquitetura de uma rede neural, ou seja, é um valor definido pelo usuário previamente ao treinamento e que não é alterado pela rede durante o processo de aprendizado. Esse termo define o quão abruptas serão as alterações nos pesos neuronais de forma que, uma alta taxa de aprendizado implica em saltos maiores dentro da superfície de erro, dificultando a convergência ou fazendo com que o modelo convirja rapidamente para soluções subótimas. Por outro lado, uma taxa de aprendizado muito baixa implica em variações muito sutis nos pesos neuronais, fazendo com que a rede demore mais a convergir ou que fique estagnada em mínimos locais.

Uma variação do algoritmo de Descida de Gradiente clássica é a sua implementação com um termo de *momentum* (BECKER & LECUN, 1989), que previne que o modelo fique estagnado em um mínimo local ao aplicar uma constante de amortecimento ao valor da variação de peso de cada conexão da iteração anterior, conforme pode ser observado na equação (2):

$$\Delta w_{ij} = \eta * \frac{\partial E}{\partial w_{ij}} + \left(\gamma * {\Delta w_{ij}}^{t-1}\right) \tag{2}$$

Onde: γ é o fator de *momentum* (número real positivo)

${\Delta w_{ij}}^{t-1}$ é o incremento de peso da iteração anterior

Com isso, os exemplos de algoritmos de otimização dos pesos neuronais representado pelas equações (1) e (2) também são chamados de algoritmos de retropropagação, ou *backpropagation*, pois é responsável por propagar os erros de cada camada da rede de trás para frente, ou seja, da camada de saída no sentido das camadas ocultas, permitindo o ajuste de pesos em todas as camadas da rede (BECKER & LECUN, 1989).

Além da Descrida de Gradiente, outros algoritmos de otimização amplamente citados na literatura incluem: o algoritmo Adam (do inglês, *Adaptive Moment Optimization*) (KINGMA & BA, 2014), que corresponde a uma variação do método de Descida de Gradiente Estocástico que se baseia na estimativa adaptativa de momentos de primeira e segunda ordem, dimensionando a taxa de aprendizado usando gradientes quadrados e aproveitando o *momentum* usando a média móvel do gradiente ao invés do gradiente em si. Dessa forma, o algoritmo Adam ajusta a taxa de aprendizado para cada parâmetro de forma adaptativa, levando em consideração não apenas a magnitude do gradiente, mas também o *momentum* acumulado dos gradientes e o quadrado dos gradientes; o algoritmo RMSProp (do inglês, *Root Mean Square Propagation*) (TIELEMAN & HINTON, 2012), projetado para melhorar a convergência em problemas onde o gradiente varia muito em diferentes direções, ajustando a taxa de aprendizado para cada parâmetro da rede de maneira adaptativa, com base no histórico dos quadrados dos gradientes. Em vez de usar uma taxa de aprendizado constante para todos os parâmetros, o RMSProp ajusta a taxa para cada peso da rede, levando em consideração as variações passadas do gradiente, ajudando a estabilizar o aprendizado e acelera a convergência, especialmente em problemas com gradientes muito ruidosos.

É observável que uma rede neural obtém sua capacidade de computação, primeiramente, através de sua estrutura distribuída paralelamente e, em segundo lugar, pela sua capacidade de aprender e, portanto, generalizar. A generalização refere-se à capacidade da rede neural de produzir saídas tão razoáveis quanto possível para entradas não encontradas durante o treinamento, ou seja, o seu processo de aprendizado. Essas duas capacidades de processamento de informações tornam possível que as redes neurais encontrem boas soluções aproximadas para problemas complexos (em grande escala) que outrora seriam intratáveis por algoritmos convencionais (HAYKIN, 2009).

O neurônio é uma unidade de processamento de informações fundamental para o funcionamento de uma Rede Neural, sendo responsável por ponderar os sinais de entrada, tratá-los através de uma função de ativação e então propagar esse sinal para os neurônios da camada seguinte. A Figura 4 apresenta uma representação genérica de um neurônio artificial, que serve como base para o design de diversas arquiteturas de Redes Neurais que serão abordadas nas próximas seções deste trabalho.

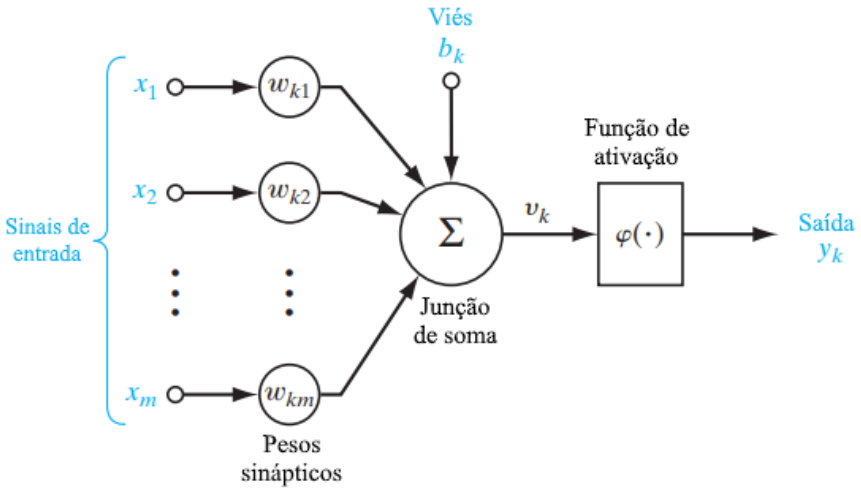

Figura 4. Representação de um neurônio artificial (HAYKIN, 2009)

No contexto da Figura 4, estão presentes três elementos básicos de um modelo neural:

- Um conjunto de sinapses, ou conexões, cada uma caracterizada por um peso ou intensidade próprios. Especificamente, um sinal x_j na entrada da sinapse j, conectada ao neurônio k, é multiplicado pelo peso sináptico w_{kj}. Diferente do peso de uma sinapse no cérebro, o peso sináptico de um neurônio artificial pode assumir valores tanto negativos quanto positivos (HAYKIN, 2009).
- Um ponto de junção para somar os sinais de entrada, ponderados pelas respectivas forças sinápticas do neurônio.
- Uma função de ativação para limitar a amplitude da saída do neurônio.

O modelo neural da Figura 4 também inclui um viés aplicado externamente, denotado por b_k. O viés tem o efeito de aumentar ou diminuir a entrada líquida da função de ativação, dependendo se ele for positivo ou negativo, respectivamente.

Em termos matemáticos, o neurônio *k* representado na Figura 4 pode ser descrito pelo par de equações (3) e (4) a seguir:

$$u_k = \sum_{j=1}^{m} w_{kj} \cdot x_j \tag{3}$$

$$y_k = \varphi(u_k + b_k) \tag{4}$$

Onde: u_k é a saída do combinador linear devido aos dados de entrada

w_{kj} são os pesos sinápticos do neurônio *k*

x_j são os sinais de entrada

b_k é o viés

$\varphi(\cdot)$ é a função de ativação

y_k é o sinal de saída do neurônio

Para uma Rede Neural, geralmente é atribuído o termo "conhecimento" para designar a sua capacidade de generalização com base no universo de informações passados a ela durante o treinamento. Porém, é possível abordar a questão do que seria o conhecimento de uma RNA propriamente dito através da seguinte definição genérica: Conhecimento refere-se à informação ou modelos armazenados usados por uma pessoa ou máquina para interpretar, prever e responder de maneira adequada ao mundo externo (FISCHLER & FIRSCHEIN, 1987). As principais características da representação do conhecimento são duas: (1) quais informações são efetivamente tornadas explícitas; e (2) como a informação é fisicamente codificada para uso posterior. Por sua própria natureza, portanto, a representação do conhecimento é orientada por objetivos (HAYKIN, 2009).

Em aplicações reais para máquinas "inteligentes", pode-se afirmar que uma boa solução depende de uma boa representação do conhecimento, e é constatado que o mesmo se aplica às Redes Neurais (WOODS, 1986; HAYKIN, 1999). No entanto, frequentemente é observado empiricamente que as formas possíveis de representação, desde as entradas até os parâmetros internos da rede, são altamente diversificadas, o que torna o desenvolvimento de uma solução satisfatória por meio de uma RNA um desafio de *design*. Um dos desafios fundamentais no *design* de redes neurais reside na representação do conhecimento, ou seja, a importância de determinar quais informações

devem ser explicitadas e como codificá-las de forma eficaz para uso subsequente. Esse desafio é particularmente evidente no contexto da estabilidade-plasticidade de um modelo neural, onde a rede deve equilibrar a capacidade de aprender novas informações, mantendo o conhecimento previamente adquirido. A incorporação de informações contextuais também é um desafio significativo pois, como as redes neurais são sistemas distribuídos, cada neurônio é potencialmente afetado pela atividade global da rede e a representação do conhecimento deve levar em conta essa interconexão e garantir que o contexto seja adequadamente incorporado no processamento da informação. A escolha da arquitetura de rede desempenha um papel crucial no desempenho e na capacidade de generalização de forma que determinar o número de camadas, o número de neurônios em cada camada e o padrão de conectividade é essencial para otimizar o desempenho da rede.

Uma das principais tarefas de uma Rede Neural é aprender um modelo do mundo (ambiente) em que está inserida e manter esse modelo suficientemente consistente com o mundo real para atingir os objetivos específicos da aplicação em questão. O conhecimento do mundo consiste em dois tipos de informação (HAYKIN, 2009):

1. O estado conhecido do mundo, representado por fatos sobre o que é e o que foi conhecido; essa forma de conhecimento é chamada de informação prévia (à priori).

2. Observações (medidas) do mundo, obtidas por meio de sensores projetados para explorar o ambiente em que a Rede Neural deve operar.

Normalmente, essas observações são intrinsecamente ruidosas, sujeitas a erros devido ao ruído dos sensores, problemas físicos e às imperfeições do sistema de aquisição e de processamento da informação. Em todo o caso, as observações obtidas fornecem o conjunto de informações a partir do qual são extraídos os exemplos utilizados para treinar a Rede Neural, e representam a base de conhecimento utilizada por ela para estabelecer o processo de aprendizado.

O tema de como o conhecimento é realmente representado dentro de uma RNA é, no entanto, bastante complexo. Dessa forma, (HAYKIN, 2009) propôs a definição de

quatro regras para a representação do conhecimento de uma Rede Neural que são de natureza geral e intuitiva, conforme descrito a seguir:

- **Regra 1:** Entradas similares (ou seja, padrões extraídos) de classes semelhantes geralmente devem produzir representações similares dentro da rede e, portanto, devem ser classificadas como pertencentes à mesma classe.
    - Isso sugere que a rede deve projetar suas representações internas de tal forma que padrões relacionados fiquem próximos uns dos outros no espaço amostral. Esse princípio é crucial para a capacidade da rede de fazer generalizações apropriadas e coerentes, facilitando a classificação de novos exemplos que compartilham características com exemplos vistos anteriormente.
- **Regra 2:** Itens que devem ser categorizados como classes distintas devem receber representações significativamente diferentes dentro da rede.
    - Isso ajuda a garantir que a rede possa distinguir efetivamente entre classes que não têm semelhança, reduzindo a chance de confusão entre classes que podem ser visivelmente diferentes. Ter representações bem separadas para classes diferentes é essencial para a acurácia da classificação e para evitar sobreposições indesejadas entre categorias distintas.
- **Regra 3:** Se uma característica específica é importante, então deve haver uma grande quantidade de neurônios envolvidos na representação dessa característica na rede.
    - Isso implica que características importantes devem ser representadas com uma quantidade adequada de neurônios da rede para garantir que elas sejam suficientemente destacadas durante o treinamento e a inferência, o que pode melhorar a capacidade da rede de reconhecer e processar essas características de forma eficaz.
- **Regra 4:** Informações prévias e invariâncias devem ser incorporadas ao *design* (arquitetura) de uma Rede Neural sempre que estiverem disponíveis, para simplificar o projeto da rede, evitando que ela tenha que aprender esses aspectos por conta própria.

- Em vez de deixar que a rede aprenda esses aspectos de forma autônoma, o que pode ser mais custoso e demorado, incorporá-los no *design* da rede pode simplificar o processo de treinamento e melhorar a eficiência da rede. Isso ajuda a reduzir a complexidade do aprendizado, garantindo que a rede não precise descobrir esses aspectos por conta própria.

Assim como existem diferentes maneiras pelas quais os seres humanos aprendem a partir do ambiente que os cerca, o mesmo ocorre com as redes neurais. De modo geral, é possível categorizar os processos de aprendizagem das redes neurais da seguinte forma: aprendizado supervisionado, aprendizado não supervisionado e aprendizado por reforço. Esses diferentes tipos de aprendizagem em redes neurais refletem os processos de aprendizado humano e como o cérebro é capaz de aprender e generalizar com base em observações e estímulos recebidos do ambiente.

2.3.3 APRENDIZADO SUPERVISIONADO

Em termos conceituais, é possível imaginar o aprendizado supervisionado como um modelo professor-aluno de ensino para um modelo de aprendizado, onde o "professor" corresponde a alguém que possui conhecimento sobre o ambiente, e esse conhecimento é representado por um conjunto de exemplos de entrada e saída, no entanto, o ambiente é desconhecido para a rede neural. Agora, supondo que tanto o professor quanto a rede neural sejam expostos a um vetor de treinamento (ou exemplo) extraído do mesmo ambiente, devido ao seu conhecimento prévio, o professor é capaz de fornecer à rede neural uma resposta desejada para esse vetor de treinamento, de forma que essa resposta desejada representa a ação ótima ou ideal que a rede neural deve executar. Dessa forma, algoritmos de aprendizado supervisionado são treinados com um conjunto de dados rotulados, onde cada entrada possui uma resposta desejada associada. O objetivo do aprendizado supervisionado é fazer com que o modelo aprenda a mapear as entradas para as saídas corretas, com base nos exemplos fornecidos durante o treinamento. Esse método é amplamente utilizado em problemas de classificação e regressão, sendo ideal para tarefas em que se conhece a resposta correta para as entradas (BAUM & WILCZEK, 1988; HAYKIN, 2009).

O processo de aprendizado supervisionado envolve várias etapas, começando pela preparação dos dados, onde podem ser aplicadas técnicas de limpeza, normalização e, dependendo do problema, uma Análise Exploratória de Dados (EDA), identificação e tratamento de *outliers*, preenchimento de dados ausentes e balanceamento de classes para garantir que os dados estejam prontos para serem usados no treinamento. Em seguida, dependendo do objetivo e da quantidade de dados disponíveis, os dados são divididos em conjuntos de treinamento, validação e teste, garantindo que o modelo seja avaliado de forma imparcial com observações nunca antes vistas pelo modelo. A etapa final envolve a avaliação do modelo, utilizando métricas como precisão, *Recall*, *F1-score*, e a matriz de confusão, especialmente em problemas de classificação desbalanceada, como a previsão de *Trips*, onde a classe de falha é muito menos representada que a classe normal.

A Figura 5 mostra uma representação esquemática do aprendizado supervisionado. É possível observar que, neste caso, a informação obtida do ambiente é conhecida pelo professor e deve ser aprendida pela rede neural. Para isso, a rede é treinada com os dados do ambiente e com as respostas ótimas correspondentes, informadas pelo professor, se adaptando ao atualizar seus pesos sinápticos de maneira a minimizar uma função de erro sobre as respostas do modelo para com os dados reais (HAYKIN, 2009).

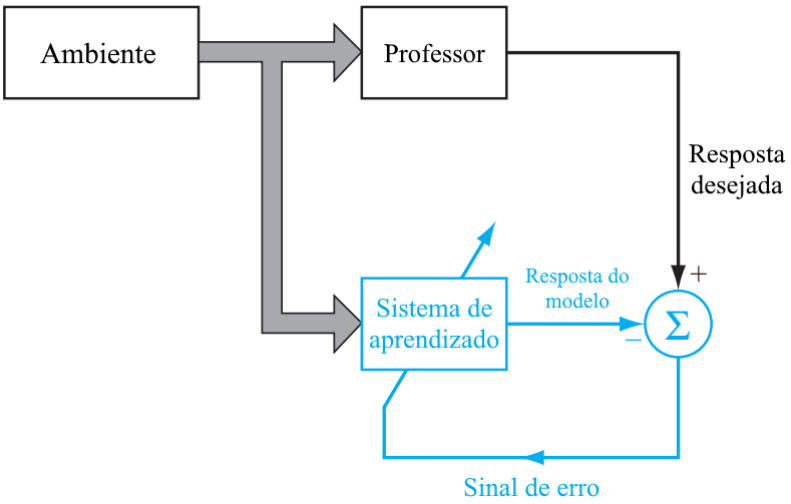

Figura 5. Representação de aprendizado supervisionado (HAYKIN, 2009)

Na Figura 5, observa-se que o processo de aprendizado supervisionado constitui um sistema de *feedback* de *loop* fechado, mas o ambiente desconhecido pelo modelo está

fora do *loop*. Como medida de desempenho para o sistema, uma alternativa possível seria considerar o erro quadrático médio, ou a soma dos erros quadrados sobre a amostra de treinamento, definido como uma função dos parâmetros livres (ou seja, pesos sinápticos) do sistema. Essa função pode ser visualizada como uma superfície de erro multidimensional com os parâmetros livres como coordenadas, sendo a verdadeira superfície de erro calculada com base em todos os exemplos possíveis de entrada e saída (HAYKIN, 2009). Cada operação do sistema sob a supervisão do professor é representada como um ponto na superfície de erro e para que o sistema melhore seu desempenho ao longo do tempo e, portanto, aprenda com o professor, o ponto de operação deve se mover sucessivamente em direção a um ponto mínimo da superfície de erro, de forma que esse ponto mínimo pode ser um mínimo local ou global. Um sistema de aprendizado supervisionado é capaz de fazer isso com as informações úteis que possui sobre o gradiente da superfície de erro correspondente ao comportamento atual do sistema (BAUM & WILCZEK, 1988).

O gradiente da superfície de erro em qualquer ponto é um vetor que aponta na direção de maior declive. No caso do aprendizado supervisionado a partir de exemplos, o sistema pode usar uma estimativa instantânea do vetor gradiente, ou seja, o modelo utiliza o gradiente calculado a partir de um único exemplo de cada vez durante o treinamento, com os índices dos exemplos presumidos como se fossem índices de tempo sequenciais, de tal forma que o modelo faz pequenas atualizações de seus parâmetros a cada novo exemplo, sem necessariamente ter uma visão completa de todos os dados. O uso dessa estimativa resulta em um movimento do ponto de operação na superfície de erro que tipicamente assume a forma de uma caminhada aleatória (do inglês, *random walk*), assumindo um comportamento irregular e presumidamente aleatório. No entanto, dado um algoritmo projetado para minimizar a função de erro, um conjunto adequado de exemplos de entrada e saída, e tempo suficiente para realizar o treinamento, um sistema de aprendizado supervisionado geralmente é capaz de aproximar um mapeamento desconhecido de entrada e saída com precisão razoável (HAYKIN, 2009).

Os parâmetros da rede são ajustados sob a influência combinada do vetor de treinamento e do sinal de erro, sendo este definido como a diferença entre a resposta desejada e a resposta real da rede. Esse ajuste é realizado de forma iterativa, passo a passo,

com o objetivo de fazer a rede neural eventualmente imitar o professor, de forma que se presume que essa imitação seja ótima em um sentido estatístico (BAUM & WILCZEK, 1988). Dessa forma, o conhecimento do ambiente disponível ao professor é transferido para a rede neural por meio do treinamento e armazenado na forma de pesos sinápticos "fixos", representando a memória de longo prazo. Quando essa condição é alcançada, o professor pode ser dispensado, permitindo que a rede neural lide com o ambiente de forma autônoma (HAYKIN, 2009).

O aprendizado supervisionado é realizado através de algoritmos que podem ser divididos em algumas categorias distintas (SCHMIDHUBER, 2022):

1. **Modelos lineares**: Algoritmos como a Regressão Logística e as Máquinas de Vetores de Suporte (SVM) são amplamente utilizados para tarefas de classificação binária, como a identificação de falhas iminentes em reatores nucleares. Esses modelos funcionam bem quando as classes são linearmente separáveis.

2. **Árvores de Decisão e Florestas Aleatórias**: As Árvores de Decisão são úteis devido à sua interpretação intuitiva, e as Florestas Aleatórias (do inglês, *Random Forest*) são eficientes para lidar com dados desbalanceados e ruidosos. Elas funcionam bem para prever falhas em sistemas complexos, como os encontrados em usinas nucleares.

3. **Redes Neurais Profundas (DNN)**: Para problemas mais complexos e não lineares, como a previsão de falhas em sistemas dinâmicos, DNNs são amplamente utilizadas. Elas são capazes de modelar relações complexas entre variáveis e capturar dependências não lineares.

Com isso, o aprendizado supervisionado requer a disponibilidade de uma resposta alvo ou desejada para a realização de um mapeamento específico de entrada e saída, através da minimização de uma função de custo de interesse. Além disso, o aprendizado supervisionado depende da disponibilidade de uma amostra de treinamento com exemplos rotulados, onde cada exemplo consiste em um sinal de entrada (estímulo) e a resposta desejada correspondente. Na prática, a coleta de exemplos rotulados é uma tarefa

que consome tempo e recursos consideráveis, especialmente quando lidamos com problemas de aprendizado em larga escala.

No contexto de previsão de desarme do reator em usinas nucleares, o aprendizado supervisionado é essencial para classificar sequências de eventos como normais ou anômalas. Nesse contexto, dados históricos de alarmes e eventos, como os provenientes do SPAL e SLOG, podem ser utilizados para treinar os modelos de aprendizado, que então são capazes de identificar padrões e relações contextuais que indicam falhas iminentes, melhorando a segurança operacional e a tomada de decisões. Além disso, o aprendizado supervisionado é essencial para o problema de previsão de eventos, pois a previsão do evento subsequente de uma determinada sequência é comparável aos rótulos das classes de um problema de classificação, tendo o modelo a tarefa de escolher o evento com maior probabilidade de ser o próximo dentro de um universo de eventos conhecidos, constatando que esses modelos de aprendizado também podem ser aplicados na tarefa de previsão quando os dados de saída esperados são valores discretos.

2.3.4 APRENDIZADO NÃO SUPERVISIONADO

No aprendizado supervisionado, o processo de aprendizado ocorre sob a orientação de um “professor”. No entanto, no paradigma conhecido como aprendizado não supervisionado, como o próprio nome sugere, não há um usuário externo para supervisionar o processo de aprendizado. Isso significa que não existem exemplos rotulados da função a ser aprendida pela rede neural ou por outro modelo computacional, como pode ser observado na Figura 6. Dessa forma, o aprendizado não supervisionado busca identificar estruturas ocultas ou padrões dentro dos dados, onde seu objetivo principal é explorar e organizar os dados sem a necessidade de uma resposta ou saída previamente definida (BECKER, 1991).

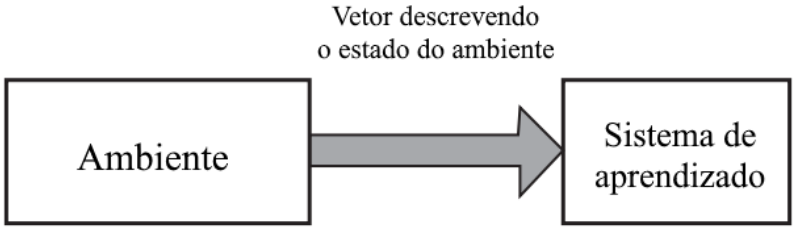

Figura 6. Representação de aprendizado não supervisionado (HAYKIN, 1999)

No aprendizado não supervisionado, ou auto-organizado, não há um professor ou crítico externo para supervisionar o processo de aprendizado. Em vez disso, é fornecida uma medida de qualidade da representação que a rede deve aprender, independente da tarefa. Os parâmetros livres da rede são otimizados em relação a essa medida. Para uma medida específica e independente da tarefa, uma vez que a rede tenha se ajustado às regularidades estatísticas dos dados de entrada, ela desenvolve a capacidade de formar representações internas para codificar características da entrada e, assim, criar classes automaticamente (BECKER, 1991; HAYKIN, 2009).

Para realizar o aprendizado não supervisionado, pode-se usar uma regra de aprendizado competitivo, utilizado em mapas auto-organizáveis, como o Mapa de Kohonen (KOHONEN, 1982). Por exemplo, é possível utilizar uma rede neural que consiste em duas camadas: uma camada de entrada e uma camada competitiva, de forma que a camada de entrada recebe os dados disponíveis e a camada competitiva é composta por neurônios que competem entre si (de acordo com uma regra de aprendizado) pela "oportunidade" de responder às características contidas nos dados de entrada. Na sua forma mais simples, a rede opera com uma estratégia de "o vencedor leva tudo", onde o neurônio com o maior total de entrada "vence" a competição e é ativado, enquanto todos os outros neurônios na rede são então desativados (HAYKIN, 1999; HAYKIN, 2009).

O aprendizado não supervisionado apresenta uma implementação que depende da disponibilização de uma medida independente da tarefa para a qualidade da representação que a rede deve aprender de maneira auto-organizada. Isso significa que o próprio modelo deve ser capaz de inferir a respectiva classe ou sequência de cada observação, assim como descobrir estruturas e padrões nos dados de maneira independente, permitindo uma compreensão mais profunda dos mesmos (BECKER, 1991).

Existem várias técnicas e algoritmos amplamente utilizados no aprendizado não-supervisionado:

1. **Clusterização**: A clusterização é o processo de agrupar dados em clusters ou grupos baseados em similaridades, de maneira a manter dados mais semelhantes nos mesmos conjuntos. Algoritmos como K-Means (MACQUEEN, 1967), DBSCAN (ESTER *et al.*, 1996) e PAM (KAUFMAN & ROUSSEEUW, 1990),

assim como Redes Neurais Artificiais (DU, 2010; SHAHID, 2023) são frequentemente usados para detectar anomalias e identificar padrões operacionais que podem indicar falhas iminentes. Esses algoritmos são úteis para agrupar dados operacionais de reatores nucleares, ajudando a identificar padrões normais e anômalos.

2. **Redução de Dimensionalidade**: Técnicas como PCA (PEARSON, 1901) e t-SNE (HINTON & ROWEIS, 2002; VAN DER MAATEN & HINTON, 2008) são utilizadas para reduzir a dimensionalidade dos dados, preservando suas características principais. Essas técnicas ajudam a visualizar grandes volumes de dados e a detectar padrões que seriam difíceis de identificar em dados de alta dimensionalidade, como as sequências de eventos em sistemas de monitoramento.

No contexto de previsão de falhas em usinas nucleares, o aprendizado não supervisionado pode ser útil para identificar anomalias e padrões desconhecidos em sequências de eventos. Técnicas como clusterização temporal podem ser aplicadas para descobrir mudanças nos padrões de operação que indicam uma falha iminente, mesmo antes que ela ocorra. Além disso, a redução de dimensionalidade pode ser usada para simplificar a análise de grandes volumes de dados, facilitando a identificação de sequências críticas que merecem atenção.

O aprendizado não supervisionado, ao buscar padrões e estruturas nos dados sem a necessidade de rótulos explícitos, é fundamental para a identificação de características subjacentes em grandes volumes de informações, como as sequências de eventos em sistemas críticos (ALVARENGA *et al.*, 1997). Para que essas técnicas de aprendizado não supervisionado possam efetivamente identificar e agrupar padrões, é necessário que os modelos de rede neural e demais modelos de aprendizado sejam capazes de processar e transformar os dados de maneira eficiente. Nesse cenário, as funções de ativação desempenham um papel crucial, pois são responsáveis por introduzir não-linearidades no modelo, permitindo que redes neurais capturem padrões complexos e interações entre as variáveis, podendo melhorar a capacidade do modelo em identificar e separar padrões de comportamento em dados, crucial para a previsão de *Trips* e outras anomalias em sistemas complexos como usinas nucleares.

2.3.5 FUNÇÕES DE ATIVAÇÃO

Uma função de ativação é usada para limitar a amplitude da saída de um neurônio, muitas vezes referida como uma função de compressão, restringindo o intervalo de valores possíveis da saída a uma faixa de valores finita e garantindo que a saída da rede neural permaneça dentro de um intervalo previsível e evite problemas de saturação de gradiente. Essa limitação da amplitude de saída ajuda a manter a estabilidade do modelo e introduz uma não linearidade, impedindo que os valores da saída cresçam indefinidamente e, assim, tornando os cálculos matriciais das redes neurais mais eficientes e controlados e permitindo que a rede aprenda e modele relações complexas entre entrada e saída. Sem funções de ativação, uma rede neural seria simplesmente uma combinação linear de suas entradas, o que a limitaria a modelar apenas relações lineares.

A não linearidade introduzida pela função de ativação é distribuída por toda a rede, permitindo que ela se aproxime de qualquer função contínua com precisão arbitrária, desde que tenha um número suficiente de neurônios (HAYKIN, 2009; HINKELMANN, 2018). Isso está relacionado ao Teorema da Aproximação Universal, que corresponde a um resultado teórico fundamental dentro do ramo do aprendizado de máquina que estabelece que, sob certas condições, uma RNA com uma única camada oculta pode aproximar qualquer função contínua com precisão satisfatória, desde que a rede tenha um número suficiente de neurônios na camada oculta e use uma função de ativação não linear apropriada (CYBENKO, 1989; HAYKIN, 1999).

Existem diferentes tipos de funções de ativação citadas na literatura, e a escolha da função adequada depende do problema a ser resolvido, dependendo de fatores como o formato dos dados de entrada e saída, da profundidade da rede, se envolve modelagem de probabilidades, seleção de características, reconstrução de dados, dentre outros aspectos, tornando a escolha de uma ou mais funções de ativação em um modelo de rede neural um desafio. Algumas das funções mais comumente utilizadas incluem:

1. **Sigmoide logística:** é uma função contínua e diferenciável que realiza a transformação de um valor real como valor de entrada para um valor dentro do intervalo [0, 1] na saída do neurônio, muitas vezes sendo associada a uma função de probabilidade, como por exemplo a probabilidade de uma observação ser

associada a uma classe ou outra, ou a probabilidade de uma determinada informação ser mantida pelos portões de uma célula LSTM (HOCHREITER & SCHMIDHUBER, 1997; KOLEN & KREMER, 2001). Quanto maior o valor de entrada (mais positivo), mais próximo o valor de saída será de um. Por outro lado, quanto menor o valor de entrada (mais negativo), mais perto a saída será de zero. Matematicamente, ela é representada pela equação (5), e sua derivada representada pela equação (6):

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{5}$$

$$\sigma'(x) = \frac{e^{-x}}{(1 + e^{-x})^2} \tag{6}$$

Uma questão relacionada com a utilização da função Sigmoide é o chamado Problema da Perda de Gradiente, ou Problema do Desaparecimento do Gradiente (do inglês, *Vanishing Gradient Problem*). Esse problema consiste em uma situação em que uma rede neural profunda se torna incapaz de realizar o processo de retropropagação, ou seja, de atualizar os pesos e vieses dos neurônios a partir da camada de saída na direção das primeiras camadas internas. Dessa forma, é dito que o sinal de erro durante a retropropagação diminui ou aumenta de maneira exponencial em função da profundidade da rede (HAYKIN, 2009). Isso ocorre porque a derivada da função Sigmoide assume valores sempre menores que um, e seu valor máximo é assumido em x = 0, como pode ser observado na Figura 7, onde a derivada vale 0,25. Dessa forma, uma rede neural profunda contendo somente funções de ativação sigmoide logística tende a apresentar valores de gradiente cada vez menores ao longo das camadas durante a retropropagação, até alcançar um conjunto de valores de atualização que, independentemente da taxa de aprendizado da rede, tornaria a rede incapaz de atualizar seus pesos e vieses, ficando completamente estagnada e, consequentemente, cessando o processo de aprendizado.

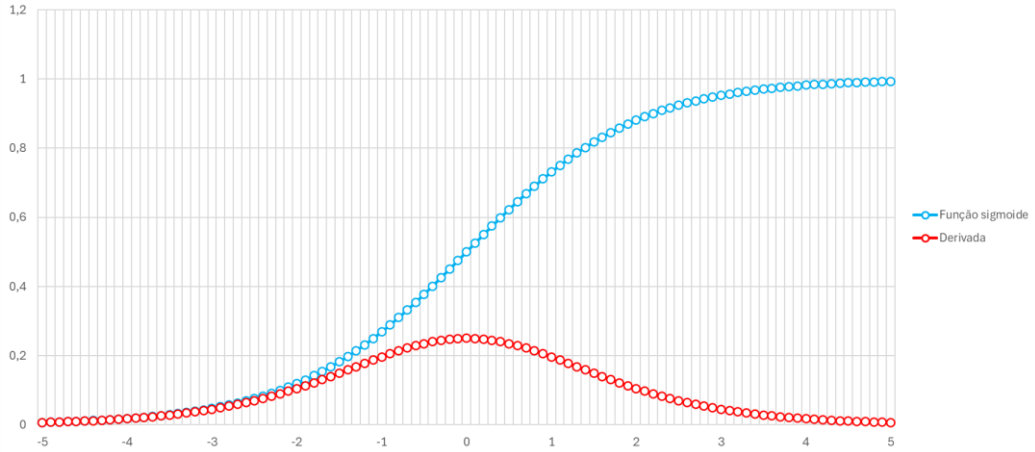

Figura 7. Função de ativação sigmoide logística e sua derivada (De autoria própria)

2. **Tangente hiperbólica:** é uma função contínua e diferenciável em todo o seu domínio (-∞, +∞) que realiza a transformação de um valor real como valor de entrada para um valor dentro do intervalo [-1, 1] na saída. Quanto maior o valor de entrada (mais positivo), mais próximo o valor de saída será de um, enquanto quanto menor o valor de entrada (mais negativo), mais perto a saída será de menos um. Matematicamente, ela é representada pela equação (7), e sua derivada representada pela equação (8):

$$\psi(x) = tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{7}$$

$$\psi'(x) = sech^{2}(x) = \frac{4}{(e^{x} + e^{-x})^{2}} \tag{8}$$

O uso da função de ativação tangente hiperbólica traz como vantagem, por exemplo, o fato de a saída da função ser centralizada no zero, diferentemente da função Sigmoide, que não possui uma saída simétrica em torno do zero, o que tende a dificultar o aprendizado da rede, tornando-a mais instável. Dessa forma, se torna mais fácil realizar um mapeamento dos valores de saída da camada como sendo fortemente negativos (mais próximos de -1), neutros (próximos de 0) ou fortemente positivos (mais próximos de 1). Essa centralização dos dados facilita o aprendizado para a camada seguinte. Assim como a função sigmoide, a função tangente hiperbólica também enfrenta o problema do desaparecimento do gradiente, pois sua derivada só assume valores significativos para valores de

entrada próximos de zero, e fora desse intervalo a derivada rapidamente se aproxima de zero. Além disso, o gradiente da função tangente hiperbólica é muito mais íngreme quando comparado com a função sigmoide, como pode ser observado na Figura 8.

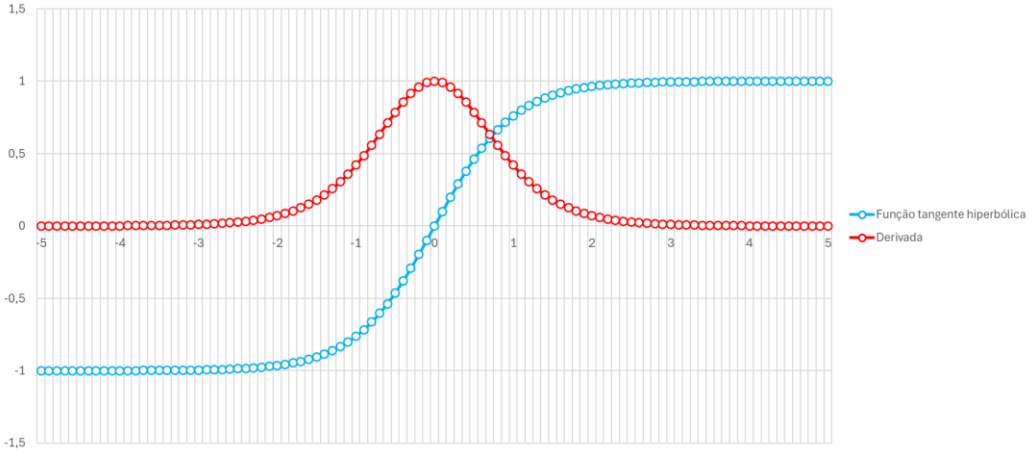

Figura 8. Função de ativação tangente hiperbólica e sua derivada (De autoria própria)

Dessa forma, ao se analisar a similaridade entre as funções sigmoide e tangente hiperbólica surge um questionamento de qual é a mais adequada dentre as duas a ser utilizada durante a definição da arquitetura de uma rede neural. Na prática, a função sigmoide é geralmente usada como função de saída em modelos de classificação, onde o objetivo é determinar a probabilidade de uma determinada classe ser selecionada, como no caso do problema de classificação prévia de *Trips* em uma usina nuclear, onde uma sequência de eventos será classificada dentro de um universo de 2 classes possíveis, configurando um problema de classificação binária: *Trip* (1) ou Não-*Trip* (0). Como a probabilidade de o modelo escolher uma classe em detrimento da outra está dentro do intervalo [0, 1], a função sigmoide se encaixa perfeitamente neste caso, pois limita a saída do neurônio de saída a esse intervalo numérico. Nesse contexto, caso o valor de saída retorne um número mais próximo de 0, a rede classificaria a sequência como Não-*Trip*, enquanto um valor mais próximo de 1 como *Trip*.

Por mais que tanto a função sigmoide quanto a tangente hiperbólica enfrentem o problema do desaparecimento do gradiente, a função tangente

hiperbólica é centrada no zero, e os valores dos gradientes não estão restritos a somente valores positivos como ocorre na sigmoide. Com isso, a não linearidade da tangente hiperbólica acaba sendo preferível em relação à não linearidade da sigmoide, principalmente quando se trata de funções de ativação em camadas internas de redes neurais profundas, onde dependendo da complexidade da rede pode ser difícil manter o gradiente durante o mecanismo de retropropagação. Ela também é utilizada como função de ativação da entrada das células LSTM, o que ajuda a mitigar problemas relacionados a desvios em valores médios já que a função é centrada em zero e adiciona uma não linearidade na entrada da célula (HOCHREITER & SCHMIDHUBER, 1997).

3. **Unidade Linear Retificada (ReLU):** é uma função contínua, porém não derivável em x = 0, que corresponde a uma variação da função linear tradicional. Ela representa uma simplificação em relação às funções de ativação sigmoide ou tangente hiperbólica, permitindo maior eficiência computacional e melhor propagação do gradiente, pois a função ativa os neurônios apenas quando a saída da transformação linear é positiva. Uma de suas principais vantagens é a mitigação do problema do desaparecimento do gradiente, comum em funções de ativação como sigmoide e tangente hiperbólica. Além disso, a ReLU acelera a convergência do treinamento ao mínimo da função de erro devido à sua capacidade de manter gradientes grandes para entradas positivas, evitando saturação do aprendizado. Matematicamente, ela é representada pela equação (9), e sua derivada representada pela equação (10):

$$ReLU(x) = \max(0, x) \tag{9}$$

$$ReLU'(x) = \begin{cases} 0 \; \forall \; x \leq 0 \\ 1 \; \forall \; x > 0 \end{cases} \tag{10}$$

Um dos problemas relacionados à utilização da função ReLU é definido como “problema da morte dos neurônios da ReLU”. Esse problema ocorre devido ao fato da derivada da função ReLU ser zero para todo valor de entrada negativo, vide equação (10) e Figura 9, fazendo com que o gradiente se torne nulo para as conexões formadas a partir dos neurônios afetados. Devido a essa questão, durante

o processo de retropropagação, os pesos e vieses para esses neurônios não são mais atualizados, deixando de contribuir para o treinamento, sendo chamados de neurônios mortos.

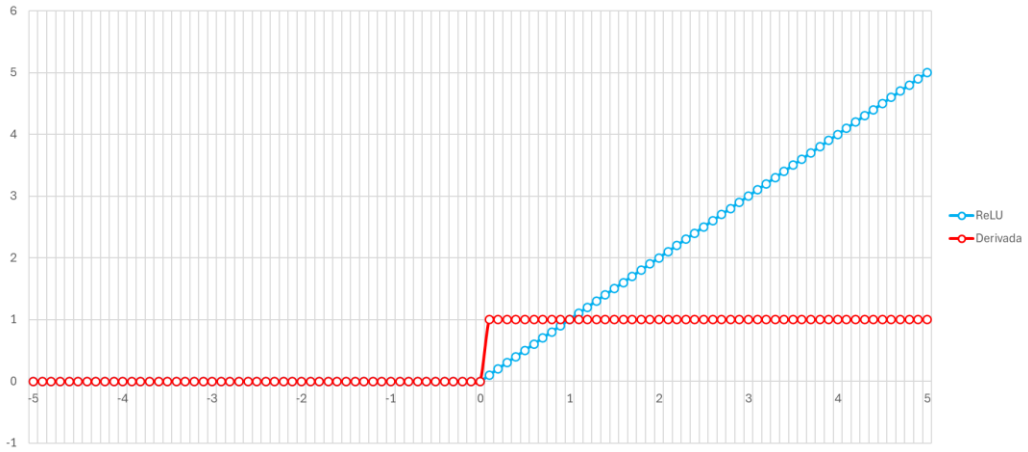

Figura 9. Função de ativação ReLU e sua derivada (De autoria própria)

Outro fator que pode levar a morte dos neurônios que utilizam a função de ativação ReLU é quando há um enorme gradiente passando pelas camadas da rede, que ocorre principalmente durante o início da fase de treinamento e/ou quando a taxa de aprendizado da rede é suficientemente alta para causar grandes distorções no aprendizado. Nesse caso, o neurônio pode acabar assumindo um alto peso negativo e um viés elevado associados a ele, fazendo com que o gradiente que passa por esse neurônio sempre seja nulo, independentemente do valor de entrada. Para mitigar o problema da morte dos neurônios associado à função ReLU, algumas variantes dessa função foram propostas na literatura, como:

a. Função de ativação *Leaky ReLU* (LReLU) (MAAS *et al.*, 2013), que permite especificar uma pequena inclinação negativa para valores negativos de entrada, evitando que o gradiente se torne nulo e permitindo manter neurônios ativos mesmo para entradas negativas.
b. Função de ativação *Parametric ReLU* (PReLU) (HE *et al.*, 2015), onde a inclinação para valores negativos é aprendida pelo modelo durante o treinamento.
c. Função de ativação *Exponential Linear Unit* (ELU) (CLEVERT *et al.*, 2015), que em vez de utilizar uma inclinação linear para valores

negativos, usa uma função exponencial, o que torna a transição mais suave.

4. **Softmax:** é uma função matemática amplamente utilizada em redes neurais para problemas de classificação multiclasse. Ela transforma um vetor de valores reais, tipicamente a intensidade dos sinais provenientes da camada anterior, em uma distribuição de probabilidade, onde cada valor do vetor de entrada é convertido em uma probabilidade, e a soma de todas as probabilidades do vetor é igual a 1. Ela é definida matematicamente pela equação (11):

$$softmax(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \tag{11}$$

onde *z* é o vetor de entrada com *K* elementos e z_i representa cada elemento desse vetor, de forma que uma função exponencial é aplicada a cada elemento z_i do vetor de entrada e esses valores são normalizados ao dividi-los pela soma de todas as exponenciais dos valores do vetor de entrada. O resultado é que cada saída da função *softmax* está no intervalo (0, 1) e a soma de todas as saídas é igual a 1, permitindo que sejam interpretadas como probabilidades. Essa função é especialmente útil na camada de saída de modelos de classificação multiclasse, pois fornece uma maneira eficaz de determinar a probabilidade de cada classe dada uma entrada. Dessa forma, a aplicação da função *softmax* na camada de saída de uma rede neural irá resultar em valores que representam a probabilidade de pertencimento a cada classe com base em um vetor de entrada, permitindo ao modelo realizar a classificação com base na maior probabilidade, servir como parâmetro para o desenvolvimento de um modelo de resposta "não-sei" (SANTOS *et al.*, 2021), utilização de modelos de Inteligência Artificial Explicável (XAI) ou permitir ao usuário tomar uma decisão embasada nas probabilidades retornadas pelo modelo.

Entre suas propriedades, pode-se destacar a sua capacidade inerente de normalização dos dados, que garante que todos os resultados estejam entre 0 e 1 e que sua soma seja 1. Além disso, a *softmax* é sensível a valores altos, amplificando as diferenças entre os valores de entrada, de forma que um valor maior no vetor resultará em uma maior probabilidade na saída correspondente.

2.4 REDES NEURAIS PROFUNDAS

As Redes Neurais Profundas (DNNs) são uma subárea dos modelos de Aprendizado de Máquina, denominada *Deep Learning*, projetadas para capturar e modelar padrões complexos em dados. A principal motivação para usar DNNs é sua capacidade de aprender representações hierárquicas e sofisticadas dos dados, algo que as técnicas tradicionais de Aprendizado de Máquina não conseguem realizar com a mesma profundidade e precisão. Essa maior capacidade de aprendizado se deve, principalmente, à presença de várias camadas de neurônios artificiais interconectados e à utilização de funções de ativação retificadoras, se tornando capaz de identificar padrões mais facilmente, diminuindo a necessidade de realizar um pré-processamento rebuscado dos dados e prevenindo problemas como o desaparecimento ou explosão do gradiente (LECUN *et al.*, 2015).

O conceito de redes neurais artificiais remonta à década de 1940, mas a popularização das DNNs ocorreu principalmente a partir dos anos 2000, quando avanços significativos em poder computacional e disponibilidade de grandes conjuntos de dados começaram a permitir o treinamento eficaz dessas redes. A partir de 2012, com o sucesso do modelo AlexNet em competições de visão computacional, o interesse por redes neurais profundas disparou, levando a uma rápida adoção em diversas áreas da inteligência artificial (KRIZHEVSKY *et al.*, 2012; SCHULZ & BEHNKE, 2012).

As DNNs oferecem certas vantagens sobre métodos tradicionais de Inteligência Artificial, sendo uma das principais a sua capacidade de aprendizado, de forma que elas podem aprender automaticamente características complexas dos dados sem a necessidade de extração manual ou de pré-processamento rebuscado, que é uma limitação comum em algoritmos mais simples. Além disso, as DNNs geralmente apresentam desempenho superior em tarefas que envolvem grandes volumes de dados e complexidade, superando outros métodos, como árvores de decisão ou regressão linear, sendo especialmente eficazes em reconhecimento de padrões e classificação (LECUN *et al.*, 2015).

Além disso, as DNNs possuem ampla aplicabilidade, sendo utilizadas em uma vasta gama de problemas, que incluem desde tarefas de Visão Computacional até Processamento de Linguagem Natural, de maneira que sua flexibilidade permite uma

adaptação eficiente a diferentes tipos de dados, tornando-as ferramentas altamente versáteis. Outro aspecto relevante é a escalabilidade dessas redes, que se beneficia diretamente do aumento da capacidade computacional e de avanços tecnológicos, como o uso de GPUs (*Graphics Processing Units*), possibilitando o treinamento de modelos em conjuntos de dados significativamente maiores do que aqueles suportados por métodos tradicionais. Adicionalmente, existem arquiteturas específicas que foram desenvolvidas para atender a diferentes demandas, como as Redes Convolucionais (CNNs), amplamente empregadas em problemas relacionados a imagens, e as Redes Recorrentes (RNNs), utilizadas para o processamento de dados sequenciais e séries temporais, proporcionando otimizações direcionadas a tarefas particulares (LECUN et al., 2015; SCHMIDHUBER, 2022).

As funções de ativação também desempenham um papel fundamental nas DNNs, pois introduzem não-linearidades no modelo, permitindo que ele aprenda padrões complexos e representações de alto nível dos dados. A função ReLU, amplamente utilizada nas DNNs, ajuda a mitigar o problema do desaparecimento do gradiente pois, diferentemente das funções sigmoide e tangente hiperbólica, não apresenta regiões em que a derivada assume valores próximos de zero para entradas positivas, o que contribui para a preservação do gradiente durante o treinamento e facilita o aprendizado em redes profundas (LECUN *et al.*, 2015). Por outro lado, funções de ativação como a sigmoide e a tangente hiperbólica continuam sendo aplicadas em contextos específicos. A função sigmoide, por exemplo, é frequentemente utilizada como função de ativação na camada de saída em problemas de classificação binária, enquanto a função tangente hiperbólica é empregada como função de ativação nas entradas das células das camadas LSTM, devido à sua capacidade de representar estados com valores centrados em zero (HOCHREITER & SCHMIDHUBER, 1997; SCHMIDHUBER, 2022).

O aprendizado por representação é um conjunto de métodos que permite que uma máquina receba dados brutos e descubra automaticamente as representações necessárias para detecção ou classificação. Métodos de aprendizado profundo são métodos de aprendizagem de representações com múltiplos níveis de representação, obtidos pela composição de módulos simples, porém não lineares, que transformam a representação em um nível (começando com a entrada bruta) em uma representação em um nível mais

alto e ligeiramente mais abstrato. Com a composição de um número suficiente dessas transformações, funções muito complexas podem ser aprendidas (LECUN *et al.*, 2015).

Para tarefas de classificação, camadas superiores de representação amplificam aspectos da entrada que são importantes para a discriminação e suprimem variações irrelevantes. Por exemplo, uma imagem é representada por uma matriz de valores de *pixels*, e as características aprendidas na primeira camada de representação geralmente indicam a presença ou ausência de bordas em orientações e localizações específicas na imagem. A segunda camada tipicamente detecta padrões ao identificar arranjos específicos de bordas, independentemente de pequenas variações nas posições das bordas. A terceira camada pode combinar padrões em combinações maiores que correspondem a partes de objetos conhecidos, e camadas subsequentes detectariam objetos como combinações dessas partes. O aspecto chave do aprendizado profundo é que essas camadas de características não são projetadas por engenheiros humanos: elas são aprendidas a partir de dados usando um procedimento de aprendizado de propósito geral (LECUN *et al.*, 2015).

Existem diversas arquiteturas de DNNs descritas na literatura, como as redes *feedforward*, que são amplamente utilizadas para tarefas de classificação e regressão; CNNs, que se destacam em tarefas de extração de características de dados estruturados, como imagens; LSTMs, que são particularmente úteis em previsão de séries temporais; Autoencoder são uma outra variação das DNNs, sendo utilizadas para redução de dimensionalidade, reconstrução de dados e detecção de anomalias; e Transformers, amplamente utilizadas em tarefas de NLP, como tradução, geração de texto e classificação de documentos, o que pode ser extremamente útil na análise de eventos críticos em sistemas de monitoramento de usinas nucleares baseados em sequências de texto.

2.5 EXEMPLOS DE ARQUITETURAS DE REDES NEURAIS

2.5.1 REDES NEURAIS FEEDFORWARD

Em uma rede neural estruturada na forma de camadas, os neurônios são organizados em diferentes níveis de maneira sequencial, de forma que os sinais que

chegam aos neurônios de uma camada são ponderados e transmitidos para os neurônios da camada seguinte. Na forma mais simples de uma rede neural em camadas, há uma camada de entrada composta por nós que recebem as informações iniciais e as projetam diretamente para uma camada de saída de neurônios (nós de computação), mas não o contrário, ou seja, os neurônios da camada de saída não projetam informação nas camadas anteriores, caracterizando, portanto, uma arquitetura estritamente do tipo *feedforward* (HAYKIN, 2009). Dessa forma, o aprendizado ocorre pela alteração dos pesos das conexões após o processamento de cada conjunto ou lote de dados, com base na mensuração do erro na saída da rede em comparação com o resultado esperado, sendo este um exemplo de aprendizado supervisionado, realizado por meio do método de retropropagação.

Além disso, as redes neurais *feedforward* podem se distinguir pela presença de uma ou mais camadas ocultas, cujos nós de computação são denominados neurônios ou unidades ocultas. O termo “oculto” refere-se ao fato de que essa parte da rede neural não é diretamente visível a partir da entrada ou da saída da rede. A função dos neurônios ocultos é atuar como um intermediário entre a entrada e a saída da rede de maneira a identificar padrões nos dados que outrora seriam difíceis ou impossíveis de serem obtidos com somente 2 camadas (uma de entrada e uma de saída). Ao adicionar uma ou mais camadas ocultas, a rede é capaz de apresentar um maior grau de abstração, extraindo informações de ordem superior de sua entrada. Em termos mais gerais, a rede adquire uma perspectiva global, apesar de sua conectividade local, graças ao conjunto extra de conexões sinápticas e à dimensão adicional das interações neurais (CHURCHLAND & SEJNOWSKI, 1992; HAYKIN, 1999).

Cada neurônio em uma camada recebe sinais das camadas anteriores, processa essas informações, ponderando-as através dos pesos sinápticos, limita a amplitude do sinal de saída por intermédio de uma função de ativação e transmite esse sinal para os neurônios da camada seguinte. As funções de ativação são essenciais para introduzir uma não-linearidade na rede, permitindo que ela aprenda representações complexas dos dados. Funções como ReLU, sigmoide e tangente hiperbólica são comumente usadas para garantir que o modelo seja capaz de capturar interações complexas entre as variáveis.

Os nós de origem na camada de entrada da rede fornecem os elementos correspondentes do padrão de ativação (vetor de entrada), que constituem os sinais de entrada aplicados aos neurônios na segunda camada, ou seja, a primeira camada oculta. Os sinais de saída da segunda camada são usados como entradas para a terceira camada, e assim por diante, para o restante da rede, seguindo sempre um fluxo de informações no sentido entrada → camadas ocultas → saída. Tipicamente em uma rede *feedforward*, os neurônios em cada camada da rede recebem como entrada apenas os sinais de saída da camada anterior, o que não é verdade para outras arquiteturas de rede, como as recorrentes. O conjunto de sinais de saída dos neurônios na camada de saída da rede constitui a resposta global da rede ao padrão de ativação fornecido pelos nós de origem na camada de entrada, representando a inferência realizada pela rede após o ajuste de seus pesos, ou seja, seu aprendizado (HAYKIN, 2009). Na Figura 10 é ilustrado um exemplo de uma arquitetura de rede neural *feedforward* multicamada, neste caso com uma única camada oculta.

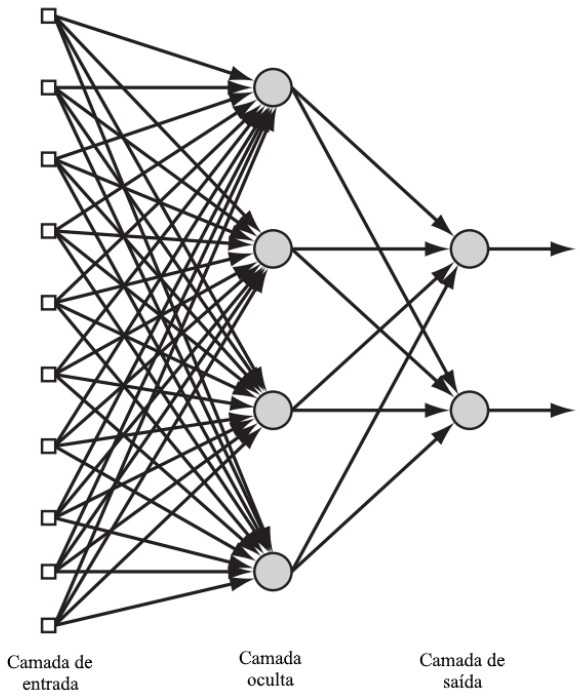

Figura 10. Rede neural *feedforward* com uma camada oculta (HAYKIN, 2009)

A rede neural ilustrada na Figura 10 é classificada como uma rede totalmente conectada (ou densamente conectada), pois cada neurônio de uma camada está conectado a todos os neurônios da camada seguinte. Essa característica implica que cada nó (ou

unidade de processamento) em uma camada transmite informações para todos os nós da camada adjacente, garantindo que todas as combinações possíveis de entrada e saída sejam consideradas na modelagem. Caso algumas das conexões entre as camadas estejam ausentes, a rede passa a ser denominada parcialmente conectada. Redes parcialmente conectadas reduzem a densidade de conexões entre os neurônios, o que pode ser vantajoso em termos de eficiência computacional e mitigação de problemas como sobreajuste, ou *overfitting*. Um exemplo clássico de redes parcialmente conectadas são as redes convolucionais (CNNs), onde cada nó é conectado apenas a uma região específica da camada anterior, em vez de todos os nós, otimizando o aprendizado para dados espaciais, como imagens. Essa estrutura de conectividade tem implicações diretas na complexidade e capacidade de aprendizado da rede, de forma que redes totalmente conectadas, como a apresentada na Figura 10, possuem maior expressividade devido ao número elevado de parâmetros treináveis, mas exigem maior poder computacional e podem ser mais suscetíveis a problemas de *overfitting* e nível de generalização quando o tamanho do conjunto de dados não é suficientemente grande (HAYKIN, 2009; LECUN *et al.*, 2015).

O treinamento das redes neurais *feedforward* é realizado por meio de um processo iterativo denominado retropropagação de forma que, durante o treinamento, os pesos das conexões entre os neurônios são ajustados com base no erro da previsão gerada pela rede. Enquanto a informação trafega através da rede a partir da entrada em direção às camadas ocultas, a retropropagação segue o caminho inverso, calculando o erro da camada de saída e o propaga de volta pelas camadas ocultas, ajustando os pesos para minimizar a função de custo, geralmente utilizando o algoritmo de Gradiente Descendente Estocástico e suas variações, como por exemplo os otimizadores Adam (KINGMA & BA, 2014) e RMSProp (TIELEMAN & HINTON, 2012).

2.5.2 REDES NEURAIS RECORRENTES

As Redes Neurais Recorrentes (RNNs) são uma classe de redes neurais projetadas para trabalhar com dados sequenciais (HAYKIN, 1999). Diferente das redes *feedforward*, onde os dados fluem de uma camada para outra de forma linear, nas RNNs há conexões recorrentes que permitem que o modelo “lembre” das entradas anteriores e as use para influenciar as previsões futuras. Isso as torna particularmente eficazes para modelar

dependências temporais e capturar padrões em dados que possuem sequências ou ordem temporal, como eventos de falhas em sistemas industriais (HOCHREITER & SCHMIDHUBER, 1997; SCHMIDHUBER, 2022).

Assim como as redes neurais *feedforward*, as RNNs utilizam dados de treinamento para promover o seu aprendizado. Elas se distinguem por sua "memória" e pela presença de pelo menos um loop de *feedback*, pois utilizam informações retidas anteriormente para influenciar a entrada e a saída atuais. Enquanto as redes neurais profundas tradicionais assumem que as entradas e saídas são independentes entre si, a saída das redes neurais recorrentes depende dos elementos anteriores dentro da sequência. Embora eventos futuros também possam ser úteis para determinar a saída de uma sequência dada, redes neurais recorrentes unidimensionais não são capazes de considerar esses eventos em suas previsões (HAYKIN, 2009).

Diz-se que existe *feedback* em um sistema dinâmico quando a saída de um elemento no sistema influencia, parcial ou totalmente, a entrada aplicada a esse mesmo elemento, criando assim um ou mais caminhos fechados para a transmissão de sinais ao redor do sistema. Esse conceito é amplamente observado em sistemas biológicos, ocorrendo, por exemplo, em praticamente todas as partes do sistema nervoso de animais (FREEMAN, 1975). No contexto das RNNs, o mecanismo de *feedback* desempenha um papel crucial, sendo responsável pela formação de laços de realimentação que permitem a retenção de informações ao longo do tempo, sendo esses *loops* de *feedback* fundamentais para a persistência da memória interna das camadas da rede, possibilitando o aprendizado e a modelagem de dados sequenciais ou temporais ao longo das iterações do treinamento (HAYKIN, 2009).

Por exemplo, uma RNN pode ser composta em uma única camada de neurônios, na qual cada neurônio transmite seu sinal de saída de volta às entradas de todos os outros neurônios dessa mesma camada. Na Figura 11, apresenta-se uma representação esquemática de uma RNN com uma camada oculta, evidenciando as conexões de *feedback* que se originam tanto dos neurônios ocultos quanto dos neurônios da camada de saída. Essas conexões de realimentação são essenciais para a capacidade das RNNs de modelar dependências temporais e padrões sequenciais em dados, sendo fundamentais

para tarefas como previsão de séries temporais, processamento de linguagem natural e reconhecimento de padrões (SCHMIDHUBER, 2022).

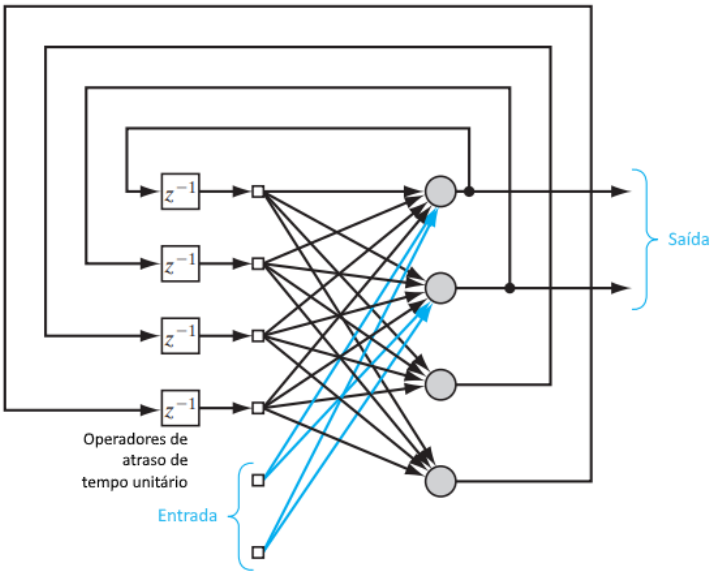

Figura 11. Rede neural recorrente com a presença de uma camada oculta (HAYKIN, 2009)

Outra característica distintiva das RNNs é o compartilhamento de parâmetros entre as camadas da rede ao longo do tempo. Diferentemente das redes *feedforward*, nas quais cada conexão entre os neurônios possui pesos independentes, as RNNs utilizam os mesmos parâmetros de peso em cada etapa temporal, tornando-as mais eficientes para lidar com sequências de dados. Apesar dessa diferença estrutural, os pesos das RNNs ainda são ajustados durante o treinamento por meio dos processos de retropropagação e descida de gradiente, permitindo que a rede aprenda padrões complexos e relações temporais nos dados, assim como ocorre em redes *feedforward* no contexto de aprendizado (HAYKIN, 2009).

O bloco fundamental de uma RNN é a unidade recorrente, que mantém um estado oculto, funcionando como uma forma de memória interna, sendo esse estado oculto atualizado a cada passo de tempo com base na entrada atual e no estado oculto do instante anterior. Esse mecanismo de realimentação permite que a rede utilize informações provenientes de entradas passadas e incorpore esse conhecimento no processamento atual, conferindo às RNNs a capacidade de modelar dependências temporais em dados

sequenciais. Na Figura 12, apresenta-se uma representação de uma RNN com múltiplas camadas ocultas, evidenciando que a saída de cada camada em um determinado instante de tempo $z - 1$ é reutilizada como entrada pela mesma camada no próximo instante, z. Assim, a cada passo de tempo, as entradas de cada camada consistem tanto nos sinais provenientes da camada anterior quanto nos sinais gerados por ela própria no instante anterior. Esse comportamento permite que informações passadas sejam preservadas ao longo do tempo, evitando que sejam rapidamente descartadas. Em resumo, cada neurônio em uma RNN recebe entradas tanto da camada anterior quanto da própria camada, o que possibilita à rede armazenar e utilizar informações históricas de forma contínua, um aspecto essencial para a modelagem de dados temporais (HAYKIN, 2009).

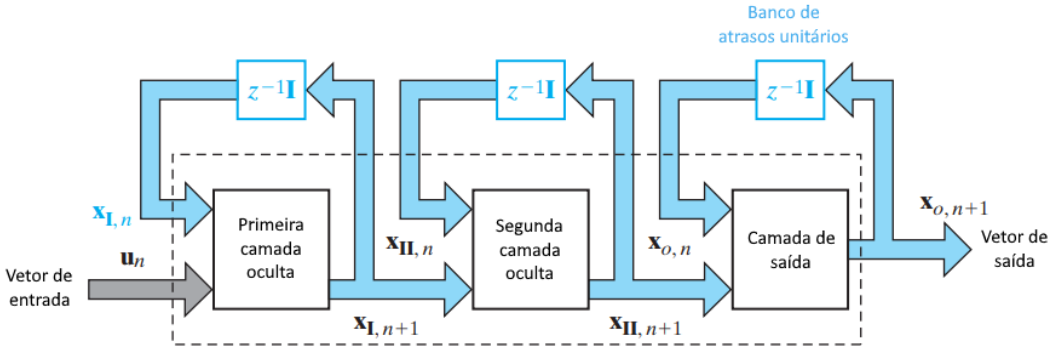

Figura 12. Rede neural recorrente com múltiplas camadas ocultas (HAYKIN, 2009)

A presença de *loops* de *feedback* tem um impacto profundo na capacidade de aprendizado da rede e em seu desempenho, pois nesse caso os neurônios de uma camada são capazes de se comunicar com os neurônios da camada seguinte e consigo mesmos, permitindo que estados anteriores de cada neurônio sejam levados em consideração durante a atualização dos pesos neuronais. Além disso, os *loops* de *feedback* envolvem o uso de ramos particulares compostos por elementos de atraso de unidade de tempo (denotados por z^{-1}), o que resulta em um comportamento dinâmico não linear, assumindo que a rede neural contenha unidades recorrentes não lineares (HAYKIN, 2009).

As RNNs utilizam o algoritmo de retropropagação através do tempo (BPTT) para calcular os gradientes necessários ao ajuste dos pesos da rede, sendo esse método uma extensão da retropropagação tradicional, adaptada especificamente para lidar com dados sequenciais. Assim como na retropropagação convencional, o BPTT ajusta os parâmetros do modelo por meio do cálculo dos erros propagados da camada de saída até a camada de

entrada, de forma que esses ajustes são realizados com o objetivo de minimizar a função de erro e aprimorar a exatidão da rede. A principal distinção entre o BPTT e a retropropagação tradicional está na maneira como os erros são acumulados. No BPTT, os erros são somados ao longo de cada passo de tempo, refletindo a natureza sequencial das RNNs e o compartilhamento de parâmetros entre os instantes temporais. Em contrapartida, nas redes *feedforward*, não há necessidade de acumular erros, uma vez que os parâmetros não são compartilhados entre camadas. Essa característica do BPTT é essencial para permitir que as RNNs capturem dependências temporais nos dados, mas também aumenta a complexidade computacional, especialmente em sequências longas, devido ao risco de problemas como o desaparecimento ou explosão do gradiente (HOCHREITER & SCHMIDHUBER, 1997; HAYKIN, 2009)

Durante o processo de treinamento, conforme mencionado, as RNNs tendem a enfrentar dois problemas conhecidos: a explosão do gradiente e desaparecimento do gradiente. Esses problemas estão relacionados ao tamanho do gradiente, que representa a inclinação da função de perda em relação aos pesos da rede. Quando o gradiente se torna muito pequeno, a atualização dos pesos sinápticos é insignificante, o que impede a rede de continuar aprendendo e a faz ficar estagnada em mínimos locais ou pontos subótimos. Por outro lado, a explosão do gradiente ocorre quando os gradientes assumem valores excessivamente grandes, tornando o modelo instável e dificultando a convergência para um ponto de mínimo. Para mitigar essas limitações, foram desenvolvidas arquiteturas baseadas em RNNs, como as redes *Long Short-Term Memory* (LSTM) (HOCHREITER & SCHMIDHUBER, 1997) e *Gated Recurrent Units* (GRUs) (CHO *et al.*, 2014). Essas arquiteturas introduziram mecanismos de controle por meio de portões que regulam o fluxo de informações dentro da rede, permitindo que a rede mantenha ou descarte informações ao longo do tempo, facilitando a aprendizagem de padrões e dependências temporais em sequências longas e reduzindo os efeitos do desaparecimento ou explosão do gradiente. As LSTMs são particularmente eficazes em modelar dependências de longo prazo, devido à sua capacidade de preservar informações relevantes em memórias de longa duração. As GRUs, por sua vez, apresentam uma estrutura mais simples e computacionalmente eficiente, enquanto ainda oferecem um alto desempenho em tarefas que envolvem dados temporais (SCHMIDHUBER, 2022).

2.5.3 REDES LSTM

As redes *Long Short-Term Memory* (HOCHREITER & SCHMIDHUBER, 1997), ou LSTM, são uma arquitetura de RNNs desenvolvidas para superar algumas limitações das RNNs tradicionais, especialmente no que diz respeito à captura de dependências temporais de longo prazo, sendo criadas para resolver problemas como o desaparecimento e a explosão do gradiente, que frequentemente ocorrem em RNNs mais simples quando lidam com sequências longas de dados. A principal inovação das LSTMs é a introdução de um mecanismo de memória e a introdução de portões lógicos que permitem armazenar e acessar informações ao longo de períodos prolongados, facilitando o aprendizado em tarefas que exigem uma compreensão mais profunda do contexto temporal.

A ideia original da LSTM introduziu a célula de memória escalar como uma unidade central de processamento e armazenamento, que evita o problema do desaparecimento do gradiente através do Carrossel de Erro Constante (CEC), responsável pela atualização do estado da célula, representado pela equação (12). A arquitetura das LSTMs é composta por unidades que incluem três portas principais: a porta de entrada, a porta de esquecimento (GERS e*t al.*, 2000) e a porta de saída. Essas portas controlam o fluxo de informações dentro da rede, permitindo que a LSTM decida quais informações devem ser retidas ou descartadas. O CEC mantém a ativação interna (chamada de estado da célula) com uma conexão recorrente de peso fixo igual a 1, que pode ser redefinida pelo portão de esquecimento, enquanto as portas de entrada e saída escalonam a entrada e a saída, respectivamente. Todas as portas são controladas pelo estado mantido, pela entrada da rede e pela ativação oculta do passo de tempo anterior (HOCHREITER & SCHMIDHUBER, 1997). Essa estrutura é fundamental para a eficácia das LSTMs em tarefas como tradução automática, previsão de séries temporais e reconhecimento de fala.

As regras de atualização da célula de memória da LSTM no passo de tempo *t* podem ser descritas pelas equações (12), (13), (14), (15), (16) e (17):

$$c_t = f_t c_{t-1} + i_t z_t \qquad \text{estado da célula} \qquad (12)$$

$$h_t = o_t \tilde{h}_t, \qquad \tilde{h}_t = \psi(c_t) \qquad \text{estado oculto} \qquad (13)$$

$$z_t = \varphi(\tilde{z}_t), \qquad \tilde{z}_t = w_z^T x_t + r_z h_{t-1} + b_z \qquad \text{entrada da célula} \qquad (14)$$

$$i_t = \sigma(\tilde{\imath}_t), \qquad \tilde{\imath}_t = w_i^T x_t + r_i h_{t-1} + b_i \quad \text{portão de entrada} \quad (15)$$

$$f_t = \sigma(\tilde{f}_t), \qquad \tilde{f}_t = w_f^T x_t + r_f h_{t-1} + b_f \quad \text{portão de esquecimento} \quad (16)$$

$$o_t = \sigma(\tilde{o}_t), \qquad \tilde{o}_t = w_o^T x_t + r_o h_{t-1} + b_o \quad \text{portão de saída} \quad (17)$$

Os vetores de peso w_z, w_i, w_f e w_o correspondem aos pesos de entrada entre as entradas x_t e a célula de entrada, a porta de entrada, a porta de esquecimento e a porta de saída, respectivamente. Os pesos r_z, r_i, r_f e r_o correspondem aos pesos recorrentes entre o estado oculto h_{t-1} e a célula de entrada, a porta de entrada, a porta de esquecimento e a porta de saída, respectivamente. Os termos de viés b_z, b_i, b_f e b_o são os correspondentes para cada uma dessas portas. Os termos φ e ψ são as funções de ativação para a entrada da célula e o estado oculto (tipicamente, tangente hiperbólica). ψ é usada para normalizar ou limitar o estado da célula, que seria ilimitado caso contrário. Além disso, todas as funções de ativação das portas são sigmoides.

De maneira geral, o funcionamento de uma célula LSTM é realizado através da atuação dos portões de entrada, esquecimento e saída sobre o fluxo de informações na célula da seguinte forma:

1. Primeiramente, o portão de esquecimento é responsável por decidir quais informações do estado anterior da célula, c_{t-1}, serão mantidas ou descartadas. As entradas para esse portão incluem o estado oculto anterior, h_{t-1}, e o novo elemento de entrada x_t, que são concatenados e processados por uma função sigmoide logística: $f_t = \sigma(\tilde{f}_t)$, onde $\tilde{f}_t = w_f^T x_t + r_f h_{t-1} + b_f$, vide equação (16). Como a função sigmoide logística normaliza o valor no intervalo [0,1], valores próximos a 1 indicam que a informação será preservada no estado da célula, enquanto valores próximos a 0 sugerem que a informação será esquecida pela célula LSTM.
2. Em seguida, o portão de entrada regula quais novas informações serão armazenadas no estado da célula. O processo é dividido em dois passos:
    a. As entradas h_{t-1} (estado oculto) e x_t (sinal de entrada) passam por uma função sigmoide logística que determina quais valores serão atualizados: $i_t = \sigma(\tilde{\imath}_t)$, onde $\tilde{\imath}_t = w_i^T x_t + r_i h_{t-1} + b_i$, vide equação (15).

b. Essas mesmas entradas, h_{t-1} (estado oculto) e x_t (sinal de entrada), são processadas por uma função tangente hiperbólica, que gera um vetor com os valores possíveis para serem adicionados ao estado da célula: $z_t = \varphi(\tilde{z}_t)$, onde $\tilde{z}_t = w_z^T x_t + r_z h_{t-1} + b_z$, vide equação (14).

3. O novo estado da célula, c_t, é atualizado com base nas informações do portão de esquecimento e do portão de entrada. Primeiro, aplica-se o esquecimento ao estado anterior da célula, e depois, somam-se os novos valores, ponderados pelo quanto foi decidido atualizá-los: $c_t = f_t c_{t-1} + i_t z_t$, vide equação (12).
4. Por fim, define-se o próximo estado oculto da célula LSTM, h_t. As entradas h_{t-1} e x_t passam por uma função sigmoide: $o_t = \sigma(\tilde{o}_t)$, onde $\tilde{o}_t = w_o^T x_t + r_o h_{t-1} + b_o$, vide equação (17). Depois disso, o novo estado da célula passa por uma função tangente hiperbólica: $\tilde{h}_t = \psi(c_t)$, vide equação (13). O resultado dessas duas funções é multiplicado para determinar quais informações o próximo estado oculto irá carregar: $h_t = o_t \tilde{h}_t$.

A Figura 13 apresenta uma representação visual dos passos envolvidos no fluxo de informações dentro de uma célula LSTM, destacando a interação entre os diferentes componentes que a compõem. É possível observar a presença dos três portões principais, o portão de entrada, o portão de esquecimento e o portão de saída, os quais desempenham papéis fundamentais na regulação do fluxo de informações ao longo do tempo. Além disso, a figura também ilustra a transmissão de informações no interior da célula, incluindo a aplicação do CEC, mecanismo responsável por preservar e atualizar o estado da célula. Sendo assim, os portões trabalham de forma coordenada para decidir quais informações devem ser adicionadas, descartadas ou mantidas na memória de longo prazo da célula, de forma que o portão de entrada controla quais novos dados serão incorporados ao estado da célula; o portão de esquecimento determina quais informações antigas devem ser descartadas; e o portão de saída regula quais partes do estado da célula serão transmitidas para a próxima etapa. Esses mecanismos permitem que a célula LSTM mantenha informações relevantes ao longo de longas sequências temporais, abordando de maneira eficiente o problema do desaparecimento e da explosão do gradiente (HAYKIN, 2009; SCHMIDHUBER, 2022).

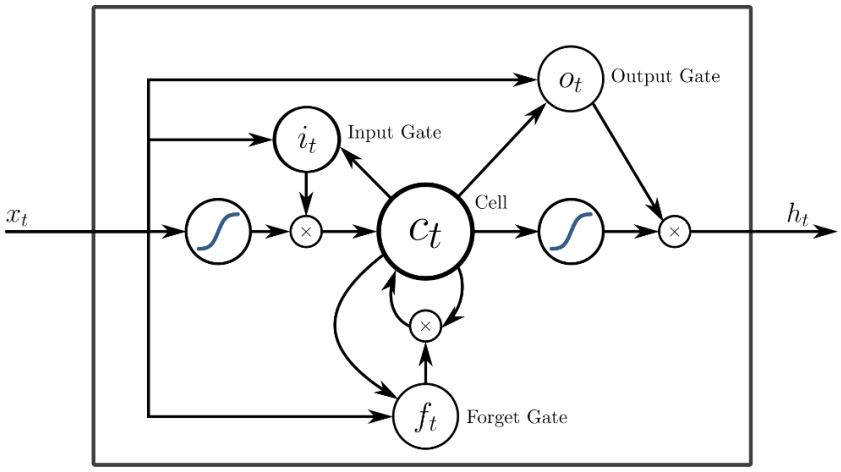

Figura 13. Representação de uma célula LSTM (GRAVES *et al.*, 2013)

Os pequenos círculos contendo o símbolo "×" na Figura 13 representam o produto de Hadamard, ou seja, uma multiplicação elemento a elemento (*element-wise multiplication*) entre as matrizes de entrada correspondentes. Já os círculos maiores contendo uma curva em forma de "S*"* azul representam a aplicação de uma função diferenciável (como a função sigmoide ou tangente hiperbólica) a uma soma ponderada. Com isso, a célula LSTM representada na Figura 13 controla o fluxo de informações utilizando portas que permitem reter ou descartar dados seletivamente, ajudando a preservar informações relevantes em longas sequências. O estado da célula (C_t) funciona como uma "memória de longo prazo", enquanto o estado oculto (h_t) é uma "memória de curto prazo" (HOCHREITER & SCHMIDHUBER, 1997).

As redes LSTM são utilizadas em diversas aplicações, como no processamento de linguagem natural, onde são capazes de capturar o contexto de sentenças de texto, resultando em traduções mais precisas e coerentes. Além disso, as LSTMs têm demonstrado um desempenho superior em várias aplicações práticas, sendo amplamente utilizadas em problemas que envolvem dados sequenciais e/ou com dependência temporal, como na previsão de séries temporais, geração de texto e tradução de documentos (BECK *et al.*, 2024). Sua capacidade de lidar com dados sequenciais complexos as torna uma ferramenta valiosa em muitos campos, principalmente no que diz respeito à Inteligência Artificial e Aprendizado Profundo, em tarefas como geração de texto no formato *seq2seq* (sequência-a-sequência) (GRAVES *et al.*, 2013; KARPATHY *et al.*, 2015), geração de texto manuscrito e replicação de caligrafia

(GRAVES *et al.*, 2013), tradução de sequências de texto (SUTSKEVER *et al.*, 2014) e geração de legendas para imagens (HOSSAIN *et al.*, 2019).

Apesar das inúmeras aplicações das redes LSTM, existem algumas limitações intrínsecas a este tipo de arquitetura (BECK *et al.*, 2024), como:

- A incapacidade de revisar decisões de armazenamento, ou seja, informações mais antigas em uma sequência são mais propensas de serem esquecidas. Dessa forma, a partir de um vetor de referência fornecido, a sequência deve ser escaneada sequencialmente para o vetor mais similar a fim de fornecer seu valor correspondente no final da sequência.
- Possui uma capacidade de armazenamento limitada, de maneira que a informação deve ser comprimida em estados de célula escalares. Esse é um dos principais motivos que impedem a paralelização do treinamento e inferência das redes LSTM, pois implica que a atualização do estado de cada célula deve ser processada sequencialmente entre as células.
- Falta de capacidade de paralelização devido à mistura de memória, de maneira que as conexões entre camadas ocultas para os estados entre um instante de tempo e outro são processadas sequencialmente.

Dessa forma, o advento das redes neurais Transformers (VASWANI *et al.*, 2017), com a proposta de utilização do mecanismo de auto-atenção (do inglês, *self-attention*) paralelizável (BAHDANAU *et al.*, 2014; VASWANI *et al.*, 2017), definiu uma nova arquitetura estado da arte na área do Aprendizado Profundo e Processamento de Linguagem Natural, superando as LSTMs em escala (BECK *et al.*, 2024). A principal vantagem dos Transformers em relação às LSTMs consta em sua capacidade de processar sequências inteiras de dados simultaneamente, ao invés de depender de cálculos sequenciais passo a passo em lotes de dados. Isso é possibilitado pelo mecanismo de *self-attention* e pela utilização de várias "cabeças" de atenção capazes de priorizar determinadas partes de uma sequência, que permite que o modelo capture dependências de longo alcance de forma eficiente, sem a necessidade de recorrer ao processamento sequencial característico das LSTMs. Assim, os Transformers não apenas superaram as LSTMs em várias tarefas de Processamento de Linguagem Natural, mas também

reduziram significativamente o tempo de treinamento e inferência (VASWANI *et al.*, 2017).

2.5.4 REDES NEURAIS TRANSFORMERS

2.5.4.1 MODELO ENCODER-DECODER

Os modelos *Sequence-to-Sequence* (*Seq2Seq*) (SUTSKEVER *et al.*, 2014; CHO *et al.*, 2014) com arquitetura Encoder-Decoder representam um marco importante no desenvolvimento de sistemas capazes de lidar com tarefas que envolvem entrada e saída sequenciais de comprimento variável, como tradução automática, sumarização de textos e previsão de eventos. Essa arquitetura em particular é composta por uma estrutura que processa e transforma uma sequência de entrada em uma representação intermediária (*encoding*), para então, ao utilizar essa representação, gerar a sequência de saída correspondente (*decoding*), podendo ser uma sequência de texto, previsão de eventos anômalos ou classificação de documentos, por exemplo.

A arquitetura *Seq2Seq* baseada em modelos Encoder-Decoder é composta por dois componentes principais: *encoder* e *decoder*. O *encoder* é responsável por processar a sequência de entrada e comprime suas informações em um vetor fixo de dimensão pré-definida, conhecido como vetor de contexto. Este vetor é uma representação abstrata da sequência de entrada e contém as informações necessárias para que o *decoder* gere a saída correspondente. O *decoder* gera, a partir do vetor de contexto, a sequência de saída de forma iterativa, elemento por elemento, utilizando as informações contidas no vetor de contexto atual e no histórico das predições feitas.

Na implementação inicial baseada em RNNs (BAHDANAU *et al*., 2015), o *encoder* e *decoder* eram compostos por Redes Neurais LSTM ou GRUs, que intrinsecamente são projetadas para processar sequências temporais e manter a memória de informações passadas. No entanto, essa abordagem apresentou limitações ao lidar com sequências muito longas, que acaba agravando a dificuldade do modelo em capturar dependências de longo prazo, um problema relacionado principalmente ao desaparecimento do gradiente que se torna mais evidente conforme o tamanho da sequência vai aumentando. A necessidade de se ter uma arquitetura capaz de manter as

dependências de longo prazo e relações contextuais entre diferentes sequências de dados é essencial em tarefas de identificação prévia de *Trip* e previsão através de longas sequências de eventos, onde a relação e contexto formado entre eventos passados e falhas futuras é crucial. Para contornar essa limitação, a introdução do mecanismo de atenção (BAHDANAU *et al*., 2015; LUONG *et al*., 2015) permitiu que o *decoder* acessasse diretamente os estados ocultos do *encoder* em cada etapa de atualização, melhorando significativamente o desempenho e capacidade de retenção de informação a longo prazo.

A introdução da arquitetura Transformer (VASWANI *et al*., 2017) revolucionou os modelos *Seq2Seq*, eliminando a necessidade de processamento sequencial das RNNs. O *encoder* e o *decoder* no Transformer utilizam o mecanismo de atenção, particularmente a atenção multi-cabeças (do inglês, *multi-head attention*), para capturar relações entre os elementos da sequência. A arquitetura do Transformer introduz uma estrutura empilhada de blocos que consiste em:

1. **Encoder**: Uma sequência de camadas compostas por atenção multi-cabeças e redes *feedforward* posicionadas com normalização.
2. **Decoder**: Uma estrutura similar ao *encoder*, com a adição de um mecanismo de atenção cruzada (do inglês, *cross-attention*), que permite acessar as representações do *encoder*.

Os Transformers se mostraram superiores em tarefas *Seq2Seq* devido à sua capacidade de capturar dependências globais em sequências de entrada e saída, enquanto preservam eficiência computacional, especialmente em comparação com RNNs (VASWANI *et al*., 2017). No contexto deste trabalho, a arquitetura *Seq2Seq* foi adaptada para prever o próximo evento em sequências operacionais de sistemas nucleares (geração de texto), além de processar sequências de dados de operação da usina para prever desarmes do reator. Dessa forma, o *encoder* recebe os dados temporais, como eventos de alarme ou medições operacionais, e gera um vetor de contexto que resume a sequência de entradas. O *decoder* então gera uma previsão sobre a evolução futura do sistema, ajudando a identificar comportamentos anômalos ou falhas em potencial.

Além disso, a introdução do mecanismo de atenção e os blocos de atenção multi-cabeças no modelo Encoder-Decoder têm melhorado significativamente o desempenho

desse modelo computacional (VASWANI *et al*., 2017). Com o mecanismo de atenção, o *decoder* pode focar em diferentes partes da entrada enquanto gera a saída, permitindo que o modelo capture de forma mais eficiente as partes relevantes de dados de longas sequências de eventos, como variáveis críticas, valores relevantes de variáveis, certas combinações de eventos, determinada ordem de ocorrência de eventos, a posição dos eventos dentro da sequência ou sequências de eventos que indicam uma falha iminente.

2.5.4.2 MECANISMO DE ATENÇÃO

O mecanismo de atenção nas redes neurais aborda a questão fundamental: "Dado um conjunto de dados de entrada, em quais partes dos dados devemos focar?" (TURNER, 2023). Para ilustrar esse conceito, é possível considerar o seguinte cenário hipotético: ao receber um livro sobre Aprendizado de Máquina, uma pessoa foi encarregada com a tarefa de extrair informações sobre a função de ativação ReLU. Nesse caso, um ser humano teria basicamente duas abordagens possíveis: a primeira abordagem consiste em ler o livro inteiro e, posteriormente, fornecer a resposta; a segunda abordagem envolve consultar o índice, localizar o capítulo sobre funções de ativação, direcionar-se diretamente à seção que explica a função ReLU e ler apenas as informações pertinentes ao contexto apresentado. Comparando as duas abordagens, é possível observar que o primeiro método demandaria um tempo considerável e resultaria em uma resposta possivelmente vaga, contendo muitas informações irrelevantes. Já a segunda estratégia seria mais eficiente e precisa, fornecendo exatamente o que foi solicitado e com menos propensão a erros (ANKIT, 2024).

A diferença entre as abordagens reside na focalização da atenção. Na primeira, não há foco específico, enquanto na segunda, a atenção é direcionada diretamente à seção relevante. Nas redes neurais Transformers, o mecanismo de atenção opera de maneira análoga, permitindo que a rede se concentre em partes específicas da entrada, atribuindo menor ênfase a informações menos relevantes (TURNER, 2023). Isso é particularmente útil em tarefas como tradução automática e geração de texto, onde, em cada etapa, a rede processa uma parte da sequência e gera uma saída correspondente. Durante esse processo, a atenção auxilia a rede a focar nas partes mais relevantes da sequência original (BAHDANAU *et al*., 2015). Por exemplo, ao gerar uma palavra em uma tradução, o

decodificador não precisa basear-se em todo o vetor de contexto proveniente do codificador, podendo se concentrar em partes específicas mais relevantes para uma palavra específica, ignorando informações menos importantes naquele momento, tornando a geração de textos mais precisa, eficiente e natural (TURNER, 2023).

Nos modelos *Seq2Seq* tradicionais, o vetor de contexto, responsável por carregar as informações da sequência de entrada, apresenta limitações ao lidar com sentenças longas devido, em grande parte ao problema do desaparecimento do gradiente, que dificulta a retenção de informações relevantes em sequências extensas. Essa dificuldade impacta negativamente a capacidade do modelo de gerar previsões precisas em tarefas que envolvem entradas de maior complexidade e extensão. A introdução do mecanismo de atenção (BAHDANAU *et al*., 2014) trouxe uma abordagem alternativa para superar essas limitações, transformando significativamente o desempenho dos modelos *Seq2Seq*, especialmente em tarefas como tradução automática e geração de texto.

O modelo de atenção permite que o decodificador se concentre em partes específicas da sequência de entrada durante a geração de cada elemento da saída, em vez de depender unicamente de uma representação compactada da sequência completa. Nos modelos *Seq2Seq* convencionais, o codificador transmitia ao decodificador apenas o estado oculto final, que representava uma compressão de toda a sequência de entrada, o que se tornava problemático para sequências longas, uma vez que informações relevantes poderiam ser perdidas durante o processo de compactação. Em contraste, o modelo com atenção fornece ao decodificador acesso a todos os estados ocultos gerados pelo codificador ao longo do processamento da sequência de entrada, estando geralmente associados a palavras ou elementos específicos da sequência, permitindo que o modelo atribua pesos distintos a cada parte da entrada, melhorando a qualidade das previsões e a capacidade de capturar dependências de longo alcance (VASWANI *et al*., 2017).

No decodificador, ocorre um processo adicional antes da geração da saída: em vez de utilizar apenas o vetor de contexto final, o decodificador nos modelos com mecanismo de atenção avalia individualmente cada um dos estados ocultos recebidos do codificador, atribuindo uma pontuação a cada estado com base em sua relevância para a tarefa atual. Essa pontuação é calculada utilizando uma função *softmax*, que atribui pesos

normalizados a cada estado oculto do codificador, indicando quais partes da sequência de entrada são mais importantes naquele momento. Para cada passo *t* do decodificador, é calculada uma pontuação de alinhamento, que mede a similaridade ou relevância entre o estado oculto atual do decodificador e cada estado oculto do codificador. Essa medida é obtida por meio de uma função de alinhamento, cuja forma pode variar dependendo da arquitetura utilizada, como funções baseadas em produto escalar ou redes neurais *feedforward*. Em seguida, cada estado oculto é ponderado por sua respectiva pontuação, amplificando os estados com pontuações mais altas, ou seja, mais relevantes para o modelo naquele instante, e atenuando os com pontuações mais baixas, ou seja, menos relevantes. Esse processo de pontuação ocorre a cada etapa do decodificador, auxiliando-o a focar de maneira dinâmica nas partes mais importantes da sequência de entrada em cada passo, resolvendo o problema de dependências de longo alcance e melhorando a qualidade das saídas, como em traduções automáticas, onde a modelagem precisa das relações entre diferentes partes da sequência de entrada é essencial para a produção de resultados coerentes e contextualmente adequados (BAHDANAU *et al*., 2014; VASWANI *et al*., 2017).

Matematicamente, a atenção é calculada a partir de 3 matrizes distintas, denominadas *Query* (Q), *Key* (K) e *Value* (V). Inicialmente, os *tokens* de entrada são convertidos em vetores de *embeddings*, i.e., representações vetoriais das palavras em uma sentença que codificam o significado de cada palavra de tal maneira que palavras mais próximas no espaço vetorial têm expectativa de terem significados semelhantes. Esses *embeddings*, denotados por *X*, são então projetados em três espaços distintos por meio de multiplicações com matrizes de pesos aprendidas durante o treinamento:

- *Query* (Q): É a matriz de consultas. Obtido pela multiplicação de *X* pela matriz de pesos W^Q (pesos aprendidos que transformam as entradas em *queries*). Cada linha de *Q* corresponde ao vetor de consulta de uma entrada específica (como uma palavra ou *token* em um texto).
- *Key* (K): É a matriz de chaves. Obtido pela multiplicação de *X* pela matriz de pesos W^K (pesos aprendidos para as chaves). Cada linha de *K* corresponde ao vetor de chaves que será comparado com as *queries*.

- *Value* (V): É a matriz de valores. Obtido pela multiplicação de *X* pela matriz de pesos W^V (pesos aprendidos para os valores). Representa a informação que se deseja recuperar, onde cada linha de *V* corresponde ao valor associado a uma chave.

Essas projeções permitem que o modelo avalie a relevância de diferentes partes da entrada em relação a si mesmas ou a outras partes da sequência, apresentando um comportamento semelhante ao observado em sistemas de busca, de forma que a aplicação das matrizes de consulta, chave e valor aplicadas no cálculo da atenção pode ser vista como um processo de recuperação de informações, ou seja, uma maneira de recuperar um conjunto de elementos com base em um vetor de probabilidade (BAHDANAU *et al*., 2014).

Para determinar a importância de cada *token* da entrada em relação aos outros, calcula-se a similaridade entre os vetores de consulta (Q) e os vetores de chave (K). Esse processo gera uma pontuação que indica o quanto cada parte do contexto anterior é relevante para o *token* atual. Uma abordagem possível é o produto escalar seguido de uma normalização pela raiz quadrada do número de dimensões do vetor de chaves, representada pela equação (18):

$$Pontuação(Q, K) = \frac{Q \cdot K^T}{\sqrt{d_k}} \tag{18}$$

onde d_k representa a dimensão do vetor de chave. A divisão por $\sqrt{d_k}$ é realizada para evitar que os valores das pontuações se tornem excessivamente grandes, o que poderia prejudicar a estabilidade numérica do modelo durante o treinamento. O produto $Q \cdot K^T$ calcula a similaridade entre cada consulta e chave. Ele resulta em uma matriz de similaridade de atenção, representando quão relevante uma chave é para uma consulta.

As pontuações resultantes são então normalizadas utilizando a função *softmax*, convertendo-as em uma distribuição de probabilidades, conforme pode ser observado na equação (19):

$$Pesos\ de\ Atenção(Q, K) = softmax\left(\frac{Q \cdot K^T}{\sqrt{d_k}}\right) = softmax[Pontuação(Q, K)] \tag{19}$$

Essa etapa assegura que os pesos de atenção atribuídos a cada *token* somem 1, facilitando a interpretação das pontuações como pesos de atenção sobre os valores.

Os pesos de atenção são utilizados para calcular uma média ponderada dos vetores de valor (V), resultando na saída da camada de atenção, conforme pode ser visualizado na equação (20):

$$Atenção(Q, K, V) = [Pesos\ de\ Atenção(Q, K)] \cdot V \quad (20)$$

Essa operação combina as informações dos *tokens* de entrada, ponderadas de acordo com sua relevância, permitindo que o modelo foque nas partes mais importantes da sequência. Em termos práticos, no contexto de Processamento de Linguagem Natural, essa operação permite que uma palavra atenda a todas as outras palavras e combine as informações relevantes para formar sua representação contextualizada (BAHDANAU *et al*., 2014).

Para capturar diferentes aspectos das relações entre *tokens*, os modelos Transformer utilizam múltiplas "cabeças" de atenção, conhecidas como *Multi-Head Attention*. Cada cabeça realiza o processo descrito para o cálculo da atenção de forma independente, permitindo que o modelo aprenda diferentes padrões de atenção. As saídas de todas as cabeças são então concatenadas e projetadas novamente para produzir a saída final da camada de atenção. Esse mecanismo é uma extensão do mecanismo de atenção tradicional, projetado para capturar diferentes aspectos das relações entre *tokens* em uma sequência de forma que, ao invés de calcular uma única função de atenção, o modelo realiza múltiplas operações de atenção em paralelo, cada uma denominada cabeça (*head*) (VASWANI *et al*., 2017).

Para cada cabeça, as entradas são projetadas em diferentes subespaços de representação por meio das matrizes de pesos específicas, W^Q, W^K e W^V, gerando as matrizes Q, K e V, respectivamente. Essas projeções permitem que cada cabeça aprenda diferentes padrões de atenção, focando em diversos aspectos das relações entre *tokens*.

Após isso, cada cabeça realiza o cálculo da atenção de forma independente, seguindo os cálculos descritos anteriormente para a atenção, ou seja: cálculo das pontuações de atenção (calcula-se a similaridade entre *Q* e *K* para obter os pesos de

atenção); aplicação da função *softmax* (as pontuações são transformadas em probabilidades); a atenção é ponderada pela matriz de valores (as probabilidades são usadas para ponderar os valores em *V*, resultando na saída da cabeça de atenção).

Com a aplicação da atenção multi-cabeças, a diferença agora se dá pela concatenação das saídas das cabeças, onde as saídas de todas as cabeças são concatenadas ao longo da dimensão das características. Se houver *h* cabeças e cada uma produz uma saída de dimensão d_k, a concatenação resultará em uma dimensão total de $h \times d_k$. A concatenação das saídas é então passada por uma projeção linear adicional, utilizando uma matriz de pesos W^O, para combinar as informações das diferentes cabeças e produzir a saída final da camada de atenção de acordo com as equações (21) e (22):

$$MultiHead(Q, K, V) = Concat(head_1, \dots, head_h)W^O \quad (21)$$

$$head_i = Atenção\left(QW_i^Q, KW_i^K, VW_i^V\right) \quad (22)$$

Essa arquitetura permite que o modelo capture múltiplas perspectivas das relações entre *tokens*, melhorando a capacidade de representar dependências complexas em sequências. A concatenação das saídas das cabeças seguida de uma projeção linear assegura que as informações combinadas sejam integradas de forma coesa na representação final (VASWANI *et al*., 2017).

2.5.4.3 CACHE KV E O USO DE MEMÓRIA NAS REDES TRANSFORMERS

O modelo Transformer trouxe uma revolução na forma como dados sequenciais são processados, ao introduzir o mecanismo de atenção como um componente central para lidar com entradas e saídas de maneira eficiente. Uma das características mais relevantes desse modelo é a utilização dos conceitos de chaves e valores no mecanismo de atenção, que desempenham um papel crucial ao permitir que o modelo se concentre seletivamente em diferentes partes da sequência de entrada durante a geração de cada elemento da saída. As chaves representam os elementos de contexto da entrada que determinam a relevância de diferentes partes da sequência, enquanto os valores contêm as informações associadas que serão utilizadas no cálculo da atenção (VASWANI *et al*., 2017). O mecanismo de atenção avalia a similaridade entre as consultas, que correspondem ao estado atual do processamento, e as chaves, atribuindo um peso (ou

pontuação de atenção) a cada elemento da sequência de entrada. Esses pesos são então aplicados aos valores, permitindo que o modelo agregue informações relevantes de diferentes partes da sequência e utilize esses dados para gerar a saída correspondente.

Esse processo de aprendizado das redes Transformer geralmente envolve cálculos matriciais intensivos que podem se tornar computacionalmente custosos e exigir grandes quantidades de memória, especialmente quando aplicados a sequências longas e quando envolve um grande universo de *tokens*. Esse custo está diretamente relacionado à forma como as operações de atenção são realizadas, principalmente no cálculo dos pesos de atenção e na aplicação desses pesos aos valores da sequência (BRANDON *et al.*, 2024).

Para melhorar a eficiência computacional e reduzir o custo de memória, especialmente em sequências longas, a técnica de cache KV (*key-value cache*) tem se mostrado uma ferramenta fundamental pois, ao invés do modelo recalcular as chaves e valores a cada passo, as informações previamente calculadas são armazenadas em cache e reutilizadas, permitindo que o modelo se concentre apenas nas novas entradas enquanto utiliza o contexto acumulado de entradas anteriores. Isso não só economiza memória, mas também acelera o tempo de inferência, tornando o modelo mais escalável (BRANDON *et al.*, 2024).

O cache KV é uma técnica usada para acelerar o processo de decodificação nas redes Transformers. De maneira geral, ele armazena as matrizes de chave (K) e valor (V) geradas nos passos anteriores do decodificador. Ao invés de recalcular essas matrizes a cada novo *token*, o modelo reutiliza as informações armazenadas no *cache*. Isso economiza tempo e processamento, permitindo que o modelo foque apenas em calcular a consulta para o novo *token* e fazer as operações de atenção sobre as chaves e valores já calculados (BRANDON *et al.*, 2024; LIU *et al.*, 2024b).

No contexto de modelos autorregressivos, como o GPT, a geração de texto ocorre *token* por *token*, onde cada novo *token* é condicionado aos *tokens* gerados anteriormente. Durante a geração de texto na camada de decodificação, a cada novo *token* gerado, o modelo tradicionalmente processa toda a sequência de entrada novamente, recalculando as representações internas (chaves e valores) para todos os *tokens* anteriores. Esse processo é ineficiente, pois envolve cálculos redundantes de multiplicações entre matriz

e vetor. O cache KV aborda esse problema armazenando as representações de chaves e valores dos *tokens* já processados. Assim, ao gerar um novo *token*, o modelo pode reutilizar essas representações armazenadas, evitando recalcular informações já computadas. Isso reduz significativamente a carga computacional, pois não é necessário reprocessar todo o contexto anterior a cada novo *token* (LIU *et al.*, 2024b).

Depois de armazenar esses vetores em *cache*, é realizado o processo de cálculos matriciais de maneira habitual: é realizado o produto escalar entre o vetor de consulta e a matriz de chave para determinar a relevância das palavras anteriores. Em seguida, é aplicada a função *softmax* para normalizar as pontuações, e por fim as probabilidades são multiplicadas pela matriz de valores, realizando a ponderação dos *tokens* de entrada para obter a atenção da sequência analisada (LIU *et al.*, 2024).

Esse mecanismo de *cache* é fundamental para otimizar a decodificação em modelos de Transformers, pois elimina a necessidade de recalcular valores desnecessários e permite que o modelo foque em gerar novas palavras com base em um contexto já processado, acelerando assim todo o processo de geração de texto. Ao implementar essa técnica, é possível melhorar a eficiência e o desempenho do modelo, tornando-o mais ágil e capaz de lidar com sequências mais longas de forma mais otimizada (LIU *et al.*, 2024b).

Na arquitetura Transformer, ao invés de passar uma sequência inteira de *embeddings* para a camada de auto-atenção, é fornecido apenas o *cache* de chave e valor anterior, junto com o *embedding* do *token* atual. A camada de auto-atenção, então, utiliza esses dados para calcular os novos vetores de chave e valor para o *token* atual, adicionando-os ao cache KV. Isso significa que o modelo é capaz de acessar rapidamente as informações de *tokens* anteriores sem ter que recalcular tudo a cada passo (BRANDON *et al.*, 2024; LIU *et al.*, 2024b).

Além disso, é necessário armazenar essas matrizes de chave e valor em alguma parte da memória da GPU, permitindo que sejam recuperadas quando o próximo *token* estiver sendo processado. Um ponto crucial a se notar é que a única interação entre o *token* atual e os *tokens* anteriores ocorre na camada de auto-atenção. Em todas as outras partes do modelo, como o *embedding* posicional, a normalização de camada (*layer norm*) e a rede neural *feedforward*, não existe interação direta entre o *token* atual e o contexto

anterior. Essa estrutura permite que, ao se utilizar o cache KV, o trabalho necessário para processar cada novo *token* seja constante, independentemente do comprimento da sequência. Isso representa uma melhoria significativa em termos de eficiência computacional, pois o trabalho não aumenta à medida que a sequência se torna mais longa, permitindo que o modelo se concentre em gerar novas saídas de forma mais rápida e eficiente (BRANDON *et al.*, 2024; LIU *et al.*, 2024b).

Embora o cache KV reduza a carga computacional e acelere a inferência, o uso intensivo de memória nas redes Transformer, especialmente com modelos que lidam com sequências muito longas, ainda é um desafio significativo, pois as representações de chaves e valores de todos os *tokens* anteriores precisam ser armazenadas durante o cálculo da atenção. Esse aumento é particularmente significativo em modelos de grande porte ou quando se trabalha com sequências longas e tamanhos de lote elevados. Por exemplo, em modelos como o OPT-30B (ZHANG *et al.*, 2022), o tamanho do cache KV pode atingir centenas de *gigabytes*, dependendo dos parâmetros do modelo e do contexto.

A utilização de técnicas como o cache KV ajuda a mitigar esse problema de uso excessivo de memória ao armazenar partes da memória que são necessárias para os cálculos subsequentes, mas algumas limitações de memória podem surgir ao lidar com sequências extremamente longas (LIU *et al.*, 2024b). Para equilibrar a eficiência computacional e o uso de memória, diversas técnicas foram desenvolvidas e discutidas na literatura, como por exemplo:

- *Multi-Query Attention* (MQA) (SHAZEER, 2019): É uma modificação do mecanismo tradicional de atenção onde, ao invés de usar uma chave e valor únicos para cada consulta, essa técnica permite que múltiplas consultas compartilhem uma única chave e valor. Isso reduz o custo computacional e melhora a eficiência, já que, em vez de calcular atenção para cada consulta individualmente, ela pode reutilizar a mesma chave e valor para várias consultas, sendo particularmente útil em configurações onde há um número grande de consultas, mas poucas chaves e valores únicos.
- *Grouped-Query Attention* (GQA) (AINSLIE *et al.*, 2023): Uma generalização do MQA, onde as consultas são agrupadas em subconjuntos, compartilhando

chaves e valores dentro de cada grupo onde, em vez de calcular a atenção para cada consulta separadamente, o modelo calcula a atenção de maneira agrupada para todos os membros de um grupo. Essa abordagem diminui o número de cálculos, pois reduz o número de operações necessárias para cada grupo de consultas.

- *Multi-Layer Key-Value* (MLKV) (ZUHRI *et al.*, 2024): Estende o compartilhamento de chaves e valores entre diferentes camadas do Transformer, reduzindo ainda mais o tamanho do KV onde, em vez de armazenar e reutilizar apenas as chaves e valores de uma única camada, o modelo pode armazenar chaves e valores de múltiplas camadas, o que permite que cada camada tenha acesso ao contexto adicional de camadas anteriores.

Essas abordagens buscam otimizar o uso de memória durante a inferência, mantendo a eficiência computacional e a qualidade das previsões dos modelos Transformer. Em aplicações práticas, como a identificação de *Trip* em usinas nucleares ou a previsão de eventos, o uso de cache KV e demais técnicas de otimização do uso de memória e de cálculos computacionais podem ser cruciais. Ao analisar sequências de eventos ou dados temporais em tempo real, o modelo pode reutilizar rapidamente o contexto passado armazenado no *cache* para prever falhas iminentes com maior eficiência, o que é essencial em sistemas críticos de monitoramento.

2.5.4.4 CODIFICAÇÃO POSICIONAL

A posição e a ordem das palavras constituem elementos fundamentais de qualquer linguagem, pois definem a gramática e, consequentemente, a semântica de uma sentença. As RNNs consideram intrinsecamente a ordem das palavras, processando-as sequencialmente, palavra por palavra, sendo este o mecanismo que integra a ordem das palavras à estrutura sequencial das RNNs. Por outro lado, a arquitetura Transformer descartou o mecanismo de recorrência em favor do mecanismo de auto-atenção multi-cabeças. Essa mudança elimina a necessidade do método recorrente de *feedback* das RNNs, resultando em um aumento substancial na velocidade de treinamento e na capacidade de retenção de informação à longo prazo dentro de uma sequência de texto. No entanto, como cada palavra em uma sentença é processada simultaneamente pelo

conjunto de blocos de codificadores e decodificadores do Transformer, o modelo, por si só, não possui uma noção inerente de posição ou ordem das palavras, tornando-se necessário um método para incorporar a ordem das palavras ao modelo. Uma solução viável para conferir ao modelo algum senso de ordem é adicionar a cada palavra uma informação que represente sua posição na sentença, sendo esse elemento denominado codificação posicional (*positional encoding*) (VASWANI *et al*., 2017; KAZEMNEJAD, 2019).

Idealmente, os seguintes critérios devem ser atendidos para o desenvolvimento de um modelo de codificação posicional (KAZEMNEJAD, 2019):

1. Cada posição em uma sentença deve ser representada por uma codificação única.
2. A distância relativa entre duas posições deve permanecer consistente, independentemente do tamanho da sentença.
3. O modelo deve ser capaz de generalizar para sentenças mais longas sem a necessidade de ajustes adicionais ou redefinições.
4. As codificações devem estar limitadas a um intervalo predefinido.
5. O método de codificação deve ser determinístico, garantindo resultados consistentes para entradas idênticas.

Dessa forma, a codificação proposta por (VASWANI *et al*., 2017) para a arquitetura Transformer atende a todos os critérios estabelecidos. Primeiramente, ela não é representada por um único número, mas sim por um vetor de dimensionalidade igual ao vetor de *embeddings*, que contém informações relativas a uma posição específica em uma sentença. Além disso, essa codificação não é incorporada diretamente ao modelo onde, em vez disso, o vetor é utilizado para enriquecer cada palavra com informações sobre sua posição na sentença. Em outras palavras, aprimora-se o dado de entrada do modelo para incorporar a ordem das palavras no processamento.

Existem diferentes abordagens para a codificação posicional, mas uma das mais utilizadas em Transformers é baseada em funções seno e cosseno. Neste método, um vetor de codificação posicional é criado para cada palavra usando uma série de funções seno e cosseno com frequências distintas, a depender da posição do *token* e da dimensionalidade

do vetor de *embedding*. As frequências dessas funções diminuem ao longo das dimensões do vetor, o que permite que o modelo capture informações posicionais em diferentes níveis de granularidade. A escolha de funções seno e cosseno se justifica por suas propriedades matemáticas que se adequam à tarefa de representar a posição das palavras (VASWANI *et al*., 2017; KAZEMNEJAD, 2019), de forma que:

- A natureza periódica das funções seno e cosseno permite que o modelo generalize para frases mais longas do que as vistas durante o treinamento.
- A codificação posicional baseada em seno e cosseno permite que a diferença relativa entre as posições de duas palavras seja representada como uma função linear. Isso facilita o aprendizado de relações de distância entre as palavras pelo modelo.

Matematicamente, o vetor correspondente à codificação posicional para cada *token* pode ser calculado da seguinte maneira: seja t a posição do *token* em uma sentença de entrada, $\overrightarrow{p_t}$ a sua codificação posicional correspondente, d a dimensão da codificação e i o índice da dimensão do vetor, então $f(t)^{(i)}$ é a função que produz o vetor de saída $\overrightarrow{p_t}^{(i)}$, sendo definida pelas equações (23) e (24):

$$\overrightarrow{p_t}^{(i)} = f(t)^{(i)} := \begin{cases} sen(w_k t), & se\ i = 2k \\ cos(w_k t), & se\ i = 2k+1 \end{cases} \quad (23)$$

$$w_k = \frac{1}{10000^{\frac{2k}{d}}} \quad (24)$$

Conforme pode ser observado a partir da definição da função na equação (23), as frequências diminuem ao longo das dimensões do vetor. Como o comprimento de onda é inversamente proporcional à frequência, ao longo das dimensões do vetor os comprimentos de onda aumentam de forma exponencial, formando uma progressão geométrica que varia de 2π (nas primeiras dimensões) até $10000 \cdot 2\pi$ (nas dimensões mais altas), o que permite capturar padrões de diferentes escalas, desde relações locais até dependências globais em uma sentença. A razão $\frac{t}{10000^{\frac{2k}{d}}}$ implica que, à medida que k aumenta, o denominador $10000^{\frac{2k}{d}}$ cresce exponencialmente, reduzindo a frequência associada a dimensões maiores do vetor. Como consequência, nas primeiras dimensões

(*k* pequeno), as frequências associadas são maiores e, em dimensões mais altas (*k* grande), as frequências tornam-se progressivamente menores (VASWANI *et al.*, 2017; KAZEMNEJAD, 2019).

A lógica por trás da codificação posicional com seno e cosseno pode ser compreendida através de uma analogia com números binários (KAZEMNEJAD, 2019). A partir da representação binária de números inteiros, é observado que a taxa de mudança dos *bits* diminui à medida que se move para os *bits* mais significativos (da direita para a esquerda), conforme pode ser observado a seguir:

0:	**0**	**0**	**0**	**0**	**8:**	**1**	**0**	**0**	**0**
1:	**0**	**0**	**0**	**1**	**9:**	**1**	**0**	**0**	**1**
2:	**0**	**0**	**1**	**0**	**10:**	**1**	**0**	**1**	**0**
3:	**0**	**0**	**1**	**1**	**11:**	**1**	**0**	**1**	**1**
4:	**0**	**1**	**0**	**0**	**12:**	**1**	**1**	**0**	**0**
5:	**0**	**1**	**0**	**1**	**13:**	**1**	**1**	**0**	**1**
6:	**0**	**1**	**1**	**0**	**14:**	**1**	**1**	**1**	**0**
7:	**0**	**1**	**1**	**1**	**15:**	**1**	**1**	**1**	**1**

Pode-se observar dessa representação a taxa de variação entre diferentes *bits*: o *bit* menos significativo (LSB) alterna a cada número, o segundo *bit* menos significativo muda a cada dois números, o terceiro a cada quatro números, e assim sucessivamente. No entanto, o uso de valores binários seria ineficiente em termos de espaço no contexto de valores de ponto flutuante (VASWANI *et al.*, 2017). Em vez disso, pode-se utilizar suas contrapartes contínuas em ponto flutuante, ou seja, as funções seno e cosseno, sendo a utilização dessas funções equivalente ao observado nos *bits* alternantes. Além disso, ao reduzir suas frequências, é possível transitar de representações mais rápidas (análogas a *bits* de menor significância) para representações mais lentas (análogas a *bits* de maior significância). Dessa forma, as funções seno e cosseno com frequências decrescentes capturam diferentes níveis de informação posicional (KAZEMNEJAD, 2019).

Os *embeddings* de palavras, que representam o significado das palavras dentro do espaço vetorial de *embeddings*, são combinados com as codificações posicionais para fornecer ao modelo uma representação completa de cada palavra em seu contexto. A

forma descrita em (VASWANI *et al*., 2017) para combinar esses dois vetores é através da soma vetorial, porém a concatenação também é uma possibilidade, sendo a escolha entre somar ou concatenar os vetores um fator dependente da arquitetura do modelo e da tarefa específica. A soma é frequentemente preferida por não aumentar a dimensionalidade dos vetores de entrada onde, para cada palavra w_t em uma sentença $[w_1, \cdots, w_t]$, o cálculo do *embedding* correspondente que será fornecido ao modelo é dado pela equação (25) (VASWANI *et al*., 2017):

$$\psi'(w_t) = \psi(w_t) + \overrightarrow{p_t} \tag{25}$$

Para que essa soma seja possível, a dimensão das codificações posicionais é mantida igual à dimensão das *embeddings* das palavras, ou seja, $d_{embedding\ da\ palavra} = d_{codificação\ posicional}$, gerando o vetor de entrada para o bloco de codificação ou decodificação do Transformer.

2.5.4.5 A ARQUITETURA TRANSFORMER

A principal inovação da arquitetura Transformer está no uso do mecanismo de auto-atenção, que permite à rede identificar quais partes da sequência são mais relevantes para o processamento de uma palavra ou *token* específico. Esse mecanismo facilita o aprendizado de dependências entre palavras distantes dentro da sequência, algo que era particularmente difícil de ser alcançado utilizando arquiteturas anteriores, como as redes *feedforward* e recorrentes (VASWANI *et al*., 2017). Além disso, por ser altamente paralelizável, o Transformer permite um treinamento mais rápido em comparação com as arquiteturas de RNNs, que processam sequências de forma sequencial e mais serializada. Dentre os componentes-chave do Transformer, os principais são:

1. **Auto-atenção (*self-attention*)**: Para cada *token* da entrada, o mecanismo de auto-atenção calcula a relevância de todos os outros *tokens* em relação a ele, capturando dependências de longo alcance sem a necessidade de percorrer a sequência de maneira sequencial, termo a termo.

2. **Camadas de Codificador (*Encoder*) e Decodificador (*Decoder*)**: O Transformer possui uma arquitetura em duas partes: o codificador, responsável por processar a sequência de entrada e gerar uma representação intermediária composta por seus

estados ocultos, e o decodificador, que utiliza essa representação para gerar uma resposta de saída.

3. **Codificação Posicional**: Como o Transformer não possui uma estrutura sequencial intrínseca (ao contrário das RNNs), ele usa codificações posicionais (*positional encodings*) para manter informações sobre a ordem e posição dos *tokens* na entrada.

A introdução de mecanismos de atenção em redes neurais, particularmente para modelos baseados em *Seq2Seq*, representou um avanço significativo em diversas tarefas de NLP, como geração de texto, tradução automática, geração de legendas para imagens, e classificação de documentos (BAHDANAU *et al.*, 2014). A principal inovação proporcionada pelo mecanismo de atenção reside na sua capacidade de permitir que os modelos foquem nas partes mais relevantes de uma sequência de entrada durante cada etapa de processamento, o que resolve limitações presentes em outras arquiteturas de Aprendizado Profundo, como o problema do desaparecimento do gradiente e as dificuldades inerentes ao processamento de sequências longas e dependências de longo prazo. Ao oferecer maior flexibilidade no tratamento de dependências temporais ou semânticas em dados sequenciais, o mecanismo de atenção aumenta a precisão e a eficiência dos modelos, consolidando-se como uma abordagem fundamental para resolver problemas complexos em aplicações modernas de Aprendizado Profundo, principalmente os relacionados a tarefas de Processamento de Linguagem Natural (VASWANI *et al*., 2017; TURNER, 2023).

Conforme já mencionado, nos modelos *Seq2Seq* tradicionais, o codificador comprime toda a sequência de entrada em um único vetor de comprimento fixo (vetor de contexto), que é então passado para o decodificador. Essa arquitetura enfrenta dificuldades com sequências longas, pois o vetor de contexto muitas vezes não captura informações suficientes sobre as partes anteriores da sequência, resultando na perda de informações críticas. O mecanismo de atenção utilizado nos Transformers muda isso da seguinte maneira (VASWANI *et al*., 2017):

1. Ao invés de enviar apenas o último estado oculto do codificador, os modelos com atenção enviam todos os estados ocultos do codificador para o decodificador.

2. Para cada *token* de saída, o decodificador avalia cada estado oculto recebido do codificador. Ele atribui uma pontuação a cada estado oculto, sendo frequentemente baseada em um produto escalar ou medida de similaridade entre o estado atual do decodificador e cada estado oculto do codificador.

3. Essas pontuações passam por uma função *softmax* para gerar uma distribuição de probabilidade sobre os estados ocultos do codificador. O decodificador, então, calcula uma soma ponderada de todos os estados ocultos, enfatizando aqueles com pontuações mais altas e reduzindo a importância dos menos relevantes.

4. A soma ponderada torna-se o vetor de contexto, que codifica as partes mais relevantes da sequência de entrada para a etapa atual.

5. O decodificador concatena esse vetor de contexto com seu estado oculto atual, processa esse vetor combinado por meio de uma rede neural *feedforward* e produz uma palavra ou *token* de saída.

Dessa forma, em um problema de predição do(s) próximo(s) *token*(s) em uma sequência de textos, o mecanismo de atenção permite que o modelo se concentre em palavras ou frases específicas da sentença de entrada que são relevantes para a escolha da próxima palavra. Ao invés de comprimir todas as informações de entrada em um único estado final, o mecanismo de atenção distribui o foco sobre diferentes partes da sentença, com base no contexto.

Por exemplo, em um tradutor composto por uma RNN simples, a sequência (ou frase) é fornecida de maneira contínua, uma palavra de cada vez, para gerar *embeddings* de palavras. Como cada palavra depende da palavra anterior, o estado oculto é atualizado de acordo, o que significa que o modelo deve ser alimentado passo a passo. No entanto, em um Transformer, todas as palavras da frase são passadas de uma vez e os *embeddings* das palavras são determinados simultaneamente (VASWANI *et al*., 2017; TURNER, 2023).

Diferentemente dos seres humanos, os computadores não possuem a capacidade de interpretar palavras diretamente; eles operam com números, vetores, matrizes e tensores. Assim, para possibilitar o Processamento de Linguagem Natural por meio de

sistemas computacionais, torna-se necessário converter palavras em representações numéricas, mais precisamente, em vetores. Nesse contexto encontra-se o conceito de espaço de *embeddings*, que atua como um espaço vetorial multidimensional ou um dicionário onde palavras com significados semelhantes ocupam posições próximas. Por exemplo, termos como “rei” e “rainha”, que compartilham características semânticas, estariam mais próximos nesse espaço em comparação com termos como “rei” e “avião”. Esse espaço de *embeddings* atribui um valor específico a cada palavra de acordo com seu significado, permitindo, dessa forma, a conversão de palavras em um vetor correspondente. No espaço de *embeddings*, palavras que têm significados semelhantes estarão mais próximas entre si, enquanto palavras que têm significados diferentes estarão mais distantes. Esse mapeamento permite que cada palavra seja representada numericamente por um vetor, de forma que os modelos sejam capazes de manipulá-las e extrair padrões, e que os computadores entendam a relação entre palavras com base nas distâncias e direções dos vetores, facilitando a análise e processamento de texto em algoritmos de *Deep Learning* (VASWANI *et al*., 2017; TURNER, 2023; ANKIT, 2024).

Outro problema enfrentado durante a representação vetorial é que, em diferentes frases, cada palavra pode assumir significados diferentes. Para resolver esse problema, são utilizados codificadores posicionais, que correspondem a vetores que fornecem contexto de acordo com a posição da palavra em uma frase. Esse processo funciona da seguinte maneira: Palavra (ou *token*) → Embedding → Embedding Posicional → Vetor Final, que é moldado com o contexto. Com isso, o *embedding* posicional adiciona informações sobre a ordem e a posição da palavra na sentença, ajudando o modelo a entender não só o significado da palavra, mas também como ela se relaciona com outras palavras na sequência, sendo esse aspecto especialmente importante para manter o sentido das palavras em diferentes contextos dentro de uma frase (VASWANI *et al*., 2017).

A auto-atenção avalia a relevância de uma palavra em relação às demais palavras de uma sentença, representando essa relação como um vetor de atenção. Para cada palavra, é gerado um vetor que captura os relacionamentos contextuais com as outras palavras na mesma sentença, permitindo ao modelo compreender as dependências entre os elementos da sequência. Entretanto, um desafio inicial da auto-atenção é o fato de que, para cada palavra, o peso atribuído a ela mesma na sequência tende a ser

significativamente maior em comparação com os pesos atribuídos às outras palavras. Isso pode atuar como uma limitação da capacidade do modelo de capturar interações contextuais completas entre as palavras. Dessa forma, para superar essa limitação, múltiplos vetores de atenção são calculados para cada palavra, permitindo que diferentes aspectos das relações contextuais sejam modelados simultaneamente, sendo esses vetores então combinados por meio de uma média ou soma ponderada, resultando no vetor de atenção final de cada palavra. Esse processo é implementado no bloco de atenção de múltiplas cabeças (*multi-head attention block*), no qual várias cabeças de atenção operam em paralelo, de maneira que cada cabeça foca em diferentes aspectos das interações contextuais, permitindo que o modelo capture de forma mais rica e abrangente as relações semânticas entre as palavras. Essa abordagem melhora significativamente a capacidade do Transformer de entender o contexto geral de uma sentença, tornando-o particularmente eficaz em tarefas de NLP (VASWANI *et al*., 2017; TURNER, 2023).

O segundo estágio no processamento do modelo Transformer envolve a aplicação de uma rede neural *feedforward*, responsável por transformar os vetores de atenção gerados no mecanismo de atenção em representações adequadas para a próxima camada do codificador ou decodificador. A rede neural *feedforward* é composta por uma série de operações aplicadas a cada vetor de atenção de forma independente, processando-os individualmente. Uma característica fundamental dessa etapa é que, diferentemente das RNNs, cada vetor de atenção é tratado de maneira independente, sem depender de informações sequenciais ou temporais. Essa independência permite que o processamento seja altamente paralelizável, o que representa uma grande vantagem computacional. Essa capacidade de paralelização é um dos fatores que tornam o Transformer substancialmente mais rápido e eficiente em comparação com modelos sequenciais, como as RNNs, especialmente em tarefas que envolvem grandes volumes de dados ou sequências extensas (VASWANI *et al*., 2017).

O codificador, cuja função principal é gerar os vetores de contexto a partir da sequência de entrada, é constituído pelas estruturas principais descritas na Tabela 1. Essa tabela apresenta um resumo das funcionalidades de cada camada do codificador, oferecendo uma visão geral de suas contribuições para o processamento dos dados. Posteriormente, cada uma dessas camadas será analisada em maior detalhe ao longo deste

capítulo. As estruturas do codificador estão organizadas de forma sequencial, seguindo a ordem em que o fluxo de informações é processado dentro do bloco, de maneira que essa organização reflete o papel central de cada camada no processamento hierárquico, garantindo que a entrada seja transformada progressivamente em representações contextuais.

Tabela 1. Resumo dos componentes principais do bloco de codificação de uma Rede Transformer

Camada	Descrição
Input Embedding	Os dados de entrada são convertidos em vetores de *embeddings*, que são representações vetoriais densas e contínuas extraídas de um dicionário predefinido pelo modelo. Esses *embeddings* capturam o significado das palavras e suas relações semânticas e sintáticas, posicionando palavras semelhantes próximas no espaço vetorial
Positional Encoding	Como o Transformer não processa dados sequencialmente, utiliza a codificação posicional para incorporar informações de ordem e posição às palavras. Essa codificação é adicionada aos *embeddings* e gerada por funções trigonométricas de diferentes frequências, criando representações específicas para cada posição
Multi-Head Attention	Permite que o modelo capture múltiplas relações contextuais em paralelo, com cada "cabeça de atenção" focando em diferentes aspectos do contexto. As saídas das cabeças são concatenadas e projetadas linearmente, formando um único vetor que consolida as informações
Add & Norm	Após o processamento pela camada de *Multi-Head Attention*, aplica-se o *Add & Norm*, composto por uma conexão residual, que preserva as informações originais ao somá-las ao resultado processado, e uma normalização de camada, que padroniza os valores para estabilizar o treinamento
Rede *Feedforward*	Uma rede neural simples que processa cada vetor de atenção de forma independente, ajustando os sinais recebidos e capturando interações mais complexas, refinando as representações para o bloco seguinte do modelo

Add & Norm	Após o processamento pela rede *feedforward*, aplica-se o *Add & Norm*, com uma conexão residual que preserva as informações originais ao somá-las ao resultado processado, e uma normalização de camada que padroniza os valores para estabilizar o aprendizado e reduzir a sensibilidade do modelo

O bloco do decodificador apresenta uma estrutura semelhante à do codificador, conforme descrito na Tabela 2. No entanto, a principal diferença entre as duas arquiteturas está na inclusão de duas componentes adicionais no decodificador: a camada de atenção multi-cabeças mascarada e a camada de saída com a função de ativação *softmax* para a geração do vetor de probabilidades.

Tabela 2. Resumo dos componentes principais do bloco de decodificação de uma Rede Transformer

Camada	**Descrição**
Output Embedding	Transforma os *tokens* de saída em representações vetoriais densas e contínuas, sendo usadas como entrada para o decodificador, permitindo que o modelo capture relações semânticas e contextuais entre os *tokens* gerados anteriormente e o restante da sequência
Positional Encoding	Adiciona informações sobre a posição dos *tokens* na sequência aos *embeddings* de saída. Isso permite ao modelo considerar a ordem dos *tokens*, essencial para capturar relações dependentes de posição na geração da sequência
Masked Multi-Head Attention	Garante que, durante o treinamento, o modelo não acesse palavras futuras da sequência. Por meio de uma máscara, impede-se que palavras subsequentes influenciem a predição, assegurando que cada *token* seja gerado com base apenas nos *tokens* anteriores
Add & Norm	Após o processamento pela camada de *Masked Multi-Head Attention* no decodificador, aplica-se o *Add & Norm*, que combina a conexão residual, preservando as informações originais ao somá-las ao resultado processado, e a normalização de camada, que padroniza as ativações para estabilizar o treinamento e acelerar a convergência

Multi-Head Attention	Avalia as representações geradas pelo codificador para identificar relações contextuais relevantes entre os *tokens* da sequência de entrada e os *tokens* de saída gerados até o momento. Ele permite que o decodificador capture múltiplas perspectivas dessas interações em paralelo, melhorando a qualidade das predições
Add & Norm	Combina uma conexão residual, que preserva as informações originais somando-as ao resultado processado, e uma normalização de camada, que padroniza as ativações para estabilizar o treinamento e melhorar a eficiência do modelo
Rede *Feedforward*	Aplica uma rede neural simples e totalmente conectada a cada *token* de forma independente. Ele ajusta e refina as representações intermediárias, capturando interações mais complexas e preparando os dados para as camadas subsequentes do modelo
Add & Norm	Combina uma conexão residual, que preserva as informações originais ao somá-las ao resultado processado, e uma normalização de camada, que estabiliza as ativações e facilita o aprendizado eficiente do modelo
Linearização e *Softmax*	Após o processamento pelos blocos do decodificador, o resultado é passado por uma camada linear e uma função *softmax*, que geram probabilidades para prever a próxima palavra ou *token* mais provável na sequência

A Figura 14 apresenta a arquitetura básica de um modelo Transformer, composta por blocos de codificadores e decodificadores que operam em conjunto para transformar os dados de entrada em uma saída contendo as probabilidades de cada *token* conhecido ser escolhido como o próximo da sequência. Essa arquitetura reflete a organização modular do modelo, onde cada componente desempenha um papel específico no processamento e na transformação das informações (VASWANI *et al.*, 2017). A partir dessa representação, torna-se possível realizar uma análise detalhada das funções de cada camada que compõe os blocos do codificador e do decodificador. Essa análise é fundamental para compreender o fluxo de informações no modelo e como cada camada

contribui para a construção de representações contextuais e para a geração de saídas e contextualizadas.

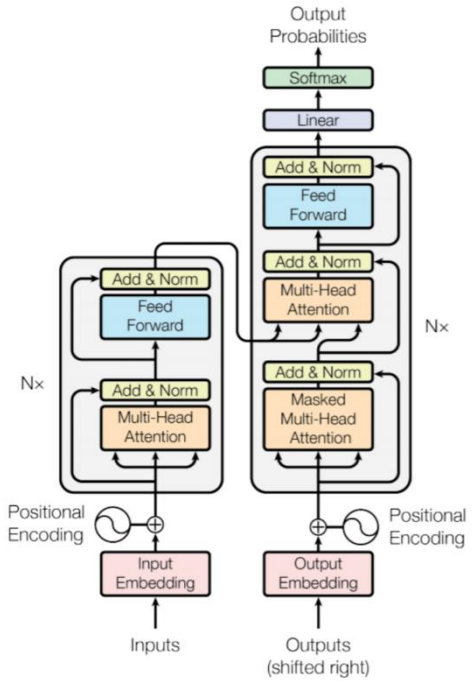

Figura 14. Arquitetura base de um modelo Transformer (VASWANI *et al.*, 2017)

A partir da representação da arquitetura de uma rede Transformer, conforme ilustrado na Figura 14, é possível analisar o fluxo de informações e compreender as funcionalidades específicas de cada camada presente nos blocos de codificação e decodificação. Dessa forma, a seguir serão descritas com maior destaque cada estrutura que compõe os blocos de uma rede Transformer.

Em um primeiro momento, é importante considerar que o significado de uma palavra pode variar conforme seu contexto em uma frase. Por exemplo, a palavra banco pode referir-se a uma instituição financeira ou a um assento/cadeira, dependendo do uso contextual. Para lidar com essa ambiguidade, os Transformers utilizam codificadores posicionais, destacados em azul na Figura 15. Esses codificadores são vetores que adicionam informações sobre a posição de cada palavra na sentença, permitindo ao modelo capturar o contexto e, consequentemente, os múltiplos significados associados.

O processo completo de entrada no modelo, realizado na entrada do bloco de codificação destacado na Figura 15, pode ser descrito da seguinte forma: inicialmente,

cada palavra é transformada em um vetor de *embedding* que representa seu significado no espaço de *embeddings* multidimensional. Em seguida, soma-se ao vetor de *embedding* o vetor de codificação posicional, que fornece a informação de posição da palavra na sentença. O resultado é um vetor final que combina significado semântico e posição, encapsulando tanto o conteúdo da palavra quanto seu contexto na sequência. Esse vetor composto é, então, alimentado ao bloco de codificação do Transformer, garantindo que o modelo possa processar as informações de maneira contextualizada e compreender tanto os significados individuais das palavras quanto o contexto em que estão inseridas.

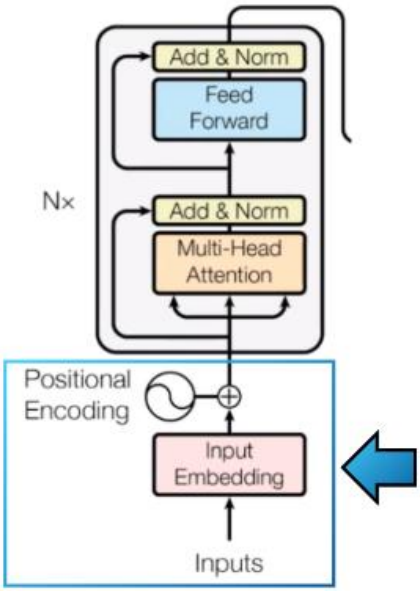

Figura 15. Bloco de Codificação de modelo Transformer – Ênfase na camada de entrada do bloco (VASWANI *et al.*, 2017)

A essência do modelo Transformer está fundamentada no conceito de auto-atenção, projetado para identificar a relevância de cada palavra em relação às demais dentro de uma frase ou sequência de entrada. Esse mecanismo permite que o modelo capture de forma eficiente as dependências contextuais entre palavras, independentemente de sua posição na sequência. No contexto da auto-atenção, cada palavra é representada por um vetor de atenção, que encapsula as relações contextuais entre a palavra em questão e todas as outras palavras da sentença. Essa abordagem possibilita que o modelo compreenda não apenas os significados individuais das palavras, mas também suas interações no contexto da sequência (VASWANI *et al.*, 2017).

Ao processar uma sequência textual, o modelo busca compreender como cada palavra se relaciona com as demais. Inicialmente, porém, é comum que uma palavra atribua a si mesma um peso consideravelmente maior, como se priorizasse sua própria relevância no cálculo da atenção. Contudo, para que o modelo capture as interações contextuais de maneira eficiente, é necessário que ele seja capaz de identificar como cada palavra contribui para o significado da sentença como um todo, e não apenas sua relação isolada (VASWANI *et al.*, 2017; TURNER, 2023).

Para alcançar esse objetivo, são utilizados múltiplos vetores de atenção para cada palavra, sendo que cada vetor é projetado para focar em diferentes aspectos do contexto da sequência de entrada. Posteriormente, esses vetores são concatenados e, em seguida, transformados por meio de uma multiplicação matricial com uma matriz de projeção. Esse processo gera um vetor de atenção final, que sintetiza de forma equilibrada as relações contextuais da palavra com as demais palavras da sentença (VASWANI *et al.*, 2017).

O uso de múltiplos vetores de atenção para cada palavra é implementado por meio de um mecanismo denominado bloco de atenção multi-cabeças (do inglês, *multi-head attention block*), indicado na cor azul na Figura 16. Esse mecanismo permite que o Transformer aprenda simultaneamente diversas relações contextuais entre palavras, e de forma paralela, aumentando sua capacidade de compreender interações complexas em uma sequência de texto. A abordagem de múltiplas cabeças de atenção possibilita que o modelo explore diferentes aspectos das relações semânticas e contextuais em simultâneo, além de contribuir significativamente para a eficiência e a precisão do modelo na captura de nuances semânticas e contextuais do texto, tornando-o altamente eficaz no Processamento de Linguagem Natural (VASWANI *et al.*, 2017).

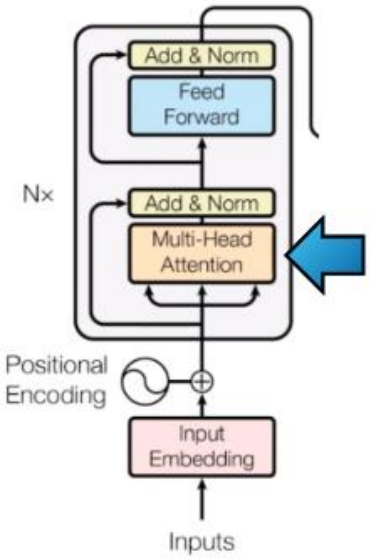

Figura 16. Bloco de Codificação de modelo Transformer – Ênfase na camada de *Multi-Head Attention* (VASWANI *et al.*, 2017)

Após o mecanismo de atenção, o próximo componente na arquitetura Transformer é a rede neural *feedforward*, destacada na Figura 17. Esse tipo de rede, que consiste em uma sequência de camadas densas, é aplicado individualmente e de maneira independente a cada vetor de atenção produzido pela etapa anterior, sendo a principal função da rede *feedforward* transformar os vetores de atenção em uma representação que seja adequada para as camadas subsequentes, seja no bloco de codificação ou de decodificação (VASWANI *et al.*, 2017).

A utilização de uma rede neural *feedforward* nessa etapa desempenha múltiplas funções fundamentais no modelo Transformer. Em primeiro lugar, ela atua como uma transformação não-linear, permitindo ao modelo capturar padrões mais complexos e representar características de alto nível e abstração nos vetores de atenção gerados pelas camadas anteriores. Essa capacidade de abstração aumenta significativamente a eficiência do Transformer em modelar relações semânticas e sintáticas refinadas presentes no texto. Adicionalmente, a rede *feedforward* atua como um mecanismo de ajuste, refinando os vetores de atenção processados para alinhá-los ao espaço representacional exigido pelas camadas subsequentes. Essa etapa é essencial para garantir a coesão e a consistência das representações internas, preparando os dados para o processamento contínuo no fluxo do modelo (VASWANI *et al.*, 2017).

Uma característica distintiva do Transformer em comparação às RNNs é que, durante a aplicação da rede *feedforward*, cada vetor de atenção é processado de forma independente, sem depender dos outros vetores na sequência, sendo essa independência um fator essencial que permite a paralelização do processamento. Em contraste com as RNNs, onde os *tokens* são processados sequencialmente devido à sua arquitetura projetada para modelar dados temporais ou sequenciais, ao mecanismo de *feedback* e ao fato de que possuem conexões internas que permitem que a saída de um passo de tempo seja usada como entrada para o próximo (propagação do estado oculto), no Transformer é possível aplicar a rede *feedforward* simultaneamente a todos os vetores de uma sentença devido à sua arquitetura intrinsecamente paralelizável. Essa capacidade de paralelização contribui significativamente para a eficiência computacional do Transformer, acelerando o processamento de dados, especialmente em grandes conjuntos de texto (VASWANI *et al.*, 2017; TURNER, 2023).

Em termos de funcionamento, a rede neural *feedforward* no Transformer geralmente consiste em um conjunto de camadas densamente conectadas com uma função de ativação não-linear (frequentemente a função ReLU) entre elas. A primeira camada transforma os vetores de atenção em um espaço de maior dimensionalidade, permitindo que o modelo capture relações mais abstratas e interações mais complexas nas representações intermediárias. A segunda camada reduz essa dimensionalidade de volta ao formato original, garantindo compatibilidade com as camadas subsequentes e que as informações estejam alinhadas ao formato esperado pelas próximas etapas do processamento. Esse processo assegura que a transformação preserve a riqueza semântica dos vetores de atenção enquanto ajusta suas representações para as necessidades específicas do restante da arquitetura. Assim, a rede neural *feedforward* desempenha um papel essencial no Transformer, atuando como um módulo de refinamento que potencializa a capacidade do modelo de processar informações complexas de maneira eficiente e paralelizável (VASWANI *et al.*, 2017; TURNER, 2023).

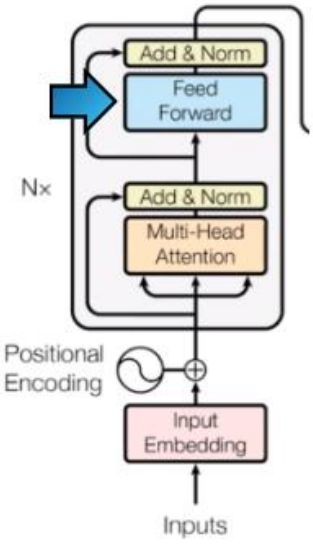

Figura 17. Bloco de Codificação de modelo Transformer – Ênfase na camada de Rede Neural *feedforward* (VASWANI *et al.*, 2017)

No treinamento de um modelo Transformer aplicado a tarefas de geração de textos e previsão do próximo *token* em uma sequência, o processo de aprendizado consiste em fornecer ao modelo sequências de texto nas quais ele é instruído a prever a palavra seguinte com base no contexto fornecido. O objetivo principal é que o modelo aprenda a mapear tanto a estrutura quanto o significado do texto, estabelecendo correspondências linguísticas e semânticas que possibilitem a geração de textos mais precisos e coesos. Durante o treinamento, a sequência de entrada passa por duas etapas fundamentais antes de ser processada pelo bloco de decodificação. Primeiramente, a sequência é convertida em representações vetoriais densas e contínuas na camada de *embedding* dos dados de saída, que encapsula as características semânticas de cada *token* em um espaço multidimensional. Em seguida, são adicionadas as informações de posição através do *Positional Encoding*, garantindo que o modelo seja capaz de considerar a ordem das palavras na sequência. Essas etapas iniciais, destacadas em azul na Figura 18, são fundamentais para que o Transformer processe adequadamente a sequência de texto, mantendo as relações contextuais e estruturais essenciais para a tarefa de geração de texto (VASWANI *et al.*, 2017; ANKIT, 2024).

Na camada de *embedding* do decodificador, cada palavra na sequência é transformada em uma representação vetorial densa que encapsula suas características

semânticas e contextuais, assim como é realizado no codificador. Em vez de lidar diretamente com os *tokens* discretos, o modelo opera utilizando esses formatos de representações numéricas, que facilitam a captura de relações complexas entre as palavras. Esse mapeamento vetorial permite ao modelo identificar semelhanças e padrões linguísticos, ajudando a compreender o contexto em que cada palavra é usada, mesmo em diferentes combinações e estruturas gramaticais. Assim, a camada de *embedding* funciona como um ponto de partida essencial para o processamento subsequente no decodificador, otimizando a interpretação dos dados de saída (VASWANI *et al.*, 2017).

Após a camada de *embedding*, aplica-se a codificação posicional, que incorpora informações sobre a posição de cada palavra na sequência. Essa etapa é fundamental para que o modelo compreenda a estrutura sequencial do texto e consiga interpretar as relações contextuais com base na posição relativa das palavras. A codificação posicional atribui valores únicos a cada posição da sequência, permitindo que o Transformer diferencie *tokens* que possuem significados semelhantes, mas que desempenham papéis distintos devido à sua localização na estrutura da frase. Dessa forma, o modelo é capaz de integrar o significado das palavras com sua organização no texto, garantindo uma análise contextual mais precisa.

No bloco de decodificação, a entrada consiste em duas fontes principais de informação:

1. A sequência parcial fornecida ao modelo até o momento, que representa o contexto utilizado para prever a próxima palavra.
2. As representações internas das palavras já processadas pelo codificador, que são refinadas a cada iteração por meio do mecanismo de atenção.

Essa abordagem de entrada no bloco de decodificação é essencial para o desempenho do Transformer em tarefas de modelagem de linguagem, garantindo que o modelo compreenda não apenas os significados individuais das palavras, mas também o contexto em que elas aparecem.

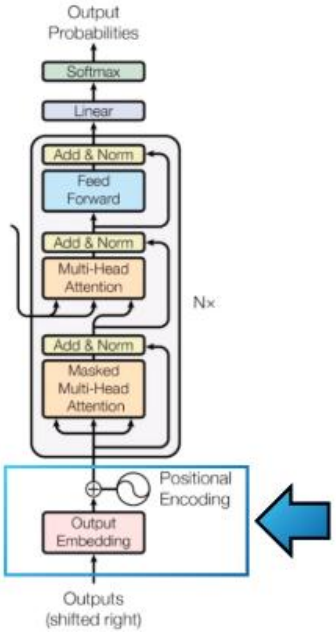

Figura 18. Bloco de Decodificação de modelo Transformer – Ênfase na camada de entrada do bloco (VASWANI *et al.*, 2017)

Após a entrada ser processada no bloco de *embedding* e pela codificação posicional, a sequência parcial passa pelo bloco de auto-atenção, onde são gerados os vetores de atenção para cada palavra. Esses vetores representam o quanto cada palavra na sequência está relacionada às outras já processadas, permitindo ao modelo capturar relações contextuais dentro da mesma sentença, de forma similar ao que ocorre no codificador.

No entanto, no caso da previsão da próxima palavra e outras aplicações dessa arquitetura, como a tradução de textos, o modelo utiliza uma camada de atenção denominada de atenção multi-cabeças mascarada (do inglês, *masked multi-head attention*), conforme destacado em azul na Figura 19. Esse mecanismo é essencial para garantir que o modelo faça previsões baseadas apenas nas palavras já observadas e não em palavras futuras da sequência. Durante o treinamento, o objetivo do modelo é prever a próxima palavra com base no contexto anterior. Para isso, ele calcula a probabilidade da próxima palavra e a compara com a palavra real da sequência. Com base nessa comparação, o modelo ajusta seus parâmetros para melhorar suas previsões. Contudo, é crucial que ele não tenha acesso às palavras que vêm depois da palavra atual, pois isso violaria o objetivo de aprendizado (VASWANI *et al.*, 2017).

O mascaramento é implementado para resolver esse problema. Ele atua ocultando as palavras futuras na sequência de entrada, transformando-as em zeros no cálculo de atenção. Dessa forma, o modelo só pode utilizar informações das palavras anteriores à posição atual. Esse mascaramento garante que as previsões sejam feitas de forma sequencial, simulando o processo natural de geração de texto palavra por palavra (VASWANI *et al.*, 2017). Além disso, como o Transformer utiliza o bloco de atenção multi-cabeças, o modelo é capaz de explorar simultaneamente diferentes aspectos contextuais das palavras já observadas, mesmo com o mascaramento aplicado. Essa capacidade permite que ele capture relações complexas e nuances semânticas na sequência, resultando em previsões mais precisas e contextualizadas.

Portanto, o uso do mascaramento no mecanismo de auto-atenção é fundamental para que o Transformer aprenda a prever a próxima palavra de maneira eficiente e consistente, garantindo que cada etapa do treinamento respeite a ordem sequencial da entrada e simule o processo de geração de texto passo a passo (TURNER, 2023).

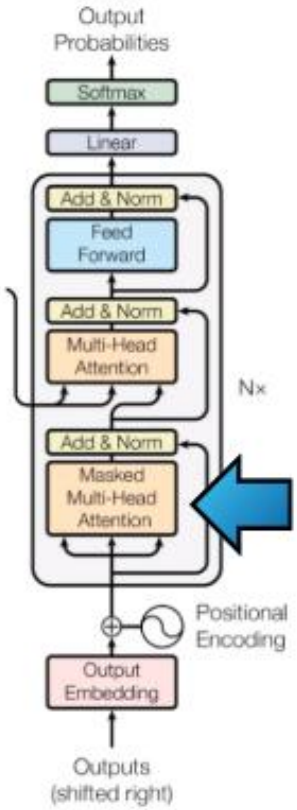

Figura 19. Bloco de Decodificação de modelo Transformer – Ênfase na camada de *Masked Multi-Head Attention* (VASWANI *et al.*, 2017)

Após a geração dos vetores de atenção pela camada de atenção multi-cabeças mascarada, esses vetores são combinados com as representações internas fornecidas pelo bloco de codificação. Esse processo ocorre em um bloco de atenção multi-cabeças *encoder-decoder*, onde as saídas geradas pelo codificador são utilizadas. No diagrama destacado na Figura 20, é possível visualizar como as representações geradas pelo codificador são conectadas ao decodificador, permitindo que este bloco integre informações contextuais extraídas da sequência de entrada original com as informações da sequência parcial gerada até o momento (ANKIT, 2024).

A principal função desse bloco é estabelecer uma conexão entre o contexto global da sequência de entrada e a sequência parcial gerada. Para isso, ele analisa como as palavras na entrada (sequência de contexto fornecida ao codificador) se relacionam com as palavras já previstas na saída (sequência parcial processada pelo decodificador). Esse mecanismo permite que o modelo identifique as dependências contextuais necessárias para prever corretamente a próxima palavra (VASWANI *et al.*, 2017).

No caso da previsão da próxima palavra, o bloco de atenção encoder-decoder realiza um mapeamento detalhado entre as palavras da sequência original e as palavras já processadas. Ele utiliza o mecanismo de atenção para determinar quais partes da entrada são mais relevantes para refinar a representação das palavras na saída. Por exemplo, ao prever a próxima palavra em uma sentença como “O cachorro está ___”, o modelo utiliza o contexto fornecido pela sequência completa do codificador para inferir que uma palavra como “dormindo” ou “latindo” seria uma previsão provável (ANKIT, 2024).

O resultado desse bloco são novos vetores de atenção, que representam não apenas as relações internas entre as palavras da sequência gerada até o momento, mas também suas conexões com as palavras da sequência de entrada. Isso permite que o modelo integre informações contextuais globais da entrada com o contexto local da saída, gerando previsões mais precisas e coerentes (VASWANI *et al.*, 2017; ANKIT, 2024).

Essa etapa é fundamental para o funcionamento do Transformer em tarefas de previsão de palavras, pois garante que o modelo compreenda tanto o contexto completo fornecido pela entrada quanto a relação dinâmica entre as palavras já previstas e as que ainda precisam ser geradas (ANKIT, 2024).

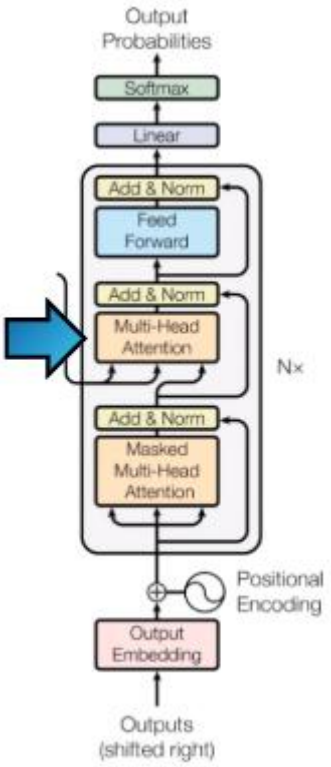

Figura 20. Bloco de Decodificação de modelo Transformer – Ênfase na camada de *Multi-Head Attention* (VASWANI *et al.*, 2017)

Após a geração dos vetores de atenção pelo bloco de atenção multi-cabeças, esses vetores passam por uma rede neural *feedforward*. Essa unidade *feedforward* atua transformando as representações produzidas pela atenção em representações ajustadas e mais refinadas, adequadas para serem utilizadas em etapas subsequentes do modelo, podendo ser tanto por outro bloco de codificação ou decodificação quanto pela camada linear e *softmax* responsável pela geração da resposta de saída do modelo (ANKIT, 2024). De maneira análoga à rede *feedforward* presente no codificador, no decodificador a presença dessa rede possui uma funcionalidade similar, apresentando duas camadas densamente conectadas, intercaladas por uma função de ativação não linear. Na primeira camada, os vetores têm sua dimensionalidade expandida, permitindo que o modelo capture padrões mais ricos e complexos entre os dados. Em seguida, a segunda camada reduz os vetores ao seu tamanho original, ajustando-os para o formato requerido pelas etapas subsequentes. A aplicação de uma rede *feedforward* no final do decodificador permite que o modelo alcance maiores níveis de abstração dos dados após todas as transformações realizadas pelas cabeças de atenção do codificador e decodificador,

proporcionando um aprendizado mais rico e uma visão mais ampla do relacionamento entre cada *token*, suas respectivas posições e contexto geral (VASWANI *et al.*, 2017).

Após o processamento completo pelos blocos do decodificador, o resultado é submetido a uma camada linear seguida por uma função *softmax*, cuja finalidade é transformar as representações geradas em probabilidades de saída. A camada linear, essencialmente, realiza uma projeção das representações do decodificador para o espaço dimensional do vocabulário, de maneira que cada vetor produzido pelo decodificador é transformado em um vetor de dimensões iguais ao número de palavras ou tokens no vocabulário do modelo. Em um problema como o da previsão do próximo *token* na geração de textos ou tradução automática, o vocabulário inclui todas as palavras que o modelo é capaz de prever, independentemente de tradução ou mapeamento entre línguas (VASWANI *et al.*, 2017).

Por fim, a saída da camada linear passa por uma camada *softmax*, responsável por converter os valores numéricos dos vetores em uma distribuição de probabilidade sobre todas as palavras no vocabulário, garantindo que essas probabilidades combinadas somem 1. Esse processo permite que o modelo preveja a palavra ou *token* mais provável como a próxima na sequência, com base no contexto fornecido pelas palavras ou tokens anteriores. Essa operação é repetida de maneira iterativa durante a geração da sequência, garantindo que a predição de cada novo elemento seja condicionada às predições anteriores (VASWANI *et al.*, 2017).

Esse processo, destacado em azul na Figura 21, que envolve a rede *feedforward*, a camada linear e a função *softmax*, é o que permite ao modelo realizar a previsão da próxima palavra com base no contexto aprendido, traduzindo as representações contextuais geradas pelo Transformer em saídas interpretáveis. Essa cadeia de operações é fundamental para o desempenho do Transformer em tarefas de processamento e modelagem de linguagem, pois garante que o modelo seja capaz de transformar representações internas abstratas em uma palavra concreta do vocabulário, com base nas relações contextuais aprendidas ao longo do treinamento (VASWANI *et al.*, 2017).

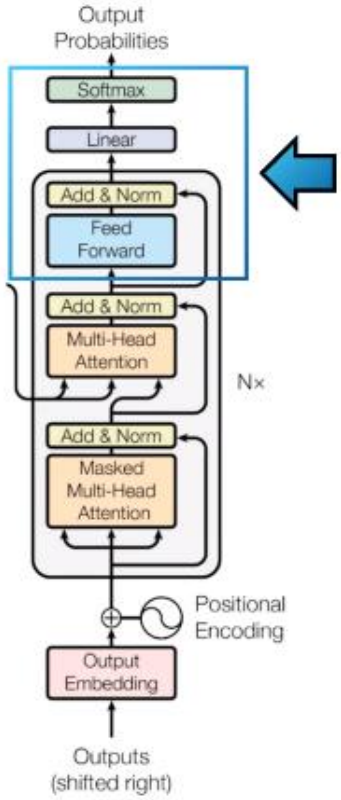

Figura 21. Bloco de Decodificação de modelo Transformer – Ênfase na camada de saída (VASWANI *et al.*, 2017)

Após o processamento de cada camada de atenção e de rede *feedforward* no codificador e decodificador, é aplicada uma operação de adição e normalização (*Add & Norm*), que combina duas operações essenciais: a conexão residual (*skip connection*) e a normalização de camada (*layer normalization*). Esse componente desempenha um papel importante na estabilização do aprendizado e na preservação das informações da entrada original ao longo das camadas do modelo.

A conexão residual adiciona a entrada original da camada que precede a operação de *Add & Norm* ao resultado processado pela atenção ou pela rede *feedforward*. Essa operação garante que as informações iniciais não sejam perdidas durante o processamento e que as camadas subsequentes tenham acesso a uma combinação das representações originais e das transformadas pela atenção. Além disso, essa conexão direta ajuda a facilitar a propagação do gradiente durante o treinamento, mitigando problemas como o desaparecimento ou explosão do gradiente em arquiteturas profundas (VASWANI *et al.*,

2017). A Figura 22 mostra uma representação de aprendizado residual, onde as saídas de uma camada, representadas por $F(x)$, são somadas às próprias conexões de entrada da camada, representadas por x, formando um vetor constituído pela saída da camada anterior após as ponderações realizadas somado com as conexões residuais, ou seja, $F(x) + x$ (HE *et al.*, 2016).

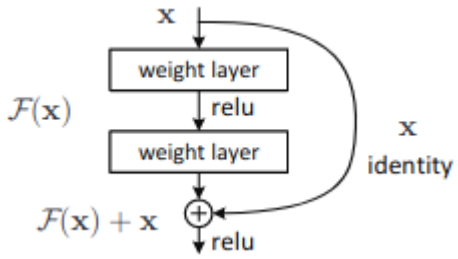

Figura 22. Representação do aprendizado residual (HE *et al.*, 2016)

Em seguida, a normalização de camada (*layer normalization*) ajusta as ativações resultantes, padronizando-as para terem média zero e variância unitária. Diferentemente da normalização em lote (*batch normalization*), a normalização de camada calcula as estatísticas de normalização diretamente a partir das entradas somadas nos neurônios dentro de uma camada oculta. Isso garante que a normalização não introduza dependências adicionais entre os exemplos de treinamento, preservando a independência entre os casos analisados. Essa abordagem tem se mostrado particularmente eficaz em redes neurais recorrentes, contribuindo para a redução do tempo de treinamento e para a melhora no desempenho de generalização de diversos modelos baseados em RNNs (BA *et al.*, 2016). A normalização de camada é realizada calculando-se as estatísticas de normalização sobre todas as unidades ocultas da mesma camada, seguindo a seguinte formulação, representada pelas equações (26) e (27):

$$\sigma^l = \sqrt{\frac{1}{H}\sum_{i=1}^{H}\left(a_i^l - \mu_l\right)^2} \tag{26}$$

$$\mu^l = \frac{1}{H}\sum_{i=1}^{H} a_i^l \tag{27}$$

onde H denota o número de unidades ocultas em uma camada e x_i é o valor da entrada correspondente à i-ésima unidade oculta da camada. Na normalização de camada,

todas as unidades ocultas de uma mesma camada compartilham os mesmos termos de normalização μ (média) e σ (desvio padrão), garantindo consistência na normalização dentro da camada. No entanto, cada caso de treinamento possui termos de normalização distintos, uma vez que esses são calculados individualmente para cada exemplo, permitindo uma normalização personalizada para cada entrada do modelo. Essa normalização reduz a sensibilidade do modelo a variações internas nas ativações, estabilizando o treinamento e acelerando a convergência (BA *et al.*, 2016).

2.5.4.6 BIDIRECTIONAL ENCODER REPRESENTATIONS FROM TRANSFORMERS

Introduzido por (DEVLIN *et al*., 2019), o *Bidirectional Encoder Representations from Transformers* (BERT) é um modelo de aprendizado profundo baseado na arquitetura Transformer, projetado para pré-treinamento de representações linguísticas bidirecionais. Diferente dos modelos anteriores, como OpenAI GPT, que utilizam pré-treinamento unidirecional (da esquerda para a direita), o BERT é totalmente bidirecional, ou seja, ele analisa simultaneamente o contexto à esquerda e à direita de cada palavra em todas as camadas do Transformer, permitindo que o modelo capture relações mais ricas entre palavras, melhorando tarefas como resolução de correferência e compreensão semântica.

O BERT é baseado no *encoder* do Transformer (VASWANI *et al*., 2017), e possui um conjunto de variantes principais, conforme pode ser observado na Tabela 3.

Tabela 3. Principais modelos pré-treinados do BERT

Modelo	Número de Camadas	Unidades Ocultas	Cabeças de Atenção	Quantidade de Parâmetros
BERT-Small	4	512	8	30 milhões
BERT-Medium	8	512	8	42 milhões
DistilBERT-Base	6	768	12	66 milhões
BERT-Base	12	768	12	110 milhões
BERT-Large	24	1024	16	340 milhões

A escolha do modelo a ser utilizado geralmente irá depender dos recursos computacionais disponíveis (restrição de *hardware*) e da capacidade de aprendizado que

se espera do modelo (*trade off* entre velocidade de inferência e precisão), pois modelos com maior quantidade de parâmetros apresentam maior tempo de inferência, porém geralmente geram resultados melhores do que modelos mais simples.

O modelo DistilBERT (SANH *et al.*, 2019) é uma versão reduzida do BERT-Base criada a partir de uma técnica denominada destilação de conhecimento (*knowledge distillation*), que consiste em transferir o conhecimento de um modelo maior (o "professor"), tipicamente com maior capacidade de aprendizado, para um modelo menor (o aluno) (BUCILUǍ *et al*., 2006; HINTON *et al*., 2015). O treinamento do DistilBERT ocorre de forma que ele aprenda não apenas os rótulos corretos das tarefas, mas também as distribuições de probabilidade geradas pelo modelo professor (BERT-Base). Isso é feito através de uma combinação de:

- **Correspondência de *logits***, onde o modelo aluno busca imitar as previsões do professor, porém, em vez de simplesmente aprender os rótulos corretos das amostras de treinamento, o aluno aprende a distribuição de probabilidades das saídas do professor. Isso é realizado através da correspondência dos *logits* entre aluno e professor, ou seja, dos valores numéricos brutos gerados pela última camada do modelo antes da aplicação da função *softmax*. Eles representam a confiança do modelo em cada possível saída, mas ainda não foram convertidos em probabilidades.
- ***Matching* de *embeddings***, que força o DistilBERT a aprender representações similares às do BERT. Nesse caso, tanto o professor quanto o aluno processam a mesma entrada, gerando seus respectivos *embeddings* das palavras. Após isso, um termo de perda (*loss*) é calculado para minimizar a diferença entre os *embeddings* gerados pelo aluno e pelo professor. Através de um processo iterativo de otimização, o termo de perda vai diminuindo, permitindo ao aluno aprender a capturar informações semânticas semelhantes às do professor, mesmo sem possuir a mesma profundidade de rede.
- ***Matching* de atenção**, para que o aluno replique os padrões de auto-atenção do modelo maior. Conforme mencionado, no Transformer, cada cabeça de atenção gera um mapeamento de atenção, que mostra quais palavras influenciam mais (possuem maior relevância) na representação de outras.

> Durante o treinamento, o modelo aluno busca replicar os padrões de atenção do professor, que é realizado através de um termo de perda, sendo este adicionado para minimizar a diferença entre os mapas de atenção do aluno e do professor.

O resultado dessa destilação é um modelo que mantém boa parte do desempenho do modelo original, mas com menor custo computacional e maior velocidade de inferência. O DistilBERT é aproximadamente 60% menor, 2x mais rápido e apresenta um menor custo energético do que o BERT-Base, tornando-se uma escolha mais apropriada para aplicações em que a velocidade de inferência é um fator crítico ou com restrições de *hardware*. O modelo destilado mantém cerca de 97% do desempenho do BERT-Base, mas com quase a metade dos parâmetros (66M contra 110M), tornando-se uma opção intermediária entre eficiência e desempenho (SANH *et al.*, 2019).

O BERT utiliza uma técnica de tokenização conhecida como *WordPiece* (DEVLIN *et al*., 2019), que corresponde a uma técnica de segmentação de palavras usada para lidar com o grande número de variações possíveis no vocabulário de um idioma. Em vez de armazenar todas as palavras completas como *tokens* individuais, o *WordPiece* divide palavras em subpalavras ou caracteres, permitindo que o modelo generalize melhor para palavras desconhecidas ou raras. O vocabulário do BERT contém aproximadamente 30.000 *tokens* (DEVLIN *et al*., 2019), compostos por palavras comuns e partes de palavras menos frequentes. Por exemplo, uma palavra como “transformers” pode ser dividida em dois tokens: “transform” e “##ers”. O prefixo “##” indica que esse *token* é uma continuação de outro *token*, permitindo que o modelo reconheça padrões mesmo em palavras compostas. Se a palavra fosse menos comum, como “tokenização”, ela poderia ser dividida em ainda mais partes, como “token” e “##ização”. Esse método melhora a capacidade do BERT de lidar com palavras novas e reduz significativamente o tamanho do vocabulário necessário para o treinamento.

Além da tokenização, o BERT também utiliza três tipos de *embeddings* para representar cada entrada de forma mais rica e contextualizada (DEVLIN *et al*., 2019). O primeiro tipo são os *Token Embeddings*, que são as representações vetoriais de cada *token* geradas a partir do vocabulário *WordPiece*. Cada palavra ou subpalavra recebe um vetor

único que captura sua semântica e será refinado ao longo das camadas do modelo, através das camadas de atenção e da abstração promovida pela rede *feedforward* do bloco de codificação.

O segundo tipo são os *Segment Embeddings*, que indicam a que frase um determinado *token* pertence. Isso é especialmente importante quando o BERT processa pares de frases, como em tarefas de pergunta e resposta ou inferência textual. Para isso, o modelo adiciona um identificador diferente para *tokens* pertencentes à frase A e frase B, permitindo que o BERT diferencie corretamente as duas partes da entrada e estabeleça relações entre elas.

Por fim, também são utilizados *Position Embeddings*, que fornecem informações sobre a posição dos *tokens* na sequência. Diferente de modelos baseados em recorrência, como LSTMs, a arquitetura Transformer do BERT não possui uma ordem intrínseca para os *tokens*, pois opera apenas utilizando o mecanismo de atenção (*self-attention*). Para que o modelo compreenda a ordem das palavras e capture a estrutura da frase, são adicionados *embeddings* que codificam a posição de cada *token* dentro da sequência. Assim, o BERT pode entender, por exemplo, na frase “O cachorro correu atrás da bola”, que a palavra “cachorro” vem antes da palavra “correu”, mantendo a coerência sintática.

Com isso, esses três tipos de *embeddings* são somados e servem como entrada para o modelo, garantindo que cada *token* carregue informações sobre seu significado, sua posição na frase e a que segmento pertence (DEVLIN *et al*., 2019).

2.5.4.7 GENERATIVE PRE-TRAINED TRANSFORMER

Introduzido por (RADFORD *et al*., 2019), o *Generative Pre-trained Transformer 2* (GPT-2) é um modelo de aprendizado profundo baseado na arquitetura Transformer, projetado para geração de linguagem natural de maneira coerente e contextualizada. Diferente do BERT, que utiliza um pré-treinamento bidirecional para compreensão de linguagem, o GPT-2 é um modelo autorregressivo, o que significa que ele prevê a próxima palavra em uma sequência considerando apenas o contexto à esquerda, processando o texto de forma unidirecional. Essa abordagem permite que o modelo gere sentenças mais naturais e fluídas, tornando-o adequado para tarefas como

complementação de texto, criação de diálogos, *chatbots* e sistemas de pergunta e resposta sem a necessidade de arquiteturas específicas para cada tarefa.

O GPT-2 é baseado no *decoder* do Transformer (VASWANI *et al*., 2017), e possui um conjunto de variantes principais, conforme pode ser observado na Tabela 4.

Tabela 4. Principais modelos pré-treinados do GPT-2

Modelo	Número de Camadas	Unidades Ocultas	Cabeças de Atenção	Quantidade de Parâmetros
DistillGPT2-Base	6	768	12	82 milhões
GPT2-Base	12	768	12	124 milhões
GPT2-Medium	24	1024	16	355 milhões
GPT2-Large	36	1280	20	774 milhões
GPT2-Extra-Large	48	1600	25	1,56 bilhões

A técnica de destilação (BUCILUǍ *et al*., 2006; HINTON *et al*., 2015) de conhecimento utilizada para obter o DistilGPT-2 segue princípios semelhantes aos usados no DistilBERT, mas com algumas adaptações específicas para um modelo de linguagem autorregressivo. O DistilGPT-2 apresenta 6 camadas em vez das 12 do GPT-2 base, mantendo cerca de 95% do desempenho do modelo original, mas sendo 2 vezes mais rápido e consumindo metade da memória (SANH *et al.*, 2019). Para atingir esse resultado, três técnicas principais foram aplicadas durante a destilação:

- **Correspondência de *logits***, onde o modelo aluno aprende não apenas os rótulos corretos das previsões, mas também as distribuições de probabilidade do modelo professor. Como o GPT-2 realiza a geração de texto prevendo o próximo *token*, o DistilGPT-2 busca imitar essa distribuição de saída, capturando a incerteza do modelo maior.
- ***Hidden State Matching***, permitindo que, em cada camada, o modelo aluno busque replicar os estados internos do professor. Como o GPT-2 gera texto passo a passo, garantir que o aluno possua representações intermediárias semelhantes às do professor ajuda a manter a coerência das frases geradas.

- **Redução de Tokenização e *Embeddings***, de forma que, para otimizar o processo de inferência, algumas camadas de *embeddings* são reduzidas ou fundidas, permitindo que o modelo funcione com menos parâmetros sem comprometer a qualidade na representação das palavras.

O GPT-2 utiliza uma técnica denominada *Byte-Pair Encoding* (BPE) para a tokenização do texto (GAGE, 1994; SENNRICH *et al*., 2015). Essa técnica corresponde a um algoritmo de compressão adaptado para o processamento de linguagem natural, dividindo palavras em subpalavras mais comuns com base na frequência estatística dentro de um *corpus* textual. O objetivo é representar palavras frequentes como *tokens* únicos, enquanto palavras raras são divididas em subpalavras menores, provando ser uma abordagem intermediária entre tokenização por palavras inteiras e tokenização caractere a caractere, permitindo um equilíbrio entre generalização e eficiência computacional.

Diferente de algoritmos de tokenização BPE tradicionais que tratam as palavras como sequências de caracteres Unicode, o tokenizador do GPT-2 interpreta as palavras como sequências de *bytes* (RADFORD *et al*., 2019), caracterizando um subtipo da técnica BPE denominada Byte-Level BPE (WANG *et al*., 2020). Isso significa que, em vez de partir de um vocabulário baseado apenas em letras, sílabas ou palavras inteiras, ele usa uma base fixa de 256 *tokens* correspondentes aos 256 (8 bits) valores possíveis de um *byte*. Na prática, essa abordagem possui algumas implicações importantes, como por exemplo:

- **Todos os caracteres são representáveis**: Como os modelos tratam os *tokens* diretamente no nível de *bytes*, qualquer caractere, símbolo, *emoji* ou caractere especial pode ser representado sem precisar de um *token* desconhecido ([UNK]).
- **Criação de um vocabulário base**: Com apenas 256 *tokens* base, qualquer texto pode ser representado, evitando a necessidade de criar um vocabulário excessivamente complexo. Após obter esse vocabulário base, são adicionados novos *tokens* até alcançar o tamanho desejado do vocabulário, aprendendo fusões (*merges*), constituindo regras utilizadas para combinar dois elementos do vocabulário existente em um novo *token*. No início, essas fusões criam

tokens com dois caracteres e, à medida que o treinamento avança, formarão subpalavras mais longas.

- **Redução da fragmentação de *tokens***: Como o vocabulário inclui todos os caracteres, menos fragmentações ocorrem ao tokenizar palavras em diferentes idiomas ou que contenham símbolos especiais.

Essa abordagem melhora a generalização do modelo, pois evita que palavras incomuns sejam convertidas em um token [UNK], i.e. desconhecido, o que poderia dificultar o aprendizado e a inferência. Dessa forma, o modelo GPT-2 foi treinado com um vocabulário de 50.257 *tokens* (RADFORD *et al*., 2019), onde cada *token* pode representar uma palavra inteira (se for comum o suficiente), uma parte de uma palavra (prefixo, sufixo ou raiz) ou caracteres individuais (para palavras muito raras ou símbolos incomuns).

Por fim, um comparativo entre os modelos BERT e GPT-2 abordados neste trabalho pode ser visualizado na Tabela 5, permitindo visualizar os cenários de aplicação de cada modelo, assim como suas particularidades:

Tabela 5. Comparativo entre os modelos BERT e GPT-2

Característica	**GPT-2**	**BERT**
Direcionalidade	Unidirecional (da esquerda para a direita)	Bidirecional (analisa simultaneamente o contexto à esquerda e à direita de cada palavra)
Objetivo de Treinamento	Modelagem de linguagem (previsão do próximo *token*)	Classificação de Sequências, Masked Language Model (MLM) e Next Sentence Prediction (NSP)
Baseado em	Transformer Decoder	Transformer Encoder
Uso principal	Geração de texto	Compreensão e classificação de texto
Tokenização	Byte-Pair Encoding	WordPiece Tokenization
Variantes	GPT-2 Base, Medium, Large, XL	BERT Small, Medium, Base, Large

2.6 EXTRAÇÃO DE CARACTERÍSTICAS A PARTIR DE TEXTOS

Para realizar o aprendizado de máquina em documentos de texto, primeiro é necessário converter o conteúdo textual em vetores numéricos de características. A maneira mais intuitiva de realizar esse processo é utilizando uma representação denominada Saco de Palavras (do inglês, *Bag of Words*), ou *BoW* (HARRIS, 1954), que funciona da seguinte maneira (MANNING *et al.*, 2008):

1. É atribuído um identificador inteiro fixo a cada palavra que aparece em qualquer documento do conjunto de treinamento (por exemplo, construindo um dicionário que mapeia as palavras para índices inteiros).

2. Para cada documento *i*, é contado o número de ocorrências de cada palavra *w* e esse valor é armazenado em $X[i, j]$, onde *j* é o índice da palavra *w* no dicionário, representando a característica número *j*.

A representação *Bag of Words* implica que o número de características geradas é equivalente ao número de palavras distintas presentes no *corpus* textual. Considerando um cenário hipotético em que se dispõe de 100.000 amostras (ou seja, 100.000 documentos) e cada documento possui 100.000 características, o armazenamento da matriz X como um *array* de vetores do tipo *float32* (número de ponto flutuante de 32 *bits*, exigindo 4 *bytes* para armazenar cada número) demandaria aproximadamente 100.000×100.000×4 *bytes*, o que equivale a 40 GB de memória RAM apenas para armazenar os valores no dicionário de características, o que pode ser difícil de gerenciar em projetos de larga escala. Um aspecto relevante dessa abordagem é que, na maioria dos casos, os valores da matriz X serão predominantemente zeros, uma vez que em um documento típico apenas algumas milhares de palavras distintas são efetivamente utilizadas. Por isso, é dito que a técnica *BoW* frequentemente resulta em um conjunto de dados esparsos e de alta dimensionalidade. Para lidar com essa característica e otimizar o uso de memória, uma abordagem eficiente consiste em armazenar apenas os elementos não nulos dos vetores de características, reduzindo significativamente o consumo de memória sem comprometer as informações relevantes para a análise (YATES & NETO, 2013).

A abordagem de *Bag of Words* simplifica a análise de texto, transformando o conteúdo em valores numéricos que os algoritmos de aprendizado de máquina podem interpretar. No entanto, como cada palavra única se torna uma característica, o número de características cresce rapidamente à medida que o vocabulário aumenta. Isso leva à criação de matrizes extremamente grandes e esparsas, ou seja, repletas de valores nulos. Para lidar com a alta dimensionalidade e a esparsidade dos dados, técnicas como matrizes esparsas (onde apenas os valores não nulos e seus respectivos índices são armazenados) são usadas para otimizar o uso de memória e aumentar a eficiência computacional (RAJARAMAN & ULLMAN, 2011).

Além disso, é importante notar que essa representação não leva em consideração a ordem das palavras, o que significa que se perde a estrutura sintática e semântica do texto original. Métodos mais avançados, como *TF-IDF* (*Term Frequency-Inverse Document Frequency*) (MANNING *et al.*, 2008; RAJARAMAN & ULLMAN, 2011; YATES & NETO, 2013), e técnicas baseadas em Redes Neurais Artificiais, como *Word Embeddings* (incorporações de palavras) (JURAFSKY & MARTIN, 2000), podem ser usadas para capturar relações mais complexas entre as palavras e melhorar o desempenho dos modelos de Aprendizado de Máquina.

2.6.1 TF-IDF

As técnicas *TF* (do inglês, *term-frequency)* (frequência do termo) e *TF-IDF*, que corresponde à frequência do termo multiplicada pela frequência inversa de documentos são esquemas comuns de ponderação de termos em recuperação de informações, que também tem sido amplamente utilizado na classificação de documentos. O cálculo da frequência de um determinado termo t dentro de um documento d pode ser visualizado na equação (28), que considera a contagem de ocorrências do termo no documento, normalizada pelo número total de termos no documento:

$$\mathrm{tf}(t,d) = \frac{número\ de\ ocorrências\ de\ \boldsymbol{t}\ no\ documento\ \boldsymbol{d}}{número\ total\ de\ termos\ no\ documento\ \boldsymbol{d}} \qquad (28)$$

Variações da equação (28) incluem, por exemplo, a aplicação de uma escala logarítmica para diminuir o impacto de termos que aparecem muitas vezes no documento,

que acabam distorcendo a relevância de outras palavras. Essa variante pode ser visualizada na equação (29):

$$\text{tf}(t, d) = 1 + \log(\textit{número de ocorrências de } \boldsymbol{t} \textit{ em } \boldsymbol{d}) \tag{29}$$

O objetivo de usar *tf-idf* em vez das frequências brutas de ocorrência de um *token* em um determinado documento é reduzir o impacto dos *tokens* que aparecem com muita frequência em um *corpus* textual. Esses *tokens*, por ocorrerem muitas vezes, tendem a ser empiricamente menos informativos do que aqueles que aparecem em uma pequena fração do *corpus* de treinamento. A equação usada para calcular o *tf-idf* de um termo t dentro de um documento d que faz parte de um conjunto de documentos pode ser visualizada na equação (30):

$$\text{tf-idf}(t, d) = \text{tf}(t, d) \times \text{idf}(t) \tag{30}$$

E a *idf* (não suavizada) é calculada pela equação (31):

$$\text{idf}(t) = \log\left(\frac{n}{df(t)}\right) + 1 \tag{31}$$

Onde n é o número total de documentos no conjunto e $df(t)$ é a frequência do termo t no documento d; a frequência do documento corresponde ao número de documentos no conjunto total de documentos que contém o termo t. O efeito de adicionar 1 à fórmula da *idf* é garantir que termos com uma *idf* igual a zero, ou seja, termos que aparecem em todos os documentos do conjunto de treinamento, não sejam completamente ignorados.

Para o cálculo da *idf* suavizada, a constante 1 é adicionada tanto ao numerador quanto ao denominador da fórmula da *idf*, como se um documento extra tivesse sido observado contendo todos os termos da coleção de documentos exatamente uma vez. Isso evita divisões por zero, conforme pode ser observado na equação (32):

$$\text{idf}(t) = \log\left(\frac{1 + n}{1 + df(t)}\right) + 1 \tag{32}$$

A técnica *tf-idf* é fundamental para melhorar a qualidade de modelos de Aprendizado de Máquina em tarefas de Processamento de Linguagem Natural. Enquanto a frequência de termos (*tf*) representa o número de vezes que uma palavra aparece em um

documento, o *idf* ajusta essa frequência, penalizando termos que aparecem em muitos documentos e, portanto, são considerados menos informativos (MANNING *et al.*, 2008).

Palavras muito comuns, como “o”, “a”, “de”, “em”, tendem a aparecer em quase todos os documentos de um *corpus*, o que dilui sua importância. Por outro lado, palavras que aparecem em poucos documentos podem ser mais representativas do conteúdo específico de cada um. O *tf-idf* permite capturar essa nuance, destacando termos mais relevantes e diferenciando melhor um documento do outro (YATES & NETO, 2013).

Esse conceito é amplamente usado em sistemas de recomendação, motores de busca e classificadores de texto, auxiliando a balancear a representatividade das palavras, preservando a informação útil e reduzindo o ruído causado por termos excessivamente frequentes (MANNING *et al.*, 2008; YATES & NETO, 2013).

2.6.2 WORD EMBEDDINGS

Os *word embeddings* são representações vetoriais densas de palavras que capturam suas semelhanças semânticas em um espaço de alta dimensionalidade. Diferentemente das abordagens tradicionais, como o *BoW* e o *TF-IDF*, que utilizam matrizes esparsas e desconsideram as relações semânticas e contextuais entre as palavras, os *embeddings* representam cada palavra como um vetor de números reais de maneira que, através dessa abordagem, palavras com significados semelhantes ocupam posições próximas no espaço vetorial (JURAFSKY & MARTIN, 2000). Essas representações são aprendidas com base no contexto em que as palavras aparecem nos textos, permitindo que as relações semânticas sejam incorporadas ao modelo. Entre os métodos amplamente utilizados e citados na literatura para gerar *word embeddings*, destacam-se técnicas como:

1. **Word2Vec** (MIKOLOV *et al.*, 2013): É um modelo reconhecido por sua capacidade de aprender representações vetoriais densas de palavras a partir de um grande *corpus* de texto. Esse método utiliza redes neurais para capturar relações semânticas e contextuais entre palavras, gerando *embeddings* que posicionam palavras semanticamente semelhantes em regiões próximas no espaço vetorial. O modelo apresenta duas variantes principais, cada uma com uma abordagem distinta para aprender as relações contextuais entre palavras:

- CBOW (*Continuous Bag of Words*): o modelo utiliza as palavras do contexto (palavras ao redor da palavra alvo) para prever a palavra central. O CBOW funciona treinando a rede neural para minimizar o erro entre a palavra predita e a palavra central real, o que permite capturar os padrões de uso das palavras no *corpus*. Por ser computacionalmente mais eficiente, o CBOW é amplamente utilizado em tarefas que envolvem grandes volumes de dados textuais.

- Skip-Gram: Em contraste com o CBOW, a abordagem Skip-Gram tenta prever as palavras do contexto com base em uma palavra alvo. O objetivo do modelo é encontrar as palavras mais prováveis de aparecer ao redor da palavra central, otimizando as representações para capturar relações semânticas mais complexas, mesmo em *corpus* menores ou mais esparsos.

2. **GloVe (*Global Vectors for Word Representation*)** (PENNINGTON *et al.*, 2014): É um método que combina a abordagem local (palavras vizinhas) e global (frequência de coocorrência em todo o *corpus*) para gerar representações vetoriais densas de palavras. Diferentemente de métodos puramente contextuais, como o Word2Vec, o GloVe utiliza informações sobre a coocorrência global de palavras em todo o *corpus* para capturar relações semânticas de maneira mais abrangente. A principal ideia do GloVe é baseada no fato de que a frequência com que duas palavras coocorrem em um *corpus* reflete informações sobre seu relacionamento semântico. Para isso, o método constrói uma grande matriz de coocorrência, onde cada célula representa o número de vezes que uma palavra aparece próxima a outra dentro de uma janela de contexto definida. Essa matriz captura o contexto global de cada palavra ao longo do *corpus*.

3. **FastText** (JOULIN *et al.*, 2016): É uma extensão do método Word2Vec que introduz uma incorporação da análise de subpalavras na geração de representações vetoriais densas de palavras. Enquanto o Word2Vec considera apenas palavras inteiras como unidades discretas, o FastText vai além,

decompondo cada palavra em seus componentes menores (n-gramas de caracteres, como prefixos, sufixos ou segmentos internos). Essa abordagem permite ao modelo capturar informações mais ricas sobre a estrutura morfológica das palavras, o que é especialmente útil para lidar com palavras raras ou previamente não vistas no treinamento.

A Figura 23 apresenta uma ilustração de como *embeddings* de palavras podem capturar relações semânticas complexas ao demonstrar que, em um espaço de *embeddings*, a posição (distância e direção) pode codificar o relacionamento entre palavras ao compreender o significado de palavras, frases, expressões e sentenças, considerando o contexto em que são utilizadas. Uma característica relevante dos *word embeddings* é a capacidade de capturar relações lineares entre palavras, permitindo operações aritméticas que refletem analogias linguísticas. Por exemplo, no contexto do espaço de *embeddings* representado na Figura 23, a diferença vetorial entre “rei” e “homem” é semelhante à diferença entre “rainha” e “mulher”.

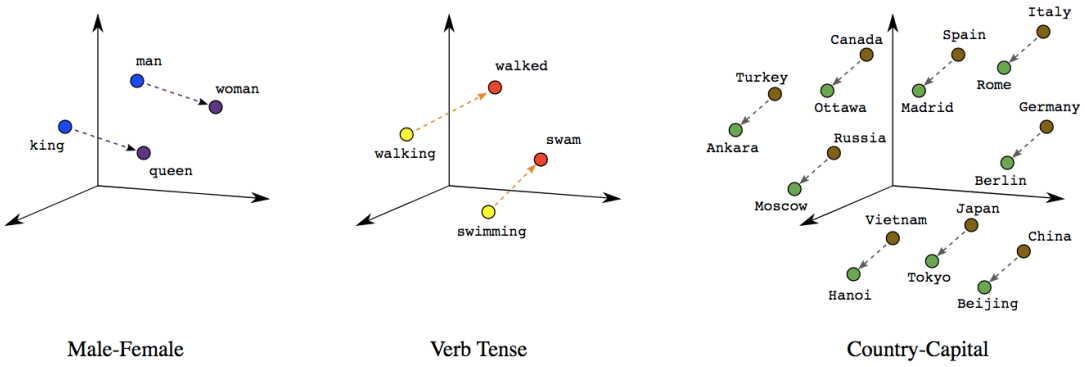

Figura 23. Representação de relações lineares entre palavras em um espaço em *embeddings* (ANALA, 2020)

Matematicamente, um espaço de *embedding* é definido como um espaço latente onde itens semelhantes são posicionados mais próximos uns dos outros em comparação a itens menos semelhantes. Nesse contexto, frases semanticamente semelhantes devem possuir vetores de *embedding* semelhantes, posicionando-se, portanto, mais próximas no espaço, consequentemente indicando uma maior similaridade entre as sentenças. Diversas tarefas úteis podem ser formuladas em termos de similaridade textual, como:

- **Busca:** Quão semelhante é uma consulta (*query*) a um determinado documento em um banco de dados?
- **Filtragem de *spam*:** Quão próxima está uma mensagem de e-mail de exemplos classificados como *spam*?
- **Agente conversacional:** Quais exemplos de intenções/respostas conhecidas estão mais próximos da mensagem do usuário?

Nesses casos, é possível calcular previamente os *embeddings* dos itens de interesse (por exemplo, documentos para busca ou exemplos para classificação) e armazená-los em um banco de dados indexado. Esse método permite capturar o poder de compreensão da linguagem natural oferecido por modelos neurais profundos e representá-lo na forma de *embeddings* textuais. Assim, à medida que novos itens são adicionados ao banco de dados, é possível executar buscas ou classificações de forma eficiente, sem a necessidade de recursos computacionais intensivos, como GPUs.

Essa comparação direta de similaridade textual é apenas uma das aplicações possíveis dos *embeddings* de texto. Com frequência, *embeddings* são utilizados como entrada em algoritmos de Aprendizado de Máquina ou em arquiteturas neurais e Aprendizado Profundo, sobre os quais componentes adicionais específicos para a tarefa são construídos. As principais características dos *embeddings* são:

- **Semelhança semântica**: Palavras com significados semelhantes têm representações vetoriais próximas no espaço de *embeddings*.
- **Representação densa**: Ao contrário das representações esparsas, onde cada palavra é representada por um vetor com muitos zeros, *word embeddings* são vetores densos com valores contínuos.
- ***Transfer learning***: Os *word embeddings* treinados em um grande *corpus* podem ser usados em outros modelos de aprendizado de máquina para diferentes tarefas de NLP, sem a necessidade de treinar um novo *embedding*.

A codificação de palavras no espaço de *embeddings* pode ser realizada através da construção de um mapeamento entre os valores únicos das entradas e vetores aleatoriamente inicializados. Esses vetores são ajustados durante o treinamento do

modelo para minimizar uma função de custo, permitindo que aprendam representações úteis com base nos dados de treinamento. Esse processo de ajuste ocorre iterativamente, ao longo de múltiplas passagens sobre os dados, até que os *embeddings* reflitam as relações semânticas e contextuais desejadas (HENNER, 2023).

Uma função de perda amplamente utilizada para treinar *embeddings* é a conhecida como *triplet loss* (CHECHIK et al., 2010). Nesse método, cada etapa do treinamento envolve a comparação de uma entrada de referência, denominada âncora (*anchor*), com duas outras entradas: uma positiva, que deve estar próxima da âncora no espaço latente, e uma negativa, que deve estar distante no espaço. O objetivo é otimizar o espaço de *embeddings* de forma que a distância entre a âncora e a entrada positiva seja minimizada, enquanto a distância entre a âncora e a entrada negativa seja maximizada, conforme pode ser visualizado na Figura 24. Esse processo incentiva o modelo a organizar o espaço de *embeddings* de maneira semântica, onde exemplos relacionados são agrupados, enquanto exemplos irrelevantes ou contrastantes são separados (HENNER, 2023).

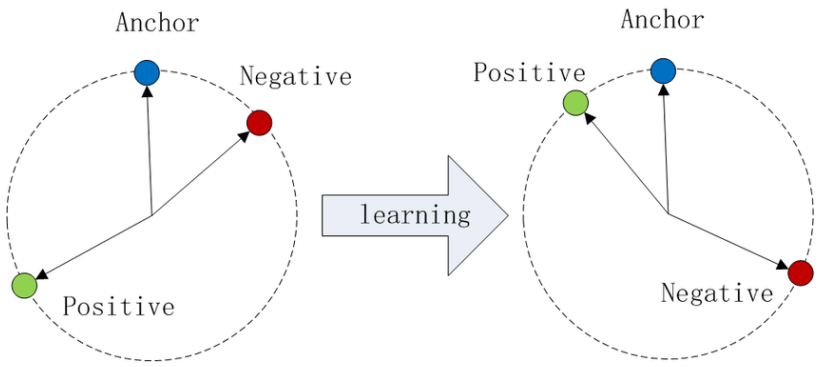

Figura 24. Efeito da minimização da *triplet loss* no treinamento (LI *et al.*, 2017)

Nesse contexto, para ilustrar o processo de criação de espaços de *embeddings*, é possível considerar um conjunto de sentenças que compõem um *corpus* textual. Inicialmente, essas sentenças são segmentadas em suas palavras individuais por meio de um processo conhecido como tokenização, onde cada palavra é tratada de forma independente. A partir disso, cada palavra é mapeada para um vetor único, atribuído de forma aleatória. Nesse estágio inicial, esses vetores funcionam como “impressões digitais” distintas para cada palavra, mas ainda não possuem relação com seu significado ou contexto.

A transformação semântica ocorre quando é introduzido o conceito de contexto. Para isso, é definida uma janela de contexto ao redor de cada palavra alvo, por exemplo, abrangendo n palavras anteriores e n palavras subsequentes. Esse contexto atua como uma espécie de "vizinhança semântica" para a palavra dentro do *corpus*. O objetivo do processo de aprendizado é ajustar os vetores de forma que palavras que compartilham contextos semelhantes passem a ter representações vetoriais próximas, enquanto aquelas sem relação contextual sejam representadas por vetores mais distantes (HENNER, 2023).

Durante o treinamento, cada palavra no *corpus* é analisada repetidamente, ajustando seu vetor com base nos vetores de suas palavras vizinhas (contexto) e afastando-o dos vetores de palavras aleatórias que não compartilham o mesmo contexto, de maneira que esse ciclo é iterado diversas vezes, refinando as representações vetoriais. Ao longo das iterações, os vetores começam a estabilizar, alcançando um ponto de convergência onde os ajustes se tornam insignificantes. Nesse estágio, os vetores deixam de ser distribuições aleatórias e passam a capturar relações significativas, baseadas nas coocorrências de palavras em seus respectivos contextos. O resultado desse processo é a construção de uma estrutura latente no *corpus*, que organiza semanticamente as palavras de acordo com suas relações contextuais. Essa estrutura atua como um mapa semântico, revelando similaridades e associações entre palavras, construído exclusivamente com base em como elas aparecem juntas no *corpus* textual (HENNER, 2023).

2.7 TOKENIZAÇÃO

No campo de Processamento de Linguagem Natural (NLP) e Aprendizado de Máquina, o processo de tokenização é uma etapa fundamental que permite a compreensão e manipulação de texto por algoritmos. Ele consiste na conversão de uma sequência de texto em partes menores, conhecidas como *tokens*. Esses *tokens* podem ser tão pequenos quanto caracteres individuais ou tão grandes quanto palavras ou até frases curtas, dependendo do nível de granularidade necessário para a tarefa em questão (DATACAMP, 2024).

A tokenização desempenha um papel crítico, pois, ao dividir o texto em partes menores, facilita a análise pelas máquinas. Isso ocorre porque, ao contrário dos seres humanos, que conseguem interpretar o texto de maneira intuitiva e global, as máquinas

precisam de estruturas formais e organizadas para entender as nuances da linguagem. Nesse ínterim, cada *token* se torna uma unidade de informação que os modelos podem processar e utilizar em análises mais complexas (DATACAMP, 2024). Dentre os principais tipos de tokenização é possível citar:

1. **Tokenização por palavras**: Nesse método, o texto é dividido em palavras ou subpalavras. Esse tipo de tokenização é comum quando se trabalha com modelos que exigem o significado semântico das palavras ou frases. Por exemplo, na frase "O céu está azul", o processo de tokenização resultaria em quatro tokens: ["O", "céu", "está", "azul"]. Contudo, existem desafios relacionados a essa abordagem, como o tratamento de contrações ("não" em vez de "n" + "ão") e pontuações, que precisam ser abordados adequadamente para evitar que o modelo perca o contexto.

2. **Tokenização por caracteres**: Nesse caso, o texto é dividido em seus caracteres constituintes. Esse método pode ser útil em situações em que o detalhe fino da palavra é importante, como quando se trabalha com idiomas onde a estrutura morfológica desempenha um papel essencial, ou quando precisa-se lidar com caracteres raros ou palavras fora do vocabulário padrão (como nomes próprios).

Além desses, técnicas mais avançadas, como a tokenização baseada em subpalavras, têm ganhado destaque na literatura, especialmente em modelos modernos como o BERT (*Bidirecional Encoder Representations from Transformers*) ou o GPT (*Generative Pre-trained Transformer*). Essas técnicas dividem as palavras em subunidades menores, como prefixos ou sufixos, capturando melhor o significado das palavras desconhecidas ou complexas. Por exemplo, em vez de tratar a palavra "incrível" como um único token, a tokenização por subpalavras poderia dividi-la em ["in", "crível"], facilitando o reconhecimento de palavras relacionadas como "incrível" e "incredulidade".

A importância da tokenização reside no fato de que, ao transformar o texto em *tokens*, os algoritmos conseguem identificar padrões de maneira mais eficaz. O reconhecimento desses padrões é fundamental para tarefas como análise de sentimentos, tradução automática, geração de texto e diversas outras aplicações em NLP. Por exemplo,

ao identificar a palavra “feliz” e associá-la a sentimentos positivos, um modelo pode inferir o tom geral de um texto, ou receber uma sequência de eventos e inferir se a sequência corresponde a um conjunto de eventos que podem levar ao desarme de um reator nuclear.

Outro aspecto crucial da tokenização é que ela ajuda a preservar o contexto do texto, uma vez que, mesmo ao dividir as frases em *tokens*, é possível manter as relações entre as palavras. Técnicas mais modernas de NLP dentro do contexto de representações contextuais entre palavras, como *word embeddings* tais como Word2Vec ou GloVe, ajudam a capturar essas relações, representando palavras em um espaço vetorial onde palavras com significados semelhantes estão localizadas próximas umas das outras.

Dessa forma, a tokenização não apenas torna o texto mais compreensível para as máquinas, mas também contribui para melhorar a precisão dos modelos de Aprendizado de Máquina, fornecendo uma base sólida para tarefas de processamento de linguagem. Sem uma tokenização eficaz, os modelos teriam dificuldade em lidar com a complexidade e a ambiguidade da linguagem humana (DATACAMP, 2024).

2.8 ALGORITMOS DE CLASSIFICAÇÃO

2.8.1 PANORAMA GERAL

A classificação é uma técnica fundamental para viabilizar a Inteligência Artificial automatizada, orientada por dados e capaz de ser utilizada como ferramenta de apoio à tomada de decisão, com inúmeras aplicações em áreas como ciência, tecnologia, medicina e negócios. De maneira geral, classificação é a tarefa de atribuir o rótulo de classe correto para uma determinada entrada (BIRD *et al.*, 2009). Em tarefas básicas de classificação, cada entrada é considerada de forma isolada de todas as outras, e o conjunto de rótulos é definido antecipadamente. Alguns exemplos de tarefas de classificação incluem:

- Decidir se um e-mail é *spam* ou não.
- Determinar qual é o tópico de um artigo de notícias, a partir de uma lista fixa de áreas como “esportes”, “tecnologia” ou “política”.

- Identificar se uma mensagem transmite um tom de “felicidade”, “tristeza”, “ironia”, etc.

A tarefa de classificação em Aprendizado de Máquina apresenta diversas variações que ampliam sua complexidade e aplicabilidade, como por exemplo na classificação multiclasse, que consiste na atribuição de uma instância a uma dentre múltiplas classes possíveis. Algoritmos como árvores de decisão, Naive Bayes e redes neurais são naturalmente adequados para esse tipo de classificação. Para modelos originalmente concebidos para classificação binária, como as Máquinas de Vetores de Suporte (SVM), existem extensões que permitem seu uso em contextos multiclasse, utilizando estratégias como “um contra todos” ou “um contra um” (BIRD *et al.*, 2009). Outra variação importante é a classificação de sequência, que envolve a classificação conjunta de uma sequência de entradas, levando em consideração a interdependência entre os elementos. Essa abordagem é essencial em tarefas onde a ordem e a relação entre os componentes da sequência influenciam diretamente a categorização final, como na análise de séries temporais ou processamento de linguagem natural (BIRD *et al.*, 2009).

Um classificador é definido como supervisionado quando é desenvolvido com base em um conjunto de treinamento composto por instâncias de entrada associadas aos seus respectivos rótulos corretos, ou seja, o algoritmo aprende a partir de exemplos rotulados previamente, desenvolvendo a capacidade de generalizar e realizar previsões para novos dados não vistos e permitindo que o algoritmo infira padrões e relações presentes nos dados rotulados. O processo de aprendizado supervisionado segue uma estrutura onde os dados de entrada passam por várias etapas de processamento até que o modelo produza um rótulo de saída para cada nova entrada, sendo dividido em duas fases principais: treinamento e teste.

Durante a fase de treinamento, o algoritmo é alimentado com um conjunto de dados de entrada juntamente com os rótulos correspondentes, permitindo que ele aprenda as correlações entre as características das entradas e as classes associadas. Esse aprendizado é realizado por meio da minimização de uma função de perda, que quantifica a discrepância entre as previsões do modelo e os rótulos reais, ajustando iterativamente os parâmetros internos do modelo para aprimorar sua precisão.

Na subsequente fase de teste ou validação (ou predição), o modelo treinado é avaliado utilizando um conjunto de dados não visto anteriormente, desprovido de rótulos. Nessa etapa, o modelo aplica o conhecimento adquirido durante o treinamento para prever os rótulos das novas entradas, atribuindo à cada instância a classe com maior probabilidade de correspondência. A eficácia do modelo é então mensurada por meio de métricas de desempenho, como acurácia, precisão, *recall* e *F1-score*, que avaliam sua capacidade de generalização para dados desconhecidos.

A Figura 25 ilustra esquematicamente o funcionamento de um algoritmo de classificação supervisionado. É possível observar que, durante a fase de treinamento, o algoritmo recebe tanto as observações de entrada quando seus respectivos rótulos esperados na saída, aprendendo a correlação entre eles. Na fase de teste ou predição, o algoritmo recebe somente os dados de entrada, sendo responsável nesse caso por retornar a classe com maior probabilidade de estar associada a esses dados com base no conhecimento adquirido.

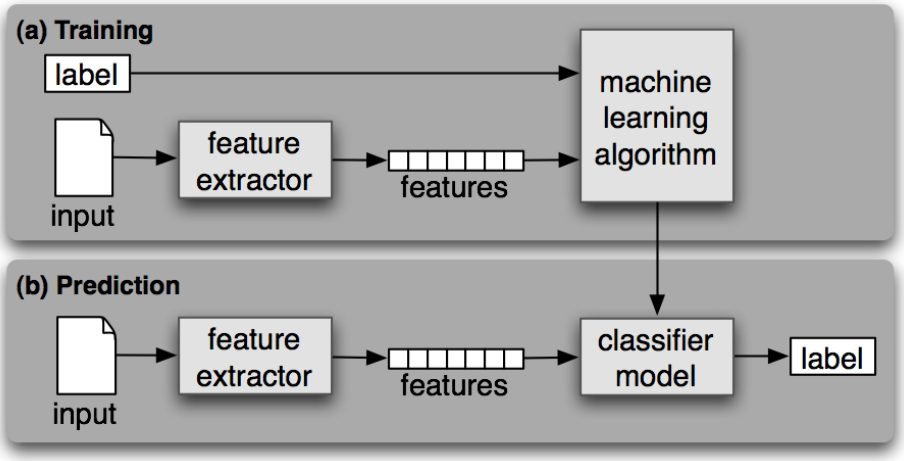

Figura 25. *Framework* utilizado por um modelo de classificação (BIRD *et al.*, 2009)

Quando a dimensão dos dados ou o tamanho da amostra se tornam grandes, surgem problemas de precisão e escalabilidade. Os conjuntos de dados modernos tendem a ser de alta dimensionalidade, por mais que em alguns casos se tenham tamanhos de amostra pequenos. O problema da alta dimensionalidade ocasionou na criação de um sub-ramo da área de gestão de dados denominada *Big Data*, que se refere a conjuntos de dados extremamente grandes e complexos que são difíceis de processar e analisar usando

métodos tradicionais de gerenciamento de dados. O conceito envolve não apenas o volume de dados, mas também a velocidade com que eles são gerados, a variedade de fontes (como dados estruturados, semiestruturados e não estruturados), a veracidade (a qualidade e a confiabilidade dos dados) e o valor que pode ser extraído deles. Esses dados geralmente contêm muitas características irrelevantes, tornando necessárias técnicas de seleção de características (ou *features*) e redução de dimensionalidade antes de aplicar classificadores tradicionais, que geralmente requerem um grande volume amostral. Embora algumas pesquisas se concentrem no desenvolvimento de métodos capazes de classificar dados de alta dimensão sem a seleção de características, o desenvolvimento de classificadores com melhor precisão e eficiência ainda são temas de pesquisa na literatura (LI & LIU, 2016; RAMÍREZ-GALLEGO *et al.*, 2017).

Diversas áreas de pesquisa científica e do cotidiano são compostas por dados em grande escala e/ou desbalanceados. Alcançar uma alta precisão de classificação, escalabilidade e um equilíbrio entre precisão e *recall* (no caso de dados desbalanceados) é um grande desafio, tanto de modelagem e pré-processamento dos quantos quanto de *design* dos modelos computacionais. Embora diversos métodos clássicos de análise de dados e classificadores tenham sido adaptados para lidar com grandes volumes de dados ou dados desbalanceados, seu desempenho ainda está em constante desenvolvimento. Existe uma necessidade crescente de desenvolver técnicas eficazes e escaláveis de mineração de dados e previsão, adequadas para dados em grande escala e/ou desbalanceados (RAMÍREZ-GALLEGO *et al.*, 2017).

Dentre os diversos métodos de classificação listados na literatura, foi optado neste trabalho por utilizar um conjunto de diferentes algoritmos de classificação consolidados e realizar um comparativo de desempenho e capacidade de generalização entre eles: Ridge (HOERL & KENNARD, 1970; HILT & SEEGRIST, 1977), Floresta Aleatória (HO, 1995; BREIMAN, 2001), Naive Bayes (HAND & YU, 2001; CARUANA & NICULESCU-MIZIL, 2006), k-Nearest Neighbors (FIX & HODGES, 1951; COVER & HART, 1967; FIX & HODGES, 1989) e SGD (FERGUSON, 1982; BOTTOU & BOUSQUET, 2011). Esse comparativo servirá como um *benchmark* para se avaliar o quão facilmente separáveis são os dados de sequências de evento em relação a sua respectiva classificação em *Trip* ou Operação Normal.

2.8.2 RIDGE

O algoritmo de classificação *Ridge* ou regularização de Tikhonov (HOERL & KENNARD, 1970; HILT & SEEGRIST, 1977) é um algoritmo de Aprendizado de Máquina projetado para tarefas de classificação multiclasse, que combina conceitos de técnicas de classificação convencionais juntamente com a regressão Ridge, caracterizando-o como um modelo linear, ou seja, mantém a linearidade na relação entre as variáveis independentes e a variável dependente (HOERL & KENNARD, 1970).

A regressão Ridge é uma técnica de regularização utilizada em modelos de regressão linear, especialmente eficaz em situações em que as variáveis independentes são altamente correlacionadas, um problema conhecido como multicolinearidade. Os métodos destinados à regressão nos quais se espera que o valor alvo seja capaz de ser representado como uma combinação linear, como Mínimos Quadrados Ordinários (OLS) (do inglês, *Ordinary Least Squares*), Ridge, Lasso, Elastic-Net, dentre outros, se baseiam no cálculo de um valor alvo ou valor previsto, denotado como $\hat{y}$, medido como uma combinação linear das variáveis do problema, conforme demonstrado na equação (33). A partir dessa equação, é definido um vetor *w*, que corresponde ao vetor de coeficientes do algoritmo, descrito na equação (34) e um escalar w_0, definido como o ponto de interseção, descrito na equação (35).

$$\hat{y}(w, x) = w_0 + w_1 x_1 + \cdots + w_p x_p \tag{33}$$

$$w = (w_1, \cdots, w_p) \tag{34}$$

$$w_0 = \hat{y}(w, 0) \tag{35}$$

Dentro desse escopo, a técnica de regressão Ridge foi desenvolvida como uma resposta à imprecisão dos estimadores de mínimos quadrados quando os modelos de regressão linear apresentam variáveis independentes que estão fortemente correlacionadas (CORTES & VAPNIK, 1995; SUYKENS & VANDEWALLE, 1999). O método tradicional de Regressão Linear ajusta um modelo linear com coeficientes para minimizar a soma residual dos quadrados entre os alvos observados no conjunto de dados e os alvos previstos pela aproximação linear ao utilizar um método de otimização para

minimizar uma função de custo. Matematicamente ele é utilizado para resolver um problema da forma representada pela equação (36).

$$\min_{w} \|Xw - y\|_2^2 \tag{36}$$

As estimativas dos coeficientes para Mínimos Quadrados Ordinários baseiam-se na independência das variáveis. Quando essas variáveis são correlacionadas e as colunas da matriz de *design* (representação matricial das variáveis independentes em modelos, onde cada linha da matriz corresponde a uma observação e cada coluna representa uma variável preditora) têm uma dependência aproximadamente linear, a matriz de *design* torna-se próxima de uma matriz singular e, como resultado, a estimativa dos mínimos quadrados torna-se altamente sensível a erros aleatórios na variável alvo observada, gerando uma alta variância. A partir disso, o método de regressão Ridge aborda alguns dos problemas de convergência dos Mínimos Quadrados Ordinários, impondo uma penalidade no tamanho dos coeficientes (HOERL & KENNARD, 1970). Os coeficientes do modelo de regressão Ridge são ajustados para minimizar uma soma residual de erros quadráticos com uma penalidade adicional, representada pela equação (37). O parâmetro de regularização (α) controla a intensidade dessa penalidade: quanto maior o valor de α, mais os coeficientes são reduzidos em magnitude, tornando o modelo mais resistente à colinearidade entre as variáveis.

$$\min_{w} \|Xw - y\|_2^2 + \alpha\|w\|_2^2 \tag{37}$$

O principal objetivo desse algoritmo é prevenir o *overfitting*, que ocorre quando um modelo se torna excessivamente complexo e se ajusta ao ruído dos dados em vez do sinal subjacente. Para mitigar esse problema, o classificador Ridge adiciona um termo de penalização à função de perda, que corresponde à soma dos quadrados dos coeficientes do modelo. Essa abordagem força os coeficientes a permanecerem pequenos, melhorando assim a capacidade de generalização do modelo para novos dados. O funcionamento do algoritmo Ridge começa com a transformação das variáveis-alvo em valores binários, tipicamente +1 e -1. Em seguida, o algoritmo treina um modelo de regressão Ridge utilizando a função de perda Erro Quadrático Médio (MSE) com um termo de penalização L2. A equação da função de custo utilizada no classificador Ridge pode ser expressa pela

equação (38), e o cálculo do vetor de rótulos (*labels*) previstos durante o processo de otimização dos pesos do algoritmo é expresso pela equação (39):

$$\text{Função de erro} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \alpha\sum_{j=1}^{p} w_j^2 \tag{38}$$

$$\hat{y}_i = X_i w + b \tag{39}$$

onde y_i são as medições reais, $\hat{y}_i$ são as medições previstas, n é o número total de amostras, p é o número de características (variáveis independentes) e w_j são os coeficientes ajustados do modelo. O parâmetro α controla a força da regularização; valores mais altos aumentam a penalização sobre os coeficientes, reduzindo assim o risco de *overfitting*, porém diminuindo a capacidade de generalização do modelo. Sendo assim, alterar o parâmetro α controla o *trade-off* entre viés e variância, e corresponde ao principal hiperparâmetro a ser definido no modelo. Durante a predição, se o valor previsto for menor que 0, o classificador atribui a classe -1; caso contrário, atribui +1. Dessa forma, pela equação (38), é possível observar que o argumento ressaltado em verde (à esquerda) corresponde à parcela representada pelo Erro Quadrático Médio, enquanto a parcela ressaltada em laranja (à direita) corresponde ao termo de penalização que aplica a regularização L2.

A seguir pode ser observado o pseudocódigo do algoritmo de classificação Ridge, onde é possível destacar que o algoritmo possui uma solução analítica, geralmente utilizada quando a matriz de características possui uma baixa dimensionalidade, podendo ser comparada à solução analítica do método dos Mínimos Quadrados Ordinários. Nesse caso, o vetor de coeficientes e o ponto de intercepto podem ser calculados algebricamente através de transposições, inversões e produtos entre matrizes. Com isso, é importante destacar que, quanto maior for a dimensionalidade da matriz *X*, maior será a complexidade computacional do algoritmo ao utilizar a solução analítica, tornando mais viável utilizar um algoritmo de otimização para minimizar a função de custo do algoritmo Ridge, fazendo com que o uso da solução numérica seja mais conveniente.

Input:
X: Matriz de características (n_samples × n_features)
y: Vetor de rótulos (n_samples x 1), convertido para {+1, -1} para classificação binária

α: Parâmetro de regularização (controla o peso da penalização L2)

Output:
w: Vetor de coeficientes ajustados (n_features x 1)
b: Intercepto (viés) ajustado

1. Inicialize:
- w ← vetor de pesos inicial (zeros ou valores pequenos)
- b ← viés inicial (0)

2. Normalize os dados:
- Opcional: Padronize X para ter média 0 e desvio padrão 1

3. Defina a função de custo:
- $L(w, b) = \|y - (Xw + b)\|^2 + \alpha\|w\|^2$
- Onde $\|\cdot\|^2$ representa a soma dos quadrados dos erros e $\alpha\|w\|^2$ é a penalização L2.

4. Solução Numérica:
- Utilizar um método para minimizar $L(w, b)$, como SVD (do inglês, *Singular Value Decomposition*), decomposição de Cholesky, Gradiente Conjugado para Matrizes Esparsas, LSQR (do inglês, *Dedicated Regularized Least-Squares Routine*) (PAIGE & SAUNDERS, 1982), SAG (do inglês, *Stochastic Average Gradient descent*), L-BFGS-B (do inglês, *Limited-memory Broyden–Fletcher–Goldfarb–Shanno Bound*) (BYRD *et al.*, 1995)

5. Solução Analítica:
- Ajuste w e b diretamente resolvendo as equações a seguir (GAO & TSAY, 2024):
  - $w = (X^T X + \alpha I)^{-1} X^T y$
  - $b = \frac{1}{n}\sum_{i-1}^{n}(y_i - X_i w)$

6. Predição:
Para prever o rótulo de uma nova amostra X_{pred}:
- Calcular $y_{pred} = X_{pred} w + b$
- Atribuir o rótulo com base no sinal de y_{pred}:
  - Se $y_{pred} \geq 0$, prever classe positiva
  - Se $y_{pred} < 0$, prever classe negativa

7. Fim.

Dessa forma, a partir da Figura 26, é possível observar uma representação gráfica do impacto do termo de regularização na determinação dos coeficientes do algoritmo Ridge. Neste gráfico, o eixo horizontal representa o parâmetro de regularização α, que controla a força da penalização, sendo que, quanto maior o valor desse parâmetro, maior

a penalização aplicada aos coeficientes do modelo. O eixo vertical mostra os valores dos coeficientes do algoritmo Ridge, de forma que cada linha colorida do gráfico representa um coeficiente diferente do modelo Ridge em função da regularização. Para valores baixos de α, ou seja, quando a regularização é fraca, os coeficientes se aproximam dos coeficientes da regressão linear sem regularização, que podem ser grandes ou até mesmo instáveis e, à medida que α aumenta, ou seja, a regularização se torna mais forte, os coeficientes começam a ser reduzidos, de forma que o modelo começa a penalizar mais fortemente os coeficientes grandes, o que leva a uma solução mais estável. Para valores muito altos de α, quase todos os coeficientes tendem a se aproximar de zero, indicando que a regularização é tão forte que o modelo praticamente ignora as variáveis de entrada. Esse comportamento é esperado no Ridge, pois a regularização impede que o modelo se ajuste excessivamente aos dados de treinamento, o que pode ser útil para evitar o *overfitting*. O parâmetro de regularização é, portanto, um fator crucial para controlar o equilíbrio entre o ajuste aos dados e simplicidade do modelo.

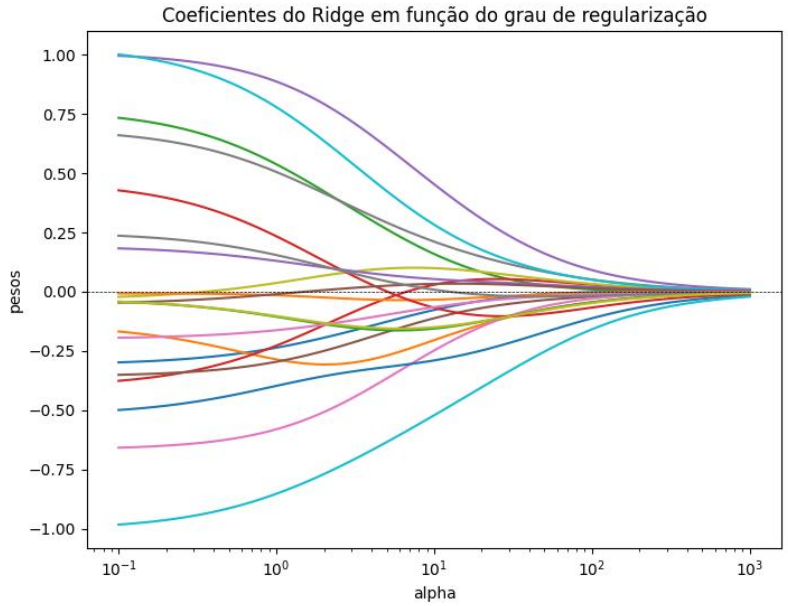

Figura 26. Impacto do termo de regularização na determinação dos coeficientes do algoritmo Ridge (De autoria própria)

Para problemas multiclasse, o algoritmo Ridge utiliza uma abordagem “um contra todos” (*one-vs-rest*), onde um classificador binário é treinado para cada classe. Entre as vantagens do classificador Ridge, destaca-se sua capacidade de prevenir o *overfitting* por meio da regularização L2, além de ser eficaz no tratamento da multicolinearidade, que

ocorre quando as características são altamente correlacionadas. Isso evita problemas comuns em modelos lineares como o método dos Mínimos Quadrados Ordinários, que necessita que a matriz $X^T X$ seja invertível, o que requer que as colunas de X sejam linearmente independentes. Caso contrário, o produto $X^T X$ se aproxima de uma matriz singular, resultando em estimativas de coeficientes altamente sensíveis a erros aleatórios nos dados observados, aumentando consideravelmente a variância do modelo (HILT & SEEGRIST, 1977).

A Figura 27 demonstra um exemplo do impacto do parâmetro de regularização na capacidade de generalização do algoritmo Ridge. Ao ajustar um algoritmo Ridge a partir de uma amostra do conjunto de dados de identificação de *Trip* utilizado neste trabalho e visualizar os resultados da classificação, o gráfico da Figura 27 exibe o percentual de acertos do modelo nos conjuntos de treinamento (curva vermelha) e teste (curva azul) em função do aumento de α. Nota-se que, para valores baixos de α, a taxa de acertos é elevada em ambos os conjuntos de treinamento e teste, indicando que o modelo está capturando bem os padrões dos dados. No entanto, à medida que α aumenta, a precisão do treinamento diminui gradualmente, refletindo a penalização dos coeficientes do modelo, o que reduz sua complexidade e mitiga o o*verfitting*. No conjunto de teste, a precisão também apresenta uma tendência de queda, mas com menor impacto inicial, até que, em valores mais altos de α, ocorre uma queda acentuada, evidenciando uma perda significativa da capacidade de generalização.

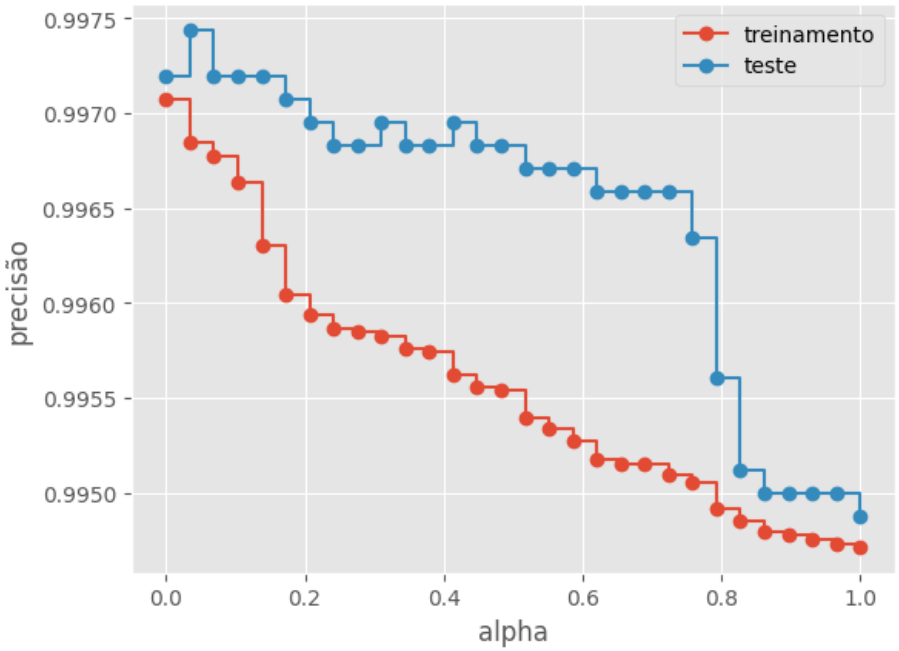

Figura 27. Impacto do termo de regularização na capacidade de generalização do algoritmo Ridge (De autoria própria)

O comportamento visualizado na Figura 27 destaca a influência crítica da regularização no ajuste de um modelo Ridge: enquanto valores muito baixos de α podem levar ao *overfitting*, valores mais altos prejudicam a expressividade do modelo, comprometendo sua precisão na predição antecipada do *Trip* do reator nuclear analisado. Dessa forma, é necessário realizar uma análise prévia para verificar o intervalo de valores do parâmetro de regularização que melhor promovam um balanço entre o viés e a variância explicada pelo modelo.

2.8.3 FLORESTA ALEATÓRIA

O algoritmo Floresta Aleatória (do inglês, *Random Forest*) (HO, 1995; BREIMAN, 2001) é uma técnica amplamente utilizada no campo do aprendizado de máquina, com aplicações tanto em tarefas de classificação quanto de regressão. Este método se baseia na construção de múltiplas árvores de decisão, combinando suas previsões para alcançar resultados mais consistentes e precisos. As árvores de decisão que compõem o algoritmo são construídas utilizando um modelo de “se-então”, no qual o espaço de características é particionado iterativamente com base em critérios de divisão e pureza dos nós, como impureza de Gini ou entropia, sendo cada árvore criada a partir

de um subconjunto aleatório dos dados de treinamento, selecionado por meio de amostragem com reposição (também conhecido como *bootstrap sampling*). Além disso, em cada nó da árvore, um subconjunto aleatório de características é escolhido para considerar as possíveis divisões, promovendo diversidade entre as árvores.

Em seu processo de aprendizagem, o algoritmo começa com a seleção aleatória de amostras do conjunto de dados original. Essa técnica, conhecida como *bootstrap*, permite que uma fração das amostras sejam repetidas, garantindo diversidade nas árvores criadas. Para cada árvore, o algoritmo escolhe aleatoriamente um subconjunto de variáveis que serão usadas para determinar os nós da árvore, sendo esses nós as condições que dividem os dados em diferentes ramos. As variáveis são selecionadas com base em critérios como impureza de Gini ou ganho de informação, que ajudam a identificar as melhores divisões possíveis (BREIMAN *et al.*, 1984).

O critério de impureza de Gini é uma métrica amplamente utilizada para avaliar a qualidade de divisões em algoritmos baseados em árvores de decisão, como a Floresta Aleatória e CART (*Classification and Regression Trees*). Ele mede o grau de homogeneidade das classes em um nó de decisão, auxiliando na escolha da melhor divisão de dados em cada etapa do treinamento da árvore (BREIMAN *et al.*, 1984). Matematicamente, a impureza de Gini é calculada pela equação (40):

$$\mathrm{G}(t) = 1 - \sum_{i=1}^{C} p_i^2 \tag{40}$$

onde C é o número total de classes (*features*) do problema e p_i é a proporção de amostras pertencentes à classe i no nó t. A partir da equação (40), é possível observar que o menor valor que a função $\mathrm{G}(t)$ pode assumir em um nó é 0, o que ocorre quando todas as amostras em um nó pertencem a uma única classe ($p_i = 1$ para uma classe e $p_i = 0$ para todas as outras). Nesse caso, diz-se que o nó é puro, e não há incerteza sobre as classificações dentro do nó. A partir dessa mesma equação é possível observar que o valor máximo ocorre quando as amostras estão uniformemente distribuídas entre todas as classes ($p_i = \frac{1}{C}$ para todas as classes) e, nesse cenário, há alta incerteza, pois as amostras são igualmente prováveis de pertencer a qualquer classe.

Durante o treinamento de uma árvore de decisão, a métrica de Gini é usada para determinar a melhor divisão em cada nó. O objetivo é minimizar a impureza de Gini resultante, maximizando assim a homogeneidade das classes nas ramificações. Para avaliar a qualidade de uma divisão em dois subconjuntos (t_L e t_R) que representam a divisão entre esquerda e direita, respectivamente, a partir de um nó, calcula-se a impureza de Gini ponderada através da equação (41):

$$\mathrm{G}_{divisão} = \frac{N_L}{N}\mathrm{G}(t_L) + \frac{N_R}{N}\mathrm{G}(t_R) \tag{41}$$

onde $\mathrm{G}(t_L)$ e $\mathrm{G}(t_R)$ são as impurezas de Gini dos subconjuntos esquerdo e direito, respectivamente, N_L e N_R são as quantidades de amostras em t_L e t_R, e N é o número total de amostras no nó atual. A divisão escolhida será aquela que minimiza $\mathrm{G}_{divisão}$, resultando em uma separação mais clara entre as classes (BREIMAN *et al.*, 1984).

Uma vez que a primeira árvore é construída, o algoritmo repete o processo para criar as demais árvores que irão compor a floresta, sendo cada árvore única devido à aleatoriedade na seleção das amostras e das variáveis. Essa diversidade é crucial para a eficácia do algoritmo, pois evita o problema do *overfitting*, onde um modelo se torna excessivamente ajustado aos dados de treinamento e perde capacidade preditiva em novos dados (BREIMAN, 2001).

Após a construção e ajuste das árvores, o modelo é capaz de realizar previsões com novos dados. Para problemas de classificação, cada árvore fornece um voto para uma classe específica, e a classe que recebe o maior número de votos é escolhida como resultado. Em problemas de regressão, a previsão final é obtida pela média dos valores previstos por todas as árvores. Essa abordagem em grupo melhora a precisão geral do modelo, pois combina as decisões individuais das árvores (BREIMAN, 2001).

A seguir pode ser observado o pseudocódigo do algoritmo Floresta Aleatória, enfatizando as entradas e saídas do algoritmo, assim como o processo iterativo de aprendizado das árvores que compõem o modelo de classificação:

Input:
X: Matriz de características (n_samples × n_features)
y: Vetor de rótulos (n_samples x 1)

T: Número de árvores na floresta
m: Número de amostras utilizadas para treinar cada árvore (sendo $m \leq$ n_samples)
k: Número de características a serem selecionadas aleatoriamente em cada nó de decisão

Output:
y_{pred}: rótulo da classe prevista

1. Construção da floresta: Para $t = 1$ até T
- Gere um conjunto de treinamento D_t com m amostras selecionadas aleatoriamente do conjunto D com reposição
- Construa uma árvore de decisão h_t com D_t utilizando o seguinte procedimento:
  - Em cada nó:
    - Selecione aleatoriamente k características do conjunto total de características
    - Escolha a melhor característica entre as k selecionadas para dividir o nó, com base em uma métrica, como o Critério de Impureza de Gini
  - Continue dividindo até atingir um critério de parada, como: O número mínimo de amostras em um nó folha, a profundidade máxima da árvore ou a pureza dos nós
- Armazene a árvore h_t no modelo de floresta

2. Classificação de um novo dado x:
- Para cada árvore h_t na floresta ($t = 1, 2, ..., T$) classifique x com a árvore h_t, produzindo a previsão y_t
- Combine as previsões y_t de todas as árvores, utilizando votação majoritária para determinar a classe final y de x

3. Fim.

A combinação das etapas descritas no pseudocódigo do algoritmo permite que a Floresta Aleatória se beneficie do princípio do aprendizado em conjunto (*ensemble learning*), onde a agregação de múltiplos modelos tende a superar o desempenho de um modelo individual. Esse comportamento é particularmente vantajoso em cenários onde há alta variância ou colinearidade nos dados, pois a Floresta Aleatória reduz o risco de *overfitting* e melhora a capacidade de generalização (BREIMAN, 2001).

Além disso, o algoritmo oferece vantagens adicionais, como a capacidade de estimar automaticamente a importância das características do problema, permitindo identificar quais variáveis têm maior influência na predição. Essa característica o torna valioso em contextos de análise exploratória de dados, aprendizado supervisionado e na seleção de variáveis, pois ao identificar as variáveis mais relevantes para um problema é possível reduzir a dimensionalidade sem comprometer a capacidade de generalização, permitindo o desenvolvimento de modelos mais simples e, ao mesmo tempo, escaláveis.

Embora o algoritmo seja computacionalmente mais intensivo do que uma única árvore de decisão, sua eficiência pode ser aprimorada por meio de técnicas de paralelização, já que cada árvore pode ser treinada de forma independente.

Além disso, um dos hiperparâmetros mais relevantes para a modelagem de um algoritmo baseado em árvores de decisão é a Poda por Complexidade de Custo Mínimo (*Minimal Cost-Complexity Pruning*), ou *ccp_alpha*, que está relacionado à poda das árvores individuais que compõem a floresta, sendo utilizado para reduzir a profundidade das árvores e o número de nós, atuando diretamente na mitigação do *overfitting*. Esse parâmetro define um limite mínimo de complexidade para os nós da árvore de decisão, "podando" de maneira recursiva os nós menos relevantes, tendo como objetivo principal eliminar divisões que agregam pouca ou nenhuma melhoria significativa na capacidade preditiva do modelo.

Durante o treinamento das árvores de decisão, cada divisão aumenta a complexidade da árvore, tornando-a mais capaz de se ajustar aos dados de treinamento, porém, nem todas as divisões contribuem de maneira relevante para a generalização do modelo. O parâmetro *ccp_alpha* define um critério para eliminar essas divisões menos relevantes de forma que, à medida que seu valor aumenta, mais divisões são removidas, resultando em árvores mais simples e rasas. Por outro lado, valores baixos permitem que as árvores cresçam mais profundamente, aumentando a complexidade do modelo.

A Figura 28 exibe a relação entre a impureza total das folhas e o valor do *ccp_alpha* efetivo em um algoritmo de Floresta Aleatória ajustado com uma amostra de exemplo baseada no conjunto de dados para predição de *Trip* utilizado neste trabalho. O eixo horizontal representa o *ccp_alpha* efetivo, que representa o valor de regularização aplicado às árvores individuais, enquanto o eixo vertical representa a impureza total das folhas, que indica o nível de desorganização ou incerteza nas previsões das árvores após a poda. A partir de uma análise do gráfico, é possível observar que, conforme o parâmetro *alpha* efetivo aumenta, a impureza total dos nós também aumenta, pois, valores maiores de *ccp_alpha* fazem com que mais divisões das árvores sejam podadas, eliminando nós que poderiam estar reduzindo a impureza. Como resultado, a complexidade das árvores

diminui, e a impureza total das folhas aumenta, pois menos ajustes finos podem ser feitos nos dados.

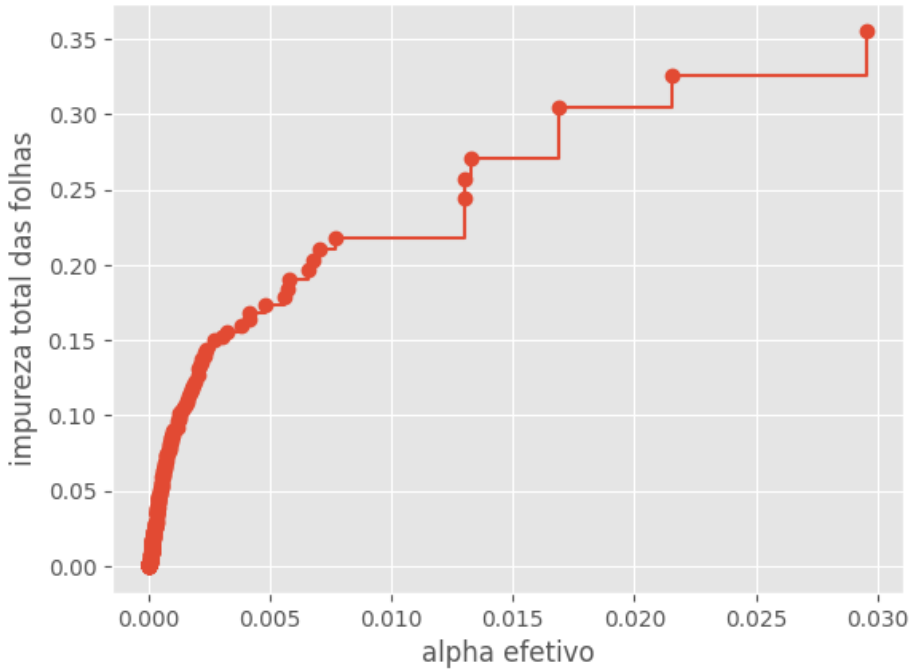

Figura 28. Impacto do parâmetro *ccp_alpha* na impureza dos nós do algoritmo Floresta Aleatória (De autoria própria)

A Figura 29 apresenta dois gráficos que ilustram o efeito do parâmetro *ccp_alpha* na estrutura das árvores dentro de uma Floresta Aleatória, de forma que o primeiro gráfico mostra a relação entre o número de nós (folhas) e *alpha*, enquanto o segundo exibe a relação entre a profundidade das árvores e *alpha*. Observa-se que, para valores muito baixos de *alpha* (tendendo a zero), as árvores possuem um grande número de nós e profundidades elevadas, indicando modelos mais complexos e potencialmente sobreajustados, característica comumente observada em árvores de decisão quando não se impõe um limite em relação ao crescimento das árvores. À medida que alpha aumenta, tanto o número de nós quanto a profundidade caem drasticamente, o que reflete o efeito da poda na simplificação da estrutura das árvores. Para valores mais altos de *alpha*, as árvores se tornam extremamente rasas e possuem poucos nós, o que pode comprometer sua capacidade preditiva devido ao *underfitting*.

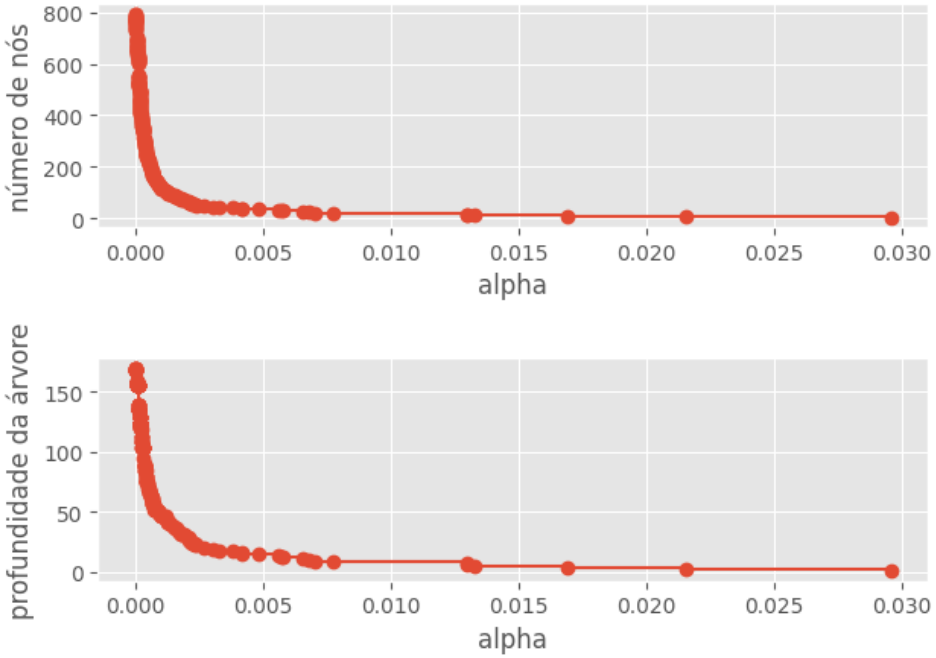

Figura 29. Impacto do parâmetro *ccp_alpha* no número de nós e na profundidade das árvores do algoritmo Floresta Aleatória (De autoria própria)

Dessa forma, a escolha adequada do parâmetro *ccp_alpha* impacta diretamente a capacidade de generalização dos algoritmos baseados em árvore de decisão. Um valor muito pequeno pode levar ao *overfitting*, pois as árvores se tornam excessivamente especializadas nos dados de treinamento, capturando ruídos irrelevantes e *outliers*. Em contrapartida, um valor muito alto pode levar ao *underfitting*, pois árvores muito simplificadas podem não capturar padrões essenciais nos dados, reduzindo a precisão do modelo. Assim, a definição ideal desse parâmetro é fundamental para equilibrar viés e variância, garantindo que o modelo seja robusto e eficiente na predição de novos dados.

2.8.4 NAIVE BAYES

O algoritmo Naive Bayes (HAND & YU, 2001; CARUANA & NICULESCU-MIZIL, 2006) é um modelo probabilístico amplamente utilizado em tarefas de classificação no campo do aprendizado de máquina e análise de dados. Sua abordagem baseia-se no Teorema de Bayes, que permite calcular a probabilidade posterior $P(C|X)$, ou seja, a probabilidade de uma classe C ocorrer dado um conjunto de observações X, a partir da probabilidade a priori $P(C)$, ou seja, a probabilidade de ocorrência de C sem considerar qualquer informação adicional ou característica X, na probabilidade

condicional $P(X|C)$, ou seja, a probabilidade de observar as características *X* dado que a classe *C* foi observada, e na probabilidade marginal $P(X)$, ou seja, a probabilidade de que um conjunto de características *X* seja observado independentemente de qual classe *C* a instância pertence. Matematicamente, a fórmula do Teorema de Bayes pode ser expressa pela equação (42) e os cálculos das probabilidades mencionadas podem ser expressos pelas equações (43), (44) e (45):

$$\mathrm{P}(C|X) = \frac{P(X|C) \cdot P(C)}{P(X)} \tag{42}$$

$$P(C) = \frac{Número\ de\ instâncias\ da\ classe\ C}{Número\ total\ de\ instâncias} \tag{43}$$

$$P(X|C) = \frac{Número\ de\ instâncias\ com\ características\ X\ na\ classe\ C}{Número\ total\ de\ instâncias\ na\ classe\ C} \tag{44}$$

$$P(X) = \sum_{C} P(X|C) \cdot P(C) \tag{45}$$

No contexto do algoritmo, $P(C|X)$ representa a probabilidade de uma classe C ser atribuída a um conjunto de características X, enquanto $P(C)$ é a probabilidade inicial da classe antes de observar as características. O algoritmo Naive Bayes calcula essas probabilidades usando dados históricos, permitindo que ele faça previsões sobre novas instâncias.

O termo "Naive" (ingênuo) no nome do algoritmo refere-se à suposição simplificadora de que todas as características $x_i \in X$ são condicionalmente independentes entre si, dado o conhecimento da classe C, implicando na equação (46):

$$\mathrm{P}(X|C) = \prod_{i=1}^{n} P(x_i|C) \tag{46}$$

Essa simplificação reduz significativamente a complexidade computacional do modelo, tornando-o eficiente e de fácil implementação, mesmo para conjuntos de dados com um número elevado de características. No entanto, essa hipótese de independência entre as classes raramente é válida em cenários do mundo real, onde as características geralmente apresentam algum grau de correlação. Apesar disso, o Naive Bayes

frequentemente demonstra bom desempenho em muitas aplicações práticas, graças à sua robustez e capacidade de generalização (HAND & YU, 2001).

Neste trabalho, foi adotada a variante *Complement Naive Bayes* (RENNIE *et al.*, 2003) (CNB), como uma extensão do algoritmo clássico *Multinomial Naive Bayes* (MNB), sendo particularmente eficaz para lidar com conjuntos de dados desbalanceados, uma característica comum em muitas aplicações de Aprendizado de Máquina, onde as classes possuem distribuições desiguais. O principal diferencial do CNB em relação ao MNB reside na forma como ele calcula as probabilidades condicional e posterior de forma que, enquanto o MNB utiliza diretamente os dados da classe alvo para estimar as probabilidades, o CNB inverte essa perspectiva, considerando as informações complementares de outras classes. Essa abordagem permite ao CNB corrigir vieses introduzidos por classes majoritárias, promovendo uma maior performance na classificação de amostras pertencentes a classes minoritárias. (RENNIE *et al.*, 2003) demonstraram que o CNB frequentemente supera o MNB em tarefas com dados desbalanceados, especialmente em problemas de classificação de texto. Nessas situações, o CNB é capaz de equilibrar a contribuição das classes e reduzir os impactos negativos de distribuições desiguais, resultando em uma melhoria consistente no desempenho geral do modelo.

Dentro do contexto da tarefa de identificação prévia de *Trip* em uma usina nuclear, a escolha da aplicação do CNB dentro das variantes do algoritmo Naive Bayes clássico se torna uma boa opção para se analisar a capacidade de generalização desse algoritmo e sua capacidade de diferenciar sequências de evento anômalas das normais, pois naturalmente é esperado que o número de eventos de desarme do reator e da turbina ocorram com muito menor frequência do que eventos comuns durante a operação. Dessa forma, é esperado que, para esse tipo de problema onde a quantidade de observações de cada classe é consideravelmente desbalanceada, o CNB se prove ser uma ferramenta útil de classificação.

Conforme mencionado, a principal diferença entre o CNB e o MNB reside na forma como as estatísticas são calculadas para determinar os pesos do modelo. No CNB, as estatísticas utilizadas derivam do complemento de cada classe, ou seja, em vez de

estimar os parâmetros com base exclusivamente nos exemplos pertencentes à própria classe, o CNB incorpora informações das outras classes. Essa abordagem faz com que os pesos atribuídos às características sejam menos propensos a variações extremas, garantindo que as previsões se mantenham estáveis, mesmo com conjuntos de dados desbalanceados, além de que a capacidade do modelo de considerar o complemento das classes ao calcular os pesos garante que o modelo não seja excessivamente influenciado por uma única classe dominante, promovendo uma melhor capacidade de generalização (RENNIE *et al.*, 2003). De maneira geral, este algoritmo funciona da seguinte forma:

1. Para cada classe, calcula-se a probabilidade de a instância dada não pertencer essa classe.
2. Após calcular as probabilidades para todas as classes, os valores são comparados, e o menor valor é identificado.
3. O menor valor, que representa a menor probabilidade de a instância não pertencer àquela classe específica, é selecionado. Isso equivale a dizer que a classe com o menor valor calculado tem a maior probabilidade de ser a classe correta.

Dado que o CNB utiliza o complemento das classes, a probabilidade condicional $P(x_i|C_k)$ (a probabilidade de observar a característica x_i dado que a instância pertence à classe C_k) é calculada considerando as informações de todas as outras classes (o complemento de C_k). Para uma característica x_i e uma classe C_k, o complemento de C_k é definido como todos os exemplos que não pertencem a C_k. Matematicamente, o procedimento de cálculo dos pesos do algoritmo é realizado através das equações (47), (48) e (49). A probabilidade condicional estimada ($\hat{\theta}_{ci}$), descrita na equação (47), representa a probabilidade condicional ajustada para a característica i pertencente à classe c, considerando o complemento da classe c.

$$\hat{\theta}_{ci} = \frac{\alpha_i + \sum_{j:y_j \neq c} d_{ij}}{\alpha + \sum_{j:y_j \neq c} \sum_k d_{kj}} \tag{47}$$

$$w_{ci} = \log\left(\hat{\theta}_{ci}\right) \tag{48}$$

$$w_{ci} = \frac{w_{ci}}{\sum_j |w_{cj}|} \tag{49}$$

Na equação (47), as somas são realizadas sobre todos os documentos j que não estão na classe c, o termo d_{ij} é a contagem ou frequência (*TF-IDF*) da característica i no documento j, o termo α é um hiperparâmetro global de regularização e suavização (evita que probabilidades no denominador se tornem zero e estabiliza os cálculos em casos de características muito raras), α_i são parâmetros de suavização específico para cada característica i (garante que mesmo características raramente observadas recebam uma contribuição mínima) onde geralmente $\alpha = \sum_i \alpha_i$, $\sum_{j:y_j \neq c} d_{ij}$ é a soma das frequências de i em todas as classes exceto c (o complemento de c) e $\sum_{j:y_j \neq c} \sum_k d_{kj}$ é a soma total das frequências de todas as características em todas as classes complementares.

A equação (48) indica que os pesos w_{ci} são calculados como o logaritmo das probabilidades ajustadas $\hat{\theta}_{ci}$. O uso da função logarítmica garante estabilidade numérica, especialmente em cenários onde as probabilidades ajustadas assumem valores muito pequenos, de forma que este efeito é particularmente mais evidente em dados de alta dimensionalidade ou quando características raras contribuem significativamente para a modelagem (RENNIE *et al.*, 2003).

Após o cálculo dos pesos logarítmicos, eles passam por um processo de normalização, conforme descrito na equação (49). Nesta etapa, cada peso w_{ci} é dividido pela soma dos valores absolutos dos pesos da classe c para todas as características. Essa normalização garante que os pesos sejam comparáveis entre classes, pois reduz a variabilidade entre os valores absolutos dos pesos das classes, promovendo maior robustez no modelo, sendo uma etapa especialmente importante em cenários com dados desbalanceados, onde classes minoritárias podem ser subrepresentadas nos cálculos de probabilidade. Além disso, caso o algoritmo não realizasse a etapa de normalização, os pesos de classes com mais características ou maiores frequências poderiam dominar a classificação, tornando o modelo enviesado (RENNIE *et al.*, 2003).

O CNB classifica uma instância t atribuindo-a à classe com a menor pontuação calculada, conforme descrito na equação (50). Esse método aborda uma das principais limitações do MNB: a tendência de documentos mais longos dominarem as estimativas de parâmetros. Essa limitação ocorre porque, no MNB, as frequências absolutas das características são utilizadas diretamente, o que pode levar a um viés em favor de

documentos mais extensos, devido à soma acumulada de suas frequências. No CNB, tal viés é mitigado pela utilização de pesos normalizados e pela lógica do complemento das classes, que reduz a influência desproporcional de documentos mais longos.

$$\hat{c} = \text{argmin}_c \sum_i t_i w_{ci} \tag{50}$$

ou seja, um documento é atribuído à classe $\hat{c}$ que minimiza a soma ponderada dos pesos w_{ci} para todas as características i, de forma que o termo $\hat{c}$ é a classe predita para a instância t e o termo t_i é o valor da característica i na instância t (geralmente, a frequência da característica no documento). O cálculo da soma é essencialmente o cálculo de pontuação para cada classe, onde a menor soma indica a classe predita. Além disso, é possível observar, a partir da equação (50), a importância da realização prévia da etapa de normalização, pois garante que as pontuações de classe sejam calculadas em uma escala comum, o que facilita o cálculo da classe predita $\hat{c}$ e assegura que a comparação entre classes seja realizada de forma consistente e menos enviesada.

2.8.5 K-NEAREST NEIGHBORS

O algoritmo k-Nearest Neighbors (kNN) (FIX & HODGES, 1951; COVER & HART, 1967; FIX & HODGES, 1989) é um método de Aprendizado de Máquina utilizado para classificação e regressão, se baseando na ideia de que dados semelhantes estão próximos uns dos outros em um espaço multidimensional. O kNN é classificado como um algoritmo não paramétrico, ou seja, ele não assume nenhuma hipótese sobre a distribuição dos dados, o que o torna particularmente útil em cenários onde a estrutura subjacente dos dados é complexa ou desconhecida, e é um algoritmo supervisionado, o que significa que ele utiliza um conjunto de dados rotulados para fazer previsões, tornando-o dependente de informações prévias sobre as classes ou valores associados aos dados de treinamento. De maneira geral, o funcionamento do algoritmo pode ser descrito através das seguintes etapas principais:

1. O primeiro passo é escolher o valor do parâmetro *K*, que determina o número de vizinhos mais próximos que serão considerados ao classificar ou estimar o valor de um novo ponto de dados. Um valor de *K* pequeno (tendendo a 1) pode levar a um modelo extremamente sensível a ruídos nos dados, resultando em *overfitting*,

enquanto valores muito altos podem diluir a influência de vizinhos relevantes, resultando em *underfitting* e a uma possível perda de informações importantes. De maneira geral, a escolha ideal do parâmetro *K* pode ser feita utilizando técnicas como validação cruzada (*cross validation*), *Grid Search* ou o Método do Cotovelo (*elbow method*) que, através de diferentes abordagens, buscam encontrar um valor ótimo do parâmetro *K*, geralmente um ponto em que continuar aumentando o valor não justifica o custo computacional adicional de processamento.

2. Para classificar um novo ponto, o algoritmo calcula a distância entre esse ponto e todos os pontos existentes no conjunto de dados de treinamento. A distância Euclidiana é a métrica mais comumente usada, mas outras métricas, como a distância de Manhattan, Minkowski ou distância de Hamming (em dados categóricos), também são citadas como alternativas na literatura, a depender do problema (FIX & HODGES, 1989). Com isso, a fórmula da distância Euclidiana entre dois pontos (x_1, y_1) e (x_2, y_2) é dada pela equação (51):

$$\mathrm{d} = \sqrt{(x_2 - x_1)^2 - (y_2 - y_1)^2} \tag{51}$$

3. Com base nas distâncias calculadas, o algoritmo seleciona os *K* pontos mais próximos do novo ponto. Esses pontos formam o conjunto dos vizinhos mais próximos, que será usado para tomar a decisão final.

4. Para problemas de classificação, o kNN atribui ao novo ponto a classe que é mais frequente entre seus *K* vizinhos, sendo uma técnica conhecida como votação majoritária, assim como observado nas árvores de decisão para classificação. Em problemas de regressão, o kNN calcula a média (ou outra métrica de agregação, como a mediana) dos valores de saída dos *K* vizinhos para prever um valor contínuo como predição.

Na prática, o método kNN é um método de aprendizado preguiçoso (*lazy learning*), o que significa que ele armazena todos os dados de treinamento e realiza os cálculos apenas no momento da predição. Isso pode levar a alta complexidade computacional em grandes conjuntos de dados, especialmente no cálculo das distâncias. Para mitigar esse problema, podem ser utilizados algoritmos de indexação, como árvores

KD ou *Ball Trees* (OMOHUNDRO, 1989). Além disso, como o kNN depende de métricas de distância, é sensível à escala dos dados e, para evitar que características com valores maiores dominem os cálculos, é essencial realizar uma etapa de pré-processamento nos dados, como a aplicação de normalização ou padronização das variáveis antes de ajustar os dados ao modelo (FIX & HODGES, 1989).

A seguir pode ser observado o pseudocódigo do algoritmo kNN, enfatizando as entradas e saídas do algoritmo, assim como o processo de aprendizado do modelo de classificação:

Input:

X: Matriz de características (n_samples × n_features)
y: Vetor de rótulos (n_samples x 1)
K: Número de vizinhos mais próximos a serem considerados
d: Métrica de distância, como a distância Euclidiana ou Manhattan
x_{pred}: Novo ponto de dados a ser classificado

Output:

y_{pred}: rótulo da classe prevista

---

1. Inicialização:
   - Receber o conjunto de treinamento *D*, o valor de *K* e o ponto a ser classificado x_{pred}

2. Cálculo das distâncias:
   - Para cada ponto x_i em D:
     - Calcular a distância entre x_{pred} e x_i usando a métrica de distância *d*:

$$\text{distância}(x_{pred}, x_i) = d(x_{pred}, x_i)$$

3. Ordenação dos vizinhos:
   - Ordenar todos os pontos $x_i \in D$ com base nas distâncias calculadas em ordem crescente

4. Seleção dos *K* vizinhos mais próximos:
   - Selecionar os *K* pontos com as menores distâncias

5. Votação majoritária:
   - Determinar a classe mais frequente entre os *K* vizinhos selecionados:
     - Contar o número de ocorrências de cada classe nos *K* vizinhos
     - Escolher a classe com a maior frequência

6. Classificação:
   - Atribuir ao ponto x_{pred} a classe mais frequente obtida no passo anterior

6. Fim.

A partir do pseudocódigo do kNN é possível observar que o algoritmo possui uma característica única em relação a métodos como Ridge, Floresta Aleatória e Naive Bayes: ele não possui pesos e probabilidades ajustáveis ou um processo explícito de treinamento. Em vez disso, o kNN opera como um método de aprendizado onde todas as operações relevantes, como cálculo de distâncias e classificação, ocorrem no momento da predição. Dessa forma, o sucesso do algoritmo depende inteiramente da seleção de *K*, da métrica de distância escolhida e da qualidade dos dados de treinamento, o que torna o kNN simples de implementar, mas sensível a escolhas inadequadas de hiperparâmetros.

2.8.6 SGD

O algoritmo *Stochastic Gradient Descent (*SGD*)* (FERGUSON, 1982; BOTTOU & BOUSQUET, 2011) é uma técnica fundamental no campo do Aprendizado de Máquina, amplamente utilizada para a otimização de modelos em tarefas de classificação, regressão e treinamento de redes neurais. O SGD é uma variação do método de Descida de Gradiente, cujo objetivo principal é minimizar uma função de custo ajustando iterativamente os parâmetros do modelo para melhorar seu desempenho preditivo. A principal diferença entre o SGD e o método clássico de descida do gradiente é a maneira como o gradiente da função de custo é calculado:

- No método tradicional, o gradiente é calculado utilizando todo o conjunto de dados de uma vez a cada iteração, o que pode ser computacionalmente custoso em processamento e armazenamento em memória, especialmente em cenários onde o número de observações e a dimensionalidade dos dados é elevada.
- O SGD, por outro lado, utiliza apenas um subconjunto aleatório dos dados para calcular o gradiente em cada iteração, tornando o algoritmo mais rápido e eficiente para conjuntos de dados grandes e de alta dimensionalidade.

Durante o treinamento, a cada iteração o algoritmo atualiza os pesos do modelo ω na direção oposta ao gradiente da função de custo J em relação aos pesos atuais, um exemplo por vez. Essa atualização dos parâmetros do algoritmo é dada pela equação (52):

$$\omega \coloneqq \omega - \eta * \nabla \mathrm{J}\left(w; x^{(i)}, y^{(i)}\right) \tag{52}$$

onde ω representa o vetor de pesos ou parâmetros do modelo, η é a taxa de aprendizado, que controla o tamanho do passo na direção do gradiente, e $\nabla J\left(w; x^{(i)}, y^{(i)}\right)$ corresponde ao gradiente da função de custo calculado para o exemplo $\left(x^{(i)}, y^{(i)}\right)$, equivalendo a uma única amostra de dados do conjunto de treinamento. Essa abordagem estocástica permite que o SGD escape de mínimos locais e atinja uma velocidade de processamento para cada iteração mais rápida em troca de uma taxa de convergência mais lenta, já que apenas um pequeno subconjunto aleatório dos dados é usado em cada iteração, permitindo que o algoritmo explore melhor a superfície da função de custo devido à maior aleatoriedade do gradiente em detrimento da velocidade de convergência. Além disso, o algoritmo pode incorporar diferentes técnicas de regularização, como L1 (Lasso) e L2 (Ridge), para evitar o *overfitting* e melhorar a generalização do modelo. A regularização atua penalizando grandes valores nos pesos, ajudando a manter um modelo mais simples e robusto à colinearidade dos dados. O algoritmo também pode ser ajustado através da escolha da taxa de aprendizado, que pode ser constante ou decrescente ao longo das iterações, dependendo da estratégia adotada (BOTTOU & BOUSQUET, 2011). Além disso, podem ser utilizadas técnicas como *momentum* ou variantes mais modernas, como os otimizadores Adam e RMSProp para contribuir para a convergência do modelo.

A Figura 30 demonstra um exemplo comparativo entre a taxa de convergência dos três principais métodos de Descida de Gradiente: o algoritmo *Batch Gradient Descent*, *Mini-batch Gradient Descent* e *Stochastic Gradient Descent* para uma superfície de erro arbitrária. Conforme mencionado, a técnica *Batch Gradient Descent* atualiza os pesos do modelo utilizando todos os dados de treinamento para calcular o gradiente da função de custo a cada iteração, fornecendo uma direção de descida do gradiente mais estável, pois o gradiente é calculado com base em toda a população de dados e convergindo de forma mais previsível para o mínimo global ou local, porém esse algoritmo se torna lento para grandes *datasets*, já que é necessário processar todo o conjunto de dados antes de cada atualização e requer uma maior quantidade de memória para armazenar os dados durante o cálculo do gradiente. Já o *Mini-batch Gradient Descent* divide os dados em pequenos lotes de tamanho fixo e atualiza os pesos do modelo com base no gradiente calculado para cada lote. Essa abordagem representa um equilíbrio entre eficiência computacional e estabilidade do gradiente, pois permite que o algoritmo processe vários subconjuntos de

cada vez, sendo mais rápido e menos computacionalmente custoso do que o *Batch Gradient Descent* e mais estável que o SGD, porém, por calcular o gradiente através de lotes de dados reduzidos, torna a direção do gradiente menos precisa do que no método *Batch Gradient Descent*, especialmente com tamanhos de lote pequenos, reduzindo a velocidade de convergência do algoritmo. O SGD, por atualizar os pesos do modelo após calcular o gradiente para um único exemplo de treino por vez, acaba tendo uma velocidade de processamento mais rápida e menor custo computacional para grandes *datasets* do que os outros dois métodos, pois faz atualizações mais frequentes nos pesos e tende a escapar de mínimos locais devido à aleatoriedade do gradiente, porém acaba por se tornar menos estável, gerando flutuações no gradiente e dificultando a convergência, podendo levar mais iterações para convergir ao mínimo global devido a essas flutuações.

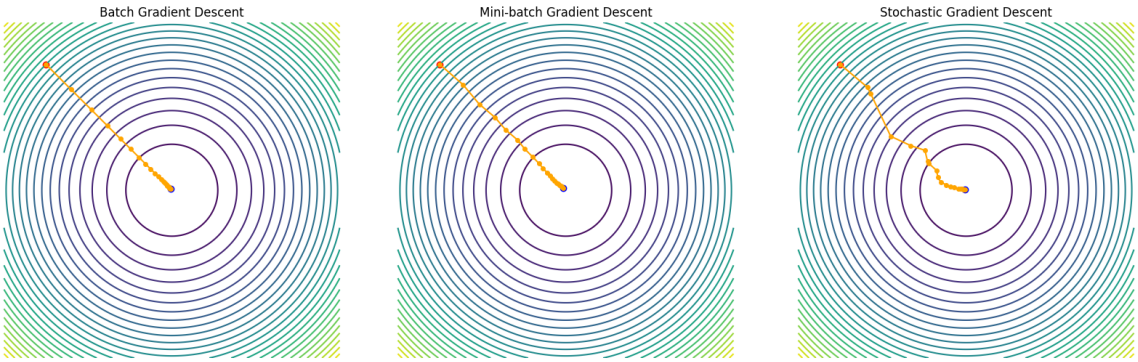

Figura 30. Comparativo entre os métodos de *Batch Gradient Descent*, *Mini-batch Gradient Descent* e *Stochastic Gradient Descent* (De autoria própria)

A partir da Figura 30 é possível observar que o método *Batch Gradient Descent* (à esquerda) é o que apresenta os gradientes mais estáveis, mantendo-se praticamente sempre na mesma direção no sentido do ponto de mínimo da superfície de erro. Já a técnica *Mini-batch Gradient Descent* (ao centro) apresenta algumas oscilações na direção do gradiente ao longo das iterações, fato que se torna mais evidente conforme o tamanho dos lotes diminui, tornando a velocidade de processamento e consumo de memória menores, porém diminuindo a taxa de convergência. Por último, o método *Stochastic Gradient Descent* (à direita) é o que apresenta o comportamento mais instável para o cálculo do gradiente dentre as três abordagens apresentadas, porém é o que exibe menor custo computacional e consumo de memória, facilitando o cálculo do gradiente para

grandes conjuntos de dados, porém, em compensação, é o que leva mais iterações para convergir.

A seguir pode ser observado o pseudocódigo do algoritmo SGD, enfatizando as entradas e saídas do algoritmo, assim como seu uso para tarefas de otimização de funções de custo em algoritmos de Aprendizado de Máquina e Aprendizado Profundo. Ele funciona ajustando os parâmetros do modelo (geralmente pesos e vieses) com base no gradiente da função de custo calculado para um pequeno subconjunto de dados (amostra ou *mini-batch*). O objetivo é minimizar a função de custo iterativamente, aproximando-se do valor ótimo, conforme pode ser observado a seguir:

Input:
X: Matriz de características (n_samples × n_features)
y: Vetor de rótulos (n_samples x 1)
η: Taxa de aprendizado
T: Número máximo de iterações
tol: Critério de parada, cessando o treinamento quando (perda > melhor_perda - tol)

Output:
w: Pesos ajustados

---

1. Inicialize:
- O vetor inicial de coeficientes *w* com valores aleatórios ou zeros

2. Para cada iteração $t = 1$ até T, repita até que um mínimo aproximado seja obtido ou um critério de parada seja atingido:
- Embaralhe aleatoriamente as amostras no conjunto de treinamento
- Para cada exemplo $\left(x^{(i)}, y^{(i)}\right)$ em (X, y) faça:
  - $\omega := \omega - \eta * \nabla J\left(w; x^{(i)}, y^{(i)}\right)$

3. Retorne os pesos ajustados ω

4. Fim.

Neste trabalho, a implementação deste estimador consistiu em utilizar modelos lineares regularizados, treinados utilizando o método SGD, onde a estimativa do gradiente da função de perda é realizada em pequenos subconjuntos de amostras (*mini-batches*), possibilitando que o modelo seja atualizado iterativamente ao longo do

treinamento, aumentando a sua capacidade de generalização para a identificação de *Trip* nas sequências de eventos analisadas.

Uma característica importante dessa implementação foi o uso de uma taxa de aprendizado decrescente, um mecanismo que desempenha um papel crucial no controle do tamanho dos passos realizados pelo modelo em direção ao mínimo da função de perda. Essa abordagem permite que o algoritmo realize uma exploração ampla e percorra longas distâncias do espaço de busca nas iterações iniciais, evitando que ele fique preso em mínimos locais ou regiões subótimas da função de perda e favorecendo a identificação de regiões de maior inclinação na superfície da função de custo. À medida que o treinamento avança, a redução gradual da taxa de aprendizado contribui para o refinamento do processo de ajuste dos parâmetros, possibilitando uma convergência mais estável em direção ao mínimo global ou local mais adequado, conforme se aproxima das iterações finais.

A função de perda utilizada no ajuste do modelo pode ser configurada para diferentes objetivos de aprendizado. No escopo deste trabalho, o estimador foi modelado para ajustar um modelo de Máquina de Vetores de Suporte Linear (*Support Vector Machines* - SVM), uma abordagem bastante utilizada para tarefas de classificação (CORTES & VAPNIK, 1995). Esse algoritmo busca identificar um hiperplano (no formato $w \cdot x + b = 0$) no espaço de entrada, que maximiza a margem de separação entre as classes, ou seja, a distância mínima entre o hiperplano e os pontos de dados mais próximos de cada classe (os vetores de suporte). Ao contrário das versões mais gerais de SVM, que empregam funções *kernel* para mapear os dados para espaços de alta dimensionalidade, o SVM Linear realiza a separação diretamente no espaço original dos dados, simplificando a modelagem e reduzindo a complexidade computacional. Com isso, para a otimização dos parâmetros do modelo, este trabalho utiliza a técnica SGD, sendo particularmente vantajosa em cenários com grandes volumes de dados, pois evita o custo computacional elevado associado à resolução explícita do problema de otimização quadrática tradicionalmente utilizado nas SVMs.

CAPÍTULO 3
DESCRIÇÃO DA FERRAMENTA PROPOSTA

3.1 PRINCIPAIS PROBLEMAS ENFRENTADOS

3.1.1 IDENTIFICAÇÃO DAS CARACTERÍSTICAS MAIS RELEVANTES

O principal desafio relacionado à identificação prévia de *Trips* e previsão de sequência de eventos corresponde primeiramente a identificar quais informações dos eventos são mais pertinentes. Isso é necessário pois os eventos contêm dados úteis para o operador analisar, porém, para um modelo de Inteligência Artificial muitas vezes a alta quantidade de informações e dimensionalidade pode prejudicar o aprendizado, pois o modelo pode priorizar erroneamente características em detrimento de outras que, para um especialista humano, seriam essenciais para realizar uma análise assertiva da sequência de eventos. Dessa forma, os dados de alarmes e chaveamentos registrados pelo SPAL de Angra 2 compreendem às seguintes informações:

- **Referência**: identificador único de cada instrumento, que associa o evento a um componente específico do sistema. A Referência é crucial para mapear a origem do evento e entender o contexto operacional em que ele ocorreu.
- **Data/hora da aquisição (*timestamp*)**: instante de tempo em que o sinal foi adquirido, registrado com precisão de 1 milissegundo. Esse dado temporal é importante para analisar a sequência dos eventos e correlações temporais, permitindo identificar padrões e tendências que possam ser úteis na previsão de eventos futuros.
- **Estado ativado/desativado**: indica se o sinal corresponde à ativação ou desativação de um alarme ou chaveamento. Esse estado binário reflete mudanças no status dos componentes do sistema, sendo uma variável crítica na identificação de comportamentos que precedem um *Trip*.
- **Validade da informação**: indica se o evento registrado é válido ou inválido, geralmente associado a uma medição errônea do sinal de origem, problemas físicos durante a medição, sinais espúrios ou alterações erráticas em um sinal em um curto espaço de tempo.

Essas características compõem a base dos eventos registrados, e um exemplo

simples de como esses dados são estruturados pode ser visto a seguir:

```
B3708  -2021/09/19 00:16:24 258-0-1-0-700-0
```

Neste exemplo, o identificador *B3708* refere-se a um instrumento específico, enquanto o *timestamp* (2021/09/19 00:16:24 258) marca o momento de aquisição do evento, com precisão de 1 milissegundo. Os valores subsequentes indicam o estado do componente no instante em questão, grupo de alarme e a validade do evento.

Para determinar quais dessas características (ou combinações delas) são mais relevantes para o modelo, foram utilizadas algumas abordagens comumente aplicadas em aprendizado de máquina, permitindo uma escolha de características mais assertiva e reduzindo a complexidade dos modelos treinados:

1. **Análise de correlação**: foi avaliada a correlação entre as variáveis para identificar quais delas têm maior influência sobre o resultado desejado, sendo nesta primeira abordagem, a identificação prévia de *Trips*. Isso ajudou a reduzir o número de variáveis, eliminando aquelas que eram redundantes ou pouco correlacionadas com a ocorrência de *Trips*. Para isso, as variáveis foram tokenizadas e agrupadas em sequências de até 50 eventos, sendo o evento topo o alarme de desarme iminente do reator. Após este agrupamento, foi observado quais características são mais correlacionadas à ocorrência de *Trip*.
2. **Técnicas de engenharia de características**: analisou-se a diferença entre o tempo de ativação e desativação de determinadas variáveis e o intervalo entre a ocorrência de eventos de variáveis críticas até a ocorrência do *Trip* de fato, que permitiu capturar informações adicionais sem aumentar a complexidade do modelo. Outra técnica foi a criação de variáveis agregadas, como a contagem de eventos de variáveis específicas dentro de um intervalo de tempo, que pode ser um forte indicativo da iminência de um *Trip*.
3. **Validação com especialistas**: a experiência dos operadores também foi essencial. Validar a seleção de características com especialistas na operação de Angra 2 ajudou a garantir que o modelo não estivesse negligenciando variáveis que, mesmo com pouca importância estatística, pudessem ter um papel crítico na análise humana dos eventos.

Dessa forma, foi possível identificar um conjunto ótimo de características que não

apenas preserva a integridade da análise operacional, mas também melhora o desempenho dos modelos computacionais, garantindo que o aprendizado seja eficiente e focado nas variáveis mais relevantes para a previsão de *Trips* e sequência de eventos, sendo essas características a “referência” e o “estado de ativação da variável”.

3.1.2 ARQUITETURA DO BANCO DE DADOS

A definição da arquitetura do Banco de Dados foi um passo essencial para garantir que os dados dos eventos registrados pelo SPAL, SLOG e SOE fossem organizados de maneira eficiente, permitindo o armazenamento e a recuperação rápida de informações críticas para a análise de sequência de eventos ou para outras tarefas como, por exemplo, para a identificação de causa-raiz de desarmes do reator. Um dos principais desafios enfrentados foi ir além da simples visualização da Referência e Estado das variáveis (ativado/desativado), que, por si só, não fornecem uma visão ampla do que realmente está acontecendo no sistema. Essa limitação se torna ainda mais evidente quando os operadores precisam investigar rapidamente a sequência de eventos que culminou em um incidente ou transiente operacional.

Para superar essa limitação, o Banco de Dados relacional (RDBMS) (do inglês, *Relational Database Management System*) proposto foi estruturado de forma a incluir informações adicionais que facilitam a análise e interpretação dos eventos. Esse formato de Banco de Dados corresponde a um sistema que permite a criação, atualização e gestão do RDBMS, que utiliza tabelas para representar os dados, de forma que cada linha da tabela corresponde a um registro, com um identificador exclusivo, e cada coluna representa um atributo de dados. O objetivo desta abordagem foi garantir que, além do identificador e do estado da variável, o Banco de Dados também armazenasse dados que proporcionassem um contexto mais abrangente sobre os eventos em questão, possibilitando uma análise mais rápida e precisa por parte dos operadores.

Para permitir uma análise mais profunda e reduzir o tempo de resposta, foi necessário mapear e armazenar informações adicionais associadas a cada evento. Os principais dados mapeados e associados a cada variável foram:

- **Descrição da variável**: Fornece um texto explicativo que descreve o que a

variável representa no contexto da usina, ajudando o operador a entender rapidamente o papel dessa variável no contexto do sistema.

- **Sistema da usina ao qual a variável está associada**: Identifica a área ou o subsistema específico da usina onde o evento ocorreu, permitindo que a análise seja direcionada para o conjunto correto de sistemas e componentes.
- **Texto associado à mudança de estado**: Explicita o significado da transição de estado da variável, por exemplo, de 0 (desativado/não atuado) para 1 (ativado/atuado) ou vice-versa. Essa informação é crucial para identificar a natureza da alteração e seu impacto no sistema.

A inclusão desses dados adicionais possibilita uma compreensão mais clara de cada evento, oferecendo ao operador e aos modelos de Inteligência Artificial uma visão mais detalhada do que está acontecendo em nível operacional.

Dentro do contexto das informações dispostas no Banco de Dados do sistema, é importante frisar o quão importante é a definição do nível de detalhamento das informações coletadas e armazenadas. Como foi mencionado, cada coluna de uma tabela representa um atributo ou dimensão de dados, de forma que quanto mais colunas uma tabela apresentar (mais dimensões), diz-se que maior é o grau de granularidade dos dados, ou seja, maior é a quantidade de detalhes e informações específicas que ela contém. A granularidade de dados é importante pois permite uma análise mais precisa e maior acesso à informação, de forma que o especialista teria acesso a uma grande quantidade de dados para o auxílio à tomada de decisão. Contudo, dados muito granulares podem dificultar a concentração e a identificação de padrões e, quanto maior o nível de detalhes, maior o volume de dados armazenados e o tempo de resposta para consultar as informações, tornando a tarefa de identificar as informações mais pertinentes a serem armazenadas dentro do escopo de um sistema computacional uma tarefa de *design* e otimização.

A comunicação com o Banco de Dados é realizada por meio da interface ODBC (do inglês, *Open Database Connectivity*), uma API (do inglês, *Application Programming Interface*) amplamente utilizada para permitir a interação entre o banco de dados e outros aplicativos de *software*, como Python, Excel e Power BI. O ODBC padroniza a conexão e comunicação entre diferentes sistemas, oferecendo uma camada de abstração que

simplifica a interface entre o Banco de Dados e os aplicativos externos, de forma que esse padrão não apenas facilita a integração entre ferramentas de análise de dados, mas também permite que diferentes tipos de aplicações acessem o Banco de Dados e realizem alterações nele, como inserção, alteração e exclusão de registros e regras. Por meio dessa interface, o modelo de Banco de Dados proposto pode ser utilizado para a realização de consultas, geração de relatórios e visualizações de dados dinâmicas, permitindo uma interação com ferramentas de análise de dados, como Python e Power BI, que podem ser empregados para a visualização interativa de eventos e padrões operacionais, assim como aplicação de métodos de Inteligência Artificial para análise preditiva e de classificação.

A utilização da interface ODBC oferece ainda outras vantagens significativas, como a padronização da linguagem SQL, a uniformidade nos tipos de dados e a interface de conexão, permitindo maior flexibilidade e interoperabilidade entre diferentes sistemas, além de também permitir uma gestão de erros, por meio de mensagens de erro padronizadas que facilitam a identificação e solução de problemas durante o processo de consulta e atualização dos dados.

Esse modelo de RDBMS visa não apenas otimizar a gestão de dados operacionais de sequência de eventos, mas também proporcionar uma fundamentação sólida para futuras implementações de modelos preditivos e sistemas de monitoramento mais avançados, essenciais para a segurança operacional e a eficiência da planta. Para isso, o banco de dados desenvolvido é composto pelas seguintes tabelas e entidades principais:

- **Tabela de Sensores**: contém as informações essenciais do conjunto de instrumentos mapeados e utilizados na usina, como Referência (chave primária), Descrição, Sistema de Aquisição e Sistema da Usina Associado. O design da Tabela de Sensores pode ser visualizado na Tabela 6:

Tabela 6. Campos da Tabela de Sensores

Campo	Tipo de Dados	Comprimento Máximo	Permite Nulos
Referencia	VARCHAR	7	Não
Sistema_Aquisicao	VARCHAR	3	Não
Descricao	VARCHAR	40	Não
KKS_Id_Code	VARCHAR	13	Não

- **Tabela de Informações de Variáveis Binárias**: contém o texto 0 → 1 e 1 → 0 de cada variável binária, indicando o texto relacionado à mudança de estado, assim como o tipo da variável (alarme ou chaveamento). Se relaciona com a "Tabela de Sensores" através do campo "Referência", que também corresponde à chave primária na tabela de Binárias, criando uma relação 1 ⇔ 1 entre elas. O design da Tabela de Variáveis Binárias pode ser visualizado na Tabela 7:

Tabela 7. Campos da Tabela de Variáveis Binárias

Campo	Tipo de Dados	Comprimento Máximo	Permite Nulos
Referencia	VARCHAR	7	Não
Texto01	VARCHAR	8	Sim
Texto10	VARCHAR	8	Sim
Tipo_Alarme	VARCHAR	1	Não

- **Tabela de Eventos**: Armazena as informações dos eventos, como o identificador da variável, o *timestamp* da aquisição e o estado (ativado/desativado). Nesta tabela a chave primária é a junção dos atributos "Referência" e "Horário de Aquisição" pois não é permitido que um instrumento possua mais de um registro associado ao mesmo instante de tempo. O design da Tabela de Eventos pode ser visualizado na Tabela 8:

Tabela 8. Campos da Tabela de Eventos

Campo	Tipo de Dados	Comprimento Máximo	Permite Nulos
Referencia	VARCHAR	7	Não
Horario_Aquisicao	VARCHAR	23	Não
Estado	VARCHAR	1	Não

A arquitetura proposta foi desenhada com o foco em acelerar a seleção e análise dos eventos. Como o tempo é um fator crítico na operação de uma usina nuclear, a capacidade de acessar rapidamente as informações relacionadas a um desarme do reator ou compreender a sequência de eventos que levou a ele pode fazer toda a diferença para a segurança e a eficiência operacional.

Na Figura 31 é possível observar o Modelo Entidade Relacionamento (MER) desenvolvido, contendo os campos de cada tabela, assim como o relacionamento entre elas. Foram criados disparadores (*triggers*), para cada tabela para garantir a integridade do Banco de Dados, de forma que a inclusão de novos sensores deve ser realizada primeiramente na tabela *Sensor* para então poder ser realizada nas demais tabelas. De maneira contrária, a exclusão de registros deve ser feita primeiramente na tabela *Eventos* e/ou *Binarias* para então poder ser realizada na tabela *Sensor*.

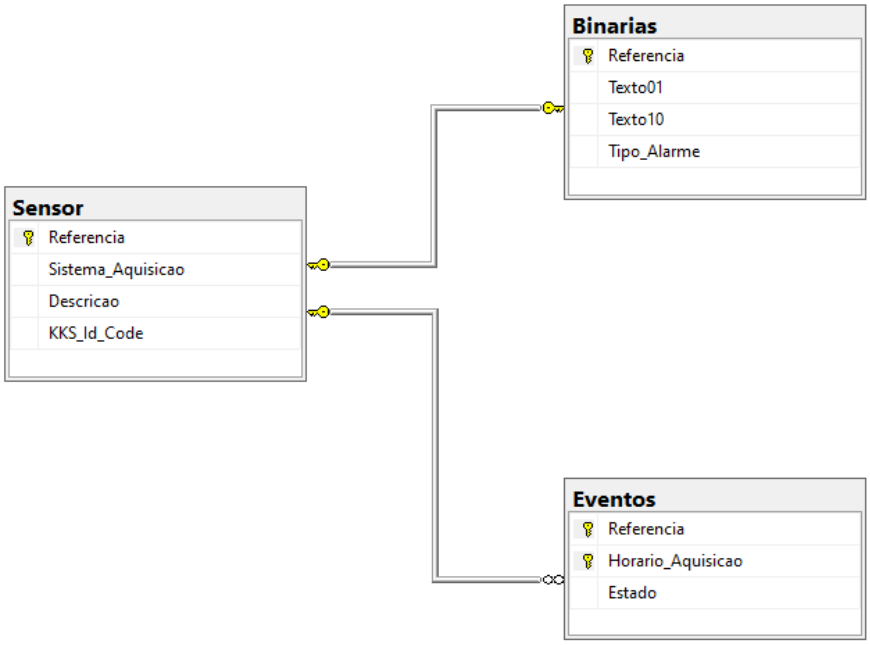

Figura 31. Modelo entidade relacionamento do Banco de Dados do sistema

Dessa forma, o Banco de Dados foi projetado não apenas para armazenar eventos de forma estruturada, mas também para permitir consultas otimizadas que facilitem a identificação de padrões, correlações e causas subjacentes. Foram adotadas algumas estratégias fundamentais para garantir a agilidade na recuperação dos dados, como:

- Ferramentas de indexação para a tabela de Eventos, ou seja, estruturas adicionais que permitem acelerar o processo de busca em tabelas ao organizar os dados de forma a facilitar o acesso. No caso da tabela de Eventos, índices foram aplicados em colunas frequentemente utilizadas como filtros ou em junções, reduzindo significativamente o tempo de execução das consultas. Isso é particularmente importante quando se lida com grandes volumes de dados,

pois evita a necessidade de percorrer toda a tabela.

- Cache de consultas frequentes, que armazena os resultados de consultas frequentemente realizadas na memória, evitando que elas sejam processadas repetidamente, como na busca por variáveis ou intervalos de tempo específicos. Em um RDBMS, isso permite respostas muito mais rápidas para solicitações recorrentes, já que os dados são recuperados diretamente do cache em vez de serem buscados e processados novamente no disco.
- *Stored procedures*, que correspondem a blocos de código SQL pré-compilados e armazenados no próprio Banco de Dados. Elas são executadas diretamente no servidor, o que reduz a latência causada pela transmissão de comandos entre o cliente e o servidor. Além de garantir mais eficiência, os procedimentos armazenados permitem encapsular operações complexas e padronizar processos, facilitando a manutenção e a segurança do sistema.

Vale ressaltar que os dados de sequência de eventos foram armazenados tanto em formato de arquivo de texto, conforme gerado originalmente pelo SPAL e SOE, quanto no Banco de Dados propriamente dito, a fim de otimizar a busca por variáveis ou intervalos de tempo específicos, assim como manter os registros originais.

3.2 DESCRIÇÃO DA FERRAMENTA

A ferramenta proposta neste trabalho consiste em um *pipeline* responsável pelo armazenamento, tratamento e aplicação dos dados de sequência de eventos provenientes do SPAL da Usina Nuclear Angra 2, de maneira que cada etapa desse *pipeline* realiza uma tarefa específica que, combinadas, serão capazes de treinar, validar e testar diferentes modelos de Inteligência Artificial para as tarefas de identificação prévia de *Trip* e previsão do(s) próximo(s) evento(s) de uma sequência de alarmes e chaveamentos da usina.

O conjunto de modelos de Aprendizado Profundo abrangidos pela ferramenta baseia-se em Redes Neurais LSTM e Transformers, correspondendo a algoritmos altamente eficazes na modelagem de sequências e dependências de longo prazo em dados de séries temporais ou documentos de texto. Além disso, para fins de comparação, foram utilizados 5 modelos de Aprendizado de Máquina amplamente mencionados e utilizados na literatura para tarefas de classificação e/ou regressão: Ridge, Floresta Aleatória, SGD

(otimizando uma SVM), Naive Bayes e kNN. A solução foi projetada para auxiliar na identificação prévia de *Trips* e para prever os eventos seguintes de uma sequência de eventos, com alta exatidão, contribuindo para a segurança e operação eficiente da usina.

Além disso, a ferramenta foi construída de maneira a utilizar um RDBMS, permitindo que as consultas e o armazenamento dos eventos sejam realizados de forma estruturada e eficiente, o que facilita a análise posterior dos eventos críticos. Para isso, foi criado um modelo de Banco de Dados SQL (do inglês, *Structured Query Language*) que incorpora informações detalhadas sobre cada evento, como Referência do instrumento, *timestamp* (data/hora de aquisição), e o estado (ativado/desativado) da variável.

A proposta visa, ainda, reduzir o tempo de treinamento e melhorar o desempenho dos modelos neurais ao utilizar GPUs (do inglês, *Graphics Processing Unit*) para o processamento, aproveitando a computação paralela para acelerar o ajuste dos parâmetros e os cálculos matriciais envolvidos no aprendizado dos modelos, reduzindo o tempo de inferência e possibilitando maior assertividade na tomada de decisão, o que é essencial para situações de alta criticidade como as encontradas em sistemas nucleares.

Esse *pipeline* de dados e modelos tem como objetivo melhorar a capacidade dos operadores da usina em prever falhas e em tomar decisões rápidas e informadas, contribuindo assim para o aumento da segurança operacional da usina nuclear.

3.3. MATERIAIS E MÉTODOS

O tempo de treinamento é um fator crítico em arquiteturas profundas de redes neurais, principalmente atualmente com o advento das redes Transformers e com a utilização de Grandes Modelos de Linguagem (do inglês, *Large Language Models* - LLM) especialmente devido à complexidade e ao volume de cálculos envolvidos no ajuste dos parâmetros treináveis dessa arquitetura de rede. Historicamente, o longo tempo de treinamento foi uma das principais barreiras para o uso mais amplo de redes neurais profundas, pois exigia recursos computacionais intensivos que nem sempre estavam disponíveis. Com a evolução da tecnologia e capacidade computacional, uma das soluções que transformou esse cenário foi o uso de computação paralela, particularmente com o auxílio de Unidades de Processamento Gráfico (GPUs) e a utilização de núcleos

de processamento CUDA (do inglês, *Compute Unified Device Architecture*). As GPUs são altamente eficientes no processamento de tarefas paralelas, tornando-as ideais para acelerar o treinamento de modelos de aprendizagem profunda.

As GPUs foram originalmente desenvolvidas para o processamento de gráficos em jogos e aplicativos visuais, mas, devido à sua arquitetura paralela, elas são extremamente eficazes para tarefas de Aprendizado de Máquina que exigem a execução de operações matemáticas complexas em grandes volumes de dados simultaneamente. Em redes neurais profundas, o treinamento envolve cálculos de derivadas, multiplicações de matrizes e atualizações de pesos neuronais que podem ser executados de maneira paralela e distribuída, reduzindo drasticamente o tempo de processamento comparado ao uso de CPUs tradicionais (ABADI *et al.*, 2015).

Neste trabalho, foram utilizados os *frameworks TensorFlow* (ABADI *et al.*, 2015) e *Huggingface's Transformers* (WOLF *et al*., 2019) para o desenvolvimento e treinamento dos modelos de aprendizagem profunda. O *TensorFlow*, desenvolvido pela equipe de aprendizado de máquina do Google, é uma plataforma de código aberto que oferece uma infraestrutura dedicada à criação e execução de modelos de redes neurais. Já o *Huggingface* é uma plataforma de código aberto composta por arquiteturas Transformers pré-treinadas e projetadas sob uma API unificada, facilitando o desenvolvimento de modelos através de linguagens de alto nível, como Python.

Para acelerar o treinamento dos modelos, o *TensorFlow* foi configurado para utilizar uma GPU, especificamente uma placa de vídeo GeForce RTX 4050 de 6 GB de memória GDDR6. Esta GPU faz uso da arquitetura Ada Lovelace, que é a oitava geração de processadores NVIDIA compatíveis com a tecnologia CUDA. A arquitetura CUDA permite que as GPUs sejam utilizadas não apenas para renderização gráfica, mas também para o processamento de tarefas gerais, incluindo o treinamento de redes neurais profundas, aproveitando seus múltiplos núcleos para executar cálculos paralelos. No caso da RTX 4050 utilizada, ela contém 2.560 núcleos CUDA operando em paralelo.

A arquitetura Ada Lovelace, desenvolvida pela NVIDIA, representa um avanço significativo na evolução das GPUs, consolidando-se como um marco tecnológico ao introduzir melhorias substanciais em desempenho computacional, eficiência energética e

suporte aprimorado a recursos avançados de Inteligência Artificial. Entre as inovações mais notáveis, destacam-se as unidades de cálculo otimizadas para Aprendizado Profundo, que incluem a 4ª geração dos núcleos de tensor (do inglês, *Tensor Cores*). Esses núcleos especializados foram projetados especificamente para operações envolvendo matrizes e tensores, estruturas fundamentais no treinamento e inferência de redes neurais artificiais. A capacidade de realizar cálculos matriciais de forma altamente paralelizada e com precisão mista (*mixed-precision*) permite ganhos expressivos na velocidade de processamento, sem comprometer a qualidade dos resultados (MICIKEVICIUS *et al.*, 2017). Essa otimização é especialmente relevante em tarefas como multiplicação de matrizes densas, convoluções e operações típicas de modelos de Aprendizado Profundo, onde o volume de dados e a complexidade dos cálculos são elevados.

O uso de GPUs no treinamento dos modelos de redes neurais resultou em uma redução significativa de tempo de processamento necessário para ajustar os parâmetros dos modelos. Essa aceleração proporcionou a capacidade de explorar, de forma mais eficiente, diferentes arquiteturas e hiperparâmetros, facilitando a identificação de configurações que oferecem o melhor desempenho para cada tarefa específica. As GPUs se destacam nesse contexto devido à sua arquitetura altamente paralelizada, que é ideal para lidar com as operações intensivas e complexas envolvidas no treinamento de redes neurais, de forma que tais operações incluem o cálculo de gradientes e o mecanismo de retropropagação, fundamentais no ajuste iterativo dos pesos e vieses durante o aprendizado supervisionado. Enquanto os processadores convencionais (CPUs) apresentam um número limitado de núcleos otimizados para tarefas sequenciais, as GPUs contêm milhares de núcleos capazes de realizar operações em paralelo com alta eficiência, permitindo ganhos significativos de desempenho em aplicações computacionalmente intensivas.

Esse impacto é especialmente relevante no treinamento de redes neurais profundas, em que o aumento no número de camadas resulta em um crescimento significativo na quantidade de parâmetros a serem ajustados e nas operações envolvidas. Em tais redes, a cada camada adicional, operações como multiplicação de matrizes, cálculo de derivadas parciais e atualização dos pesos se tornam ainda mais custosas do ponto de vista computacional (SINGH *et al.*, 2015).

CAPÍTULO 4
ESTUDO DE CASO, RESULTADOS E DISCUSSÕES

4.1. APRESENTAÇÃO DO ESTUDO DE CASO

4.1.1. DESCRIÇÃO DO CONJUNTO DE DADOS

Nesta dissertação, como estudo de caso, foi utilizado um conjunto de dados aquirido e armazenado pelo SPAL, SLOG e pelo Servidor de Eventos da usina nuclear Angra 2 para treinar, validar e testar os modelos de Aprendizado Profundo e os 5 métodos citados de Aprendizado de Máquina (Ridge, Floresta Aleatória, SGD+SVM, Naive Bayes e k-Nearest Neighbors) que compõem o conjunto de modelos da ferramenta proposta. No total, o conjunto de dados é formado por eventos gravados a partir de agosto de 2014 até setembro de 2021, totalizando aproximadamente 15,8 milhões de registros de variáveis binárias, distribuídas entre eventos que requerem ou não sinalização de alarme para o operador. Esse conjunto de dados, antes de passar pela etapa de pré-processamento e tratamento, é composto por dois arquivos de registros de aquisição da usina que são constituídos por eventos de alarmes e chaveamentos durante a operação e períodos de parada da usina, assim como eventos classificados como inválidos pelo sistema de aquisição e, por se tratar inicialmente de dois arquivos distintos, ao uni-los podem-se constatar que existem sequências de eventos duplicadas, comuns aos dois arquivos.

A Tabela 9 mostra os possíveis estados de operação que uma sequência de eventos pode representar dentro do escopo deste trabalho para a tarefa de classificação: “Trip” e “Não Trip”. Como se trata de um problema de classificação binária (0 ou 1), para cada classe foi atribuído um rótulo, que permitiu a cada modelo relacionar uma dada sequência de eventos a uma situação de Trip $\Leftarrow 1$ ou Não Trip $\Leftarrow 0$.

Tabela 9. Estados de operação a partir da sequência de eventos

ID	Estado	Descrição
0	NÃO TRIP	Representa uma sequência de eventos que não ocasionou Trip
1	TRIP	Representa uma sequência de eventos que ocasionou o desligamento da turbina e do reator

Dentre todas as variáveis envolvidas nos eventos desse conjunto de dados, foram classificadas como sinais de *Trip* todas as que possuem como descrição o sinal de Desligamento do Reator (RESA) (do inglês, *Reactor Shutdown Alarm*) ou Desligamento da Turbina (TUSA) (do inglês, *Turbine Shutdown Alarm*), correspondendo no total a 9 alarmes distintos analisados e compreendendo tanto desligamentos manuais quanto automáticos da usina.

Dessa forma, após importar os dados do SPAL e SLOG, foi possível observar que, dos 15,8 milhões de registros, aproximadamente 630 (~0,004%) correspondem a sinais de desligamento do reator. Vale ressaltar que os sinais de RESA aquiridos não necessariamente indicam que o reator de fato desarmou, mas sim que o respectivo alarme ou chaveamento teve seu estado alterado e/ou que variáveis redundantes também foram alternadas, onde a aquisição de um sinal $0 \Rightarrow 1$ seguido de $1 \Rightarrow 0$ indica que a situação de *Trip* foi prevenida. Além disso, esse conjunto de sequências foi posteriormente ampliado para 3.819 sequências de Trip através da alteração da ordem de determinados eventos chave, garantindo uma maior variabilidade de exemplos para os modelos computacionais.

4.1.2. MODELAGEM DO CONJUNTO DE DADOS

No processo de modelagem do conjunto de dados para as tarefas de identificação de *Trip*, o *dataset* original foi particionado em sequências contínuas de no máximo 50 eventos e de forma que sequências distintas não podiam apresentar eventos em comum. A segmentação dos dados em blocos de tamanho limitado e pré-definido visou preservar a estrutura sequencial dos eventos e para garantir que o modelo consiga identificar previamente um *Trip* sem necessitar de uma quantidade excessiva de observações, facilitando dessa forma o treinamento dos modelos para capturarem dependências ou relações entre eventos prévios ao desarme do reator. Para a previsão de eventos, foram realizados experimentos variando o comprimento máximo das sequências, onde foram treinados modelos mais simples, capazes de receber até 50 eventos, até modelos mais complexos, capazes de receber e processar sequências de até 1000 eventos. Essa variação do comprimento máximo da sequência foi realizada com o objetivo de se constatar a diferença entre a capacidade de retenção de informação à longo prazo das Redes LSTM e Transformer, além do custo computacional envolvido e tempo de inferência.

Em ambos os casos, identificação de *Trip* e previsão de eventos, todas as sequências com uma quantidade de eventos menor do que o comprimento máximo definido para o modelo deveram ser devidamente tratadas durante a etapa de pré-processamento dos dados, de forma a manter a uniformidade de comprimento das sequências. Isso é realizado ao preencher o restante dos termos da sequência com zeros até alcançar o comprimento estabelecido, processo conhecido como *padding*. Seguindo o mesmo raciocínio, quando uma sequência apresentou um número de eventos maior do que o tamanho máximo estabelecido, a sequência foi truncada até o valor máximo de eventos permitido.

Uma vez criadas as sequências, para os experimentos de previsão de *Trip* foi aplicada uma remoção do evento topo de cada sequência, i.e., o evento de *Trip* propriamente dito. Esse procedimento foi implementado para prevenir que os modelos aprendam a correlação entre os eventos de RESA e a ocorrência do *Trip* de fato, priorizando o aprendizado através dos eventos que precederam o desarme do reator. A remoção do topo da sequência é uma estratégia importante para reduzir o viés no modelo a eventos específicos, focando nos padrões dinâmicos ao longo da sequência, ao invés de focar em um determinado evento de *Trip* específico.

A fim de manter somente as informações mais importantes provenientes do SPAL para cada evento, conforme discutido na seção 3.1.1, as sequências de eventos foram modeladas considerando o identificador de cada variável e seu respectivo estado, unidos pelo caractere *underscore* (_). A cadeia de texto representando uma sequência é formada, portanto, por eventos seguindo o formato a seguir:

B1_0	B2_0	B3_1	B4_0	...	B50_1

A *string* (cadeia de caracteres) correspondente a esse exemplo seria: “B1_0 B2_0 B3_1 B4_0 ... B50_1”. Com isso, é possível observar que cada componente dessa sequência é separado por um espaço em branco. Essa modelagem é importante pois, para os modelos de aprendizado abrangidos neste trabalho, é utilizado um delimitador para especificar onde cada palavra de uma sentença deve ser separada em *tokens* constituintes, de forma que o modelo possa ser capaz de aprender características importantes, como a ordem de cada evento, seu significado, contexto e posição dentro da sequência.

Dessa forma, após o pré-processamento e separação das sequências de eventos, foram geradas no total 8.706 sequências, sendo 3.819 de *Trip* (44%) e 4.887 de Não-*Trip* (56%), conforme pode ser observado na Figura 32. Por mais que no conjunto de dados original existam muito mais sequências de Não-*Trip*, foi optado por utilizar apenas 4.887 para priorizar o balanceamento entre as classes, evitando que se tenham muito mais observações de uma em detrimento da outra, mas ainda mantendo uma diversidade de informações disponíveis para o modelo. Esse balanceamento é crucial em tarefas de aprendizado supervisionado, pois evita que o modelo se torne tendencioso para a classe majoritária (Não-*Trip*), garantindo que o modelo seja igualmente capaz de identificar eventos de ambas as classes.

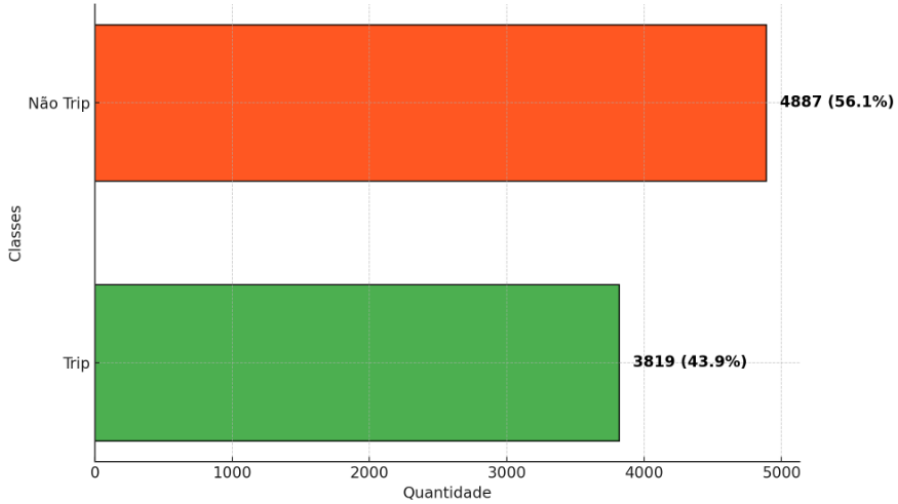

Figura 32. Distribuição dos dados utilizados para treinamento e validação dos modelos para identificação de Trip (De autoria própria)

Para a tarefa de previsão do próximo evento de uma sequência, foi utilizada a mesma base de dados que havia sido empregada em experimentos anteriores. No entanto, houve uma modificação importante: em vez de utilizar todos os dados disponíveis, optou-se por trabalhar com um subconjunto de 10.000 eventos extraídos dos dados originais. A partir desse conjunto, foram criadas janelas de eventos utilizando a técnica de n-gramas, ou seja, sequências contínuas de *n* elementos extraídos de um conjunto maior, geralmente palavras ou caracteres de um texto, de forma que esses elementos são considerados na mesma ordem em que aparecem no texto original (DELLER & HANSEN, 2005).

No presente trabalho, os n-gramas são usados para gerar sequências de palavras a partir do *corpus* de entrada, estruturando os dados de forma a permitir o treinamento de

modelos de aprendizado supervisionado. Cada sequência gerada é composta por palavras consecutivas, onde as primeiras $n-1$ palavras são usadas como entrada, e a *n*-ésima palavra é usada como rótulo (saída esperada). Essa abordagem permite que o modelo aprenda padrões sequenciais no texto, sendo particularmente útil para tarefas de predição de sequência de textos e modelagem de linguagem, pois permite estruturar os dados de forma a capturar a dependência entre cada *token*. Como exemplificado na Figura 33, o método de n-gramas funciona ao aplicar uma janela de tamanho fixo que percorre a sequência original, dividindo-a em subsequências menores que possuem o mesmo tamanho da janela utilizada.

Sequência de Eventos	A	B	C	D	E	F	G	H	I	J	K
1ª Janela	A	B	C	D	E	F	G				
2ª Janela	A	B	C	D	E	F	G	H			
3ª Janela	A	B	C	D	E	F	G	H	I		
4ª Janela	A	B	C	D	E	F	G	H	I	J	
5ª Janela	A	B	C	D	E	F	G	H	I	J	K

Figura 33. Representação de sequências de n-gramas (De autoria própria)

Utilizando a técnica de n-gramas, foram criadas sequências com tamanhos variando de 20 a 50 eventos, o que resultou em um total de 308.419 sequências (~61,7 MB de memória); 20 a 100 eventos, resultando em 808.819 sequências (~320,73 MB de memória); 20 a 500 eventos, totalizando 4.569.019 sequências (~9,14 GB de memória); 20 a 1.000 eventos, totalizando 8.828.019 sequências (~32,90 GB de memória). O que se pode verificar empiricamente é que, ao utilizar o método dos n-gramas, a maior parte das sequências geradas corresponde a matrizes esparsas, ou seja, praticamente composta de zeros, fenômeno observado principalmente quando o tamanho fixado das sequências se torna cada vez maior, o que leva a uma explosão combinatória no número de n-gramas gerados. Como resultado, o consumo de memória para armazenar os vetores cresce exponencialmente, tornando-se um problema crítico, especialmente quando a abordagem envolve sequências de n-gramas a partir de 500 eventos de maneira que, nessas circunstâncias, o uso de memória pode ultrapassar facilmente a capacidade de grande parte dos computadores *desktop* domésticos atuais.

Para contornar esse problema e viabilizar o treinamento de redes neurais com o volume de dados gerado pelos n-gramas, foi adotada a técnica de processamento em lotes (*batch processing*). Essa abordagem consiste em dividir o conjunto de dados em porções

menores, chamadas de *batches*, que são processadas de forma incremental durante o treinamento do modelo. Essa técnica apresenta como principal vantagem a redução da quantidade de memória necessária para manter a informação, já que apenas uma fração dos dados é carregada na memória por vez, permitindo que grandes conjuntos de dados sejam processados em máquinas com recursos computacionais limitados. Em uma linguagem de programação como Python, esse processo é realizado ao preparar uma função geradora com processamento em lotes e embaralhamento (*shuffle*), que recebe o conjunto de dados original e gera, conforme demandado pelo modelo, as sequências de eventos de cada lote. Dessa forma, assim que um lote é processado pelo modelo, ele é excluído da memória e é tratado pelo coletor de lixo (do inglês, *garbage collector*) do interpretador do Python, permitindo que o próximo lote seja gerado e processado.

Tecnicamente, durante o treinamento em lotes, o conjunto de dados de N amostras é dividido em partes de tamanho B (*batch size*), onde B geralmente é definido com base nos recursos computacionais disponíveis. Dessa forma, o modelo realiza as seguintes etapas para cada lote:

1. Executa uma passagem para frente (*forward pass*) com B exemplos gerados, calculando as previsões para estes dados.
2. Calcula o erro (através da função de perda) para os B exemplos.
3. Utiliza o erro para calcular os gradientes.
4. Atualiza os pesos do modelo utilizando os gradientes calculados.
5. O próximo lote é processado, repetindo o ciclo.

Essa abordagem foi fundamental para permitir que sequências muito longas pudessem ser processadas, permitindo o treinamento de redes neurais de forma mais eficiente, mesmo com um *dataset* de grandes proporções. Além disso, o processamento em lotes possibilitou explorar combinações maiores de n-gramas sem que o consumo de memória fosse um fator limitante, garantindo a escalabilidade do método para problemas reais.

Na prática, cada subsequência extraída pelos n-gramas contém uma sequência contínua de eventos, onde o último evento da janela pode ser utilizado como o valor a ser

previsto (*target*), enquanto os eventos anteriores atuam como as variáveis preditoras (*features*), o que possibilita que o modelo de aprendizado de máquina ou rede neural capture os padrões de recorrência e dependência existentes nos dados. A escolha do tamanho máximo da sequência é crucial, uma vez que sequências muito pequenas podem não capturar toda a complexidade da série de eventos e acabam diminuindo a velocidade de convergência do modelo, enquanto sequências muito grandes podem introduzir ruído ou tornar o modelo excessivamente complexo e apresentar um alto custo computacional.

A utilização de n-gramas facilita, portanto, a organização dos dados em um formato adequado para análise, permitindo que padrões contextuais sejam detectados e utilizados para melhorar a exatidão das previsões. Essa técnica é particularmente útil em cenários onde o objetivo é prever o próximo evento de uma série, pois ajuda a identificar dependências ou correlações que podem ser sutis, mas que são fundamentais para a modelagem preditiva (DELLER & HANSEN, 2005).

A Figura 34 mostra como a técnica de n-gramas foi aplicada neste trabalho. Como exemplo nesta figura, foi utilizado um valor de *n* igual a 3, representado uma trigrama. Desta forma, ao se utilizar uma janela de 3 eventos, cada modelo seria responsável por prever o 4º evento de uma dada sequência.

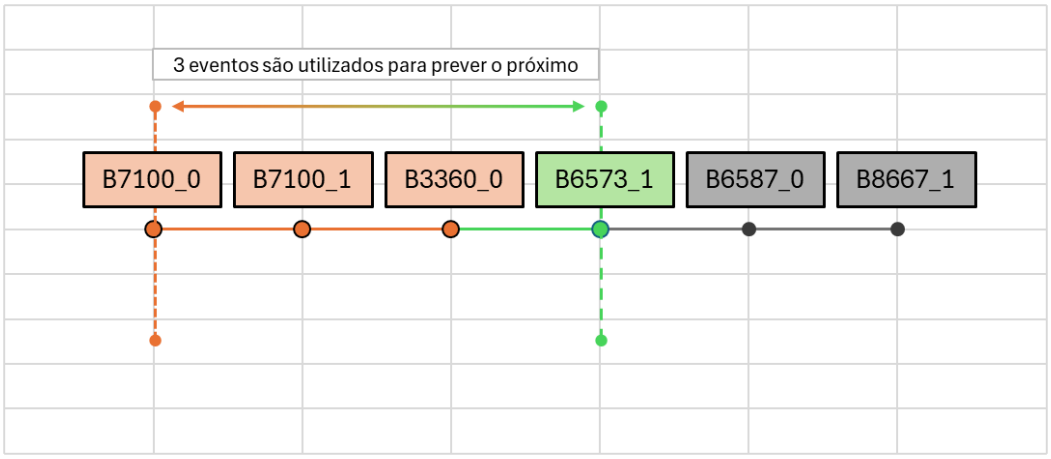

Figura 34. Exemplo de uso da técnica de n-gramas com $n = 3$ (De autoria própria)

A técnica de n-gramas é amplamente utilizada para modelar conjuntos de dados que envolvem documentos de texto. Sendo assim, nesse trabalho, para os modelos de Transformer, LSTM e Aprendizado de Máquina desenvolvidos é utilizada a técnica de n-gramas, para modelar tanto o conjunto de dados de sequência de eventos para previsão de

Trip quanto para previsão do próximo evento, com o objetivo de fornecer aos modelos uma maior variedade de observações e sequências operacionais. Ao fornecer aos modelos subsequências de eventos de tamanho variável, espera-se verificar o impacto do uso de janelas de eventos na precisão dos modelos e avaliar a capacidade dos algoritmos em lidar com dependências de curto e longo prazo dentro das sequências de eventos. Os resultados obtidos com a aplicação dos n-gramas foram comparados com modelos treinados sem a utilização dessa técnica, permitindo uma análise sobre os ganhos proporcionados pela modelagem de subsequências.

4.1.3. ANÁLISE EXPLORATÓRIA DO CONJUNTO DE DADOS

A Análise Exploratória de Dados (EDA) é um processo crucial no âmbito da Ciência e Análise de Dados que visa compreender e resumir as principais características de um conjunto de dados antes de aplicar técnicas de modelagem. Esse processo envolve a utilização de métodos estatísticos, gráficos e outras ferramentas analíticas para investigar a estrutura e os padrões dos dados, além de identificar anomalias, relações entre variáveis, e comportamentos inesperados. Entre as principais técnicas utilizadas na EDA estão o cálculo de estatísticas descritivas, como média, mediana e variância, além da criação de visualizações gráficas como histogramas, *box plots*, *scatter plots* e correlações entre variáveis. Essas técnicas ajudam os analistas a formarem hipóteses, compreender a distribuição dos dados e detectar possíveis *outliers* ou dados ausentes (CHATFIELD, 1995; BAILLIE *et al.*, 2022).

A EDA é importante, por exemplo, para permitir que os analistas de dados obtenham *insights* iniciais sobre os dados, fornecendo uma visão clara das tendências e padrões que podem influenciar a escolha de modelos e técnicas de aprendizado de máquina. Além disso, ela auxilia na detecção de problemas nos dados, como valores extremos, erros de medição ou valores ausentes, que podem comprometer a qualidade e a precisão dos modelos. Através da EDA, também é possível identificar relações entre variáveis, o que pode ser útil para a seleção de características ou para determinar quais variáveis são mais relevantes para um modelo preditivo. Em resumo, a EDA é um passo fundamental para garantir que os dados estejam prontos e bem compreendidos antes de

avançar para as fases de modelagem e interpretação, aumentando a eficiência e a precisão dos resultados.

Dessa forma, em um primeiro momento pode-se analisar todas as sequências de eventos em situações de *Trip* e compreender quais são as variáveis mais recorrentes, conforme pode ser visualizado na Figura 35. Esse processo serviu como um primeiro passo para observar quais eventos têm a maior tendência de serem indicativos prévios de que um desarme do reator poderá acontecer.

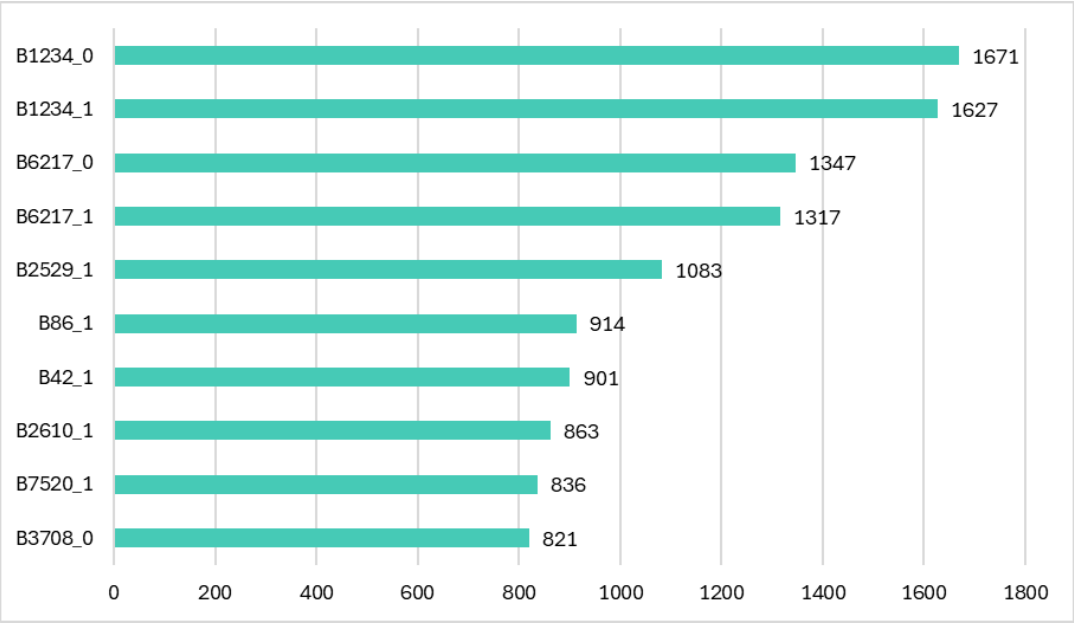

Figura 35. Eventos mais recorrentes em situações de *Trip* (De autoria própria)

A partir da Figura 35, observa-se que a variável *B1234* (Transmissor) é a mais recorrente, com 1.671 ocorrências no estado não-atuado (*B1234_0*) e 1.627 no estado atuado (*B1234_1***)**. Esse comportamento sugere que a variável *B1234* alterna frequentemente entre ativação e desativação previamente ou concomitantemente com os eventos de *Trip*. Essa oscilação indica que o sistema ou componente relacionado a essa variável está diretamente envolvido no processo de falha, sendo frequentemente desativado e reativado, talvez como uma tentativa de estabilizar o sistema, de prevenir um colapso maior ou a atuação recorrente do alarme. O mesmo padrão pode ser visto na variável *B6217* (Transmissor), que possui 1.347 ocorrências no estado desativado (*B6217_0*) e 1.317 no estado ativado (*B6217_1*). Esses valores próximos entre os estados ativado e desativado sugerem que o componente relacionado à variável *B6217* também

alterna de forma similar, o que reforça a hipótese de que esses sistemas estão lidando com transições rápidas de estado durante um evento crítico ou à iminência de ocorrência dele.

Já variáveis como *B2529* (Falha Atuador Lógico), B86 (Falha Atuador Lógico), e B42 (Falha Atuador Lógico) mostram um comportamento diferente. Essas variáveis são muito mais recorrentes em seus estados ativados, com 1.083 ocorrências para *B2529_1*, 914 para *B86_1*, e 901 para *B42_1*. Isso sugere que esses eventos ocorrem principalmente quando as variáveis estão ativas, o que é esperado por serem todas representativas de eventos de falha de atuadores lógicos, indicando que esses componentes desempenham alertas específicos quando o sistema está próximo da ocorrência de *Trip*. A ativação dessas variáveis parece estar fortemente associada a iminência de falha durante eventos críticos, talvez como representativo de alarmes de falha de componentes em regiões críticas da usina ou mecanismos de segurança acionados durante uma falha.

Essa análise revela padrões importantes sobre o comportamento dos sistemas durante um evento de *Trip*. A oscilação constante de algumas variáveis entre ativação e desativação pode ser um sinal de instabilidade no processo ou várias tentativas recorrentes de reverter o estado de falha, o que exige um monitoramento mais rigoroso. Além disso, a ativação recorrente de outras variáveis sugere que certos sistemas estão sob constante pressão durante esses eventos e podem requerer uma revisão para garantir que estejam operando da forma mais eficiente possível. Uma investigação mais profunda sobre as causas dessas transições frequentes, especialmente nas variáveis mais críticas, pode ser essencial para evitar falhas recorrentes e melhorar a confiabilidade geral do sistema.

Em resumo, compreender os padrões de ativação e desativação dessas variáveis é fundamental para identificar vulnerabilidades nos sistemas e adotar ações corretivas ou preventivas. Manter um foco e inspeção das variáveis mais recorrentes, como *B1234* e *B6217*, pode proporcionar *insights* valiosos sobre o comportamento do sistema durante falhas, enquanto o acompanhamento das variáveis ativas durante os *Trips* pode ajudar a melhorar as respostas automáticas a esses eventos críticos.

Após a análise das sequências de *Trip*, é possível também analisar as sequências de eventos em situações de operação normal e compreender quais são as variáveis mais recorrentes, conforme pode ser observado na Figura 36. Essa segunda etapa do processo

serviu para observar quais eventos têm a maior tendência de serem indicativos de operação normal, já esperados de ocorrer durante os ciclos de operação da usina.

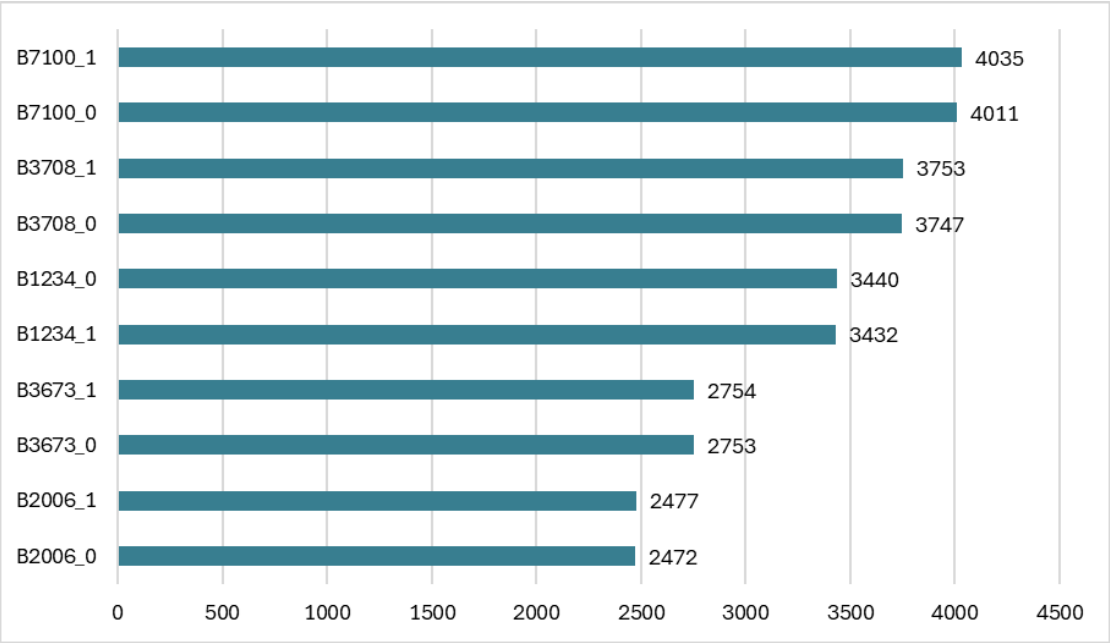

Figura 36. Eventos mais recorrentes em Operação Normal (De autoria própria)

A partir da Figura 36, pode-se observar que a variável *B7100* (CC Ar Comprimido) é a mais recorrente, com 4.035 ocorrências quando ativada (*B7100_1*) e 4.011 quando desativada (*B7100_0*). Esses números elevados indicam que essa variável, independentemente de estar ativada ou desativada, é altamente relevante na operação normal. A pequena diferença entre os estados sugere que o sistema monitorado pela variável *B7100* pode ser alternado com frequência, mas permanece constantemente dentro de uma zona de operação controlada, o que pode indicar uma função regulatória ou de estabilização do sistema ao qual ela está incluída.

Em seguida, tem-se as variáveis *B3708* (Transmissor) e *B1234* (Transmissor), ambas também apresentando uma alta quantidade de ocorrências tanto em estado ativado quanto desativado. A variável *B3708* aparece 3.753 vezes quando ativada (*B3708_1*) e 3.747 vezes quando desativada (*B3708_0*), enquanto *B1234* tem 3.440 ocorrências no estado desativado (*B1234_0*) e 3.432 no estado ativado (*B1234_1*). Esses valores são próximos, o que pode sugerir que esses componentes alternam entre os estados de forma controlada e previsível durante a operação normal, assim como a variável *B7100*. Elas

podem estar associadas a componentes ou processos que são ativados e desativados em função de condições operacionais específicas, mas que, no geral, mantêm um equilíbrio entre os dois estados. Contudo, é interessante notar que essas duas variáveis aparecem com alta frequência tanto no gráfico de eventos de *Trip* (Figura 35) quanto no de Não-*Trip* (Figura 36), indicando que uma análise de recorrência dessas variáveis não é suficiente para afirmar com exatidão se estão mais correlacionadas com um estado operacional ou outro, podendo ser indicativo de serem variáveis comumente ativadas e desativadas durante a operação normal ou transientes operacionais.

A variável *B3673* (Atuador), com 2.754 ocorrências no estado ativado (*B3673_1*) e 2.753 no estado desativado (*B3673_0*), também mostra uma alternância equilibrada entre os dois estados. Isso sugere que o comportamento de ativação e desativação da variável *B3673* é esperado e não representa anomalias na operação normal. Esse padrão reforça a ideia de que muitos dos componentes monitorados estão em um estado de funcionamento balanceado, alternando conforme necessário durante a operação.

Por fim, a variável *B2006* (Ativação da Bomba 1) aparece com 2.477 ocorrências no estado ativado (*B2006_1*) e 2.472 no estado desativado (*B2006_0*), mantendo, assim como as demais variáveis, uma pequena diferença entre os dois estados. A similaridade de ocorrências nos dois estados sugere um comportamento regular e cíclico dessa variável, que pode estar relacionada a processos que requerem alternância constante entre ativação e desativação, sem que isso afete adversamente a operação do sistema.

De forma geral, a análise dessas variáveis indica um comportamento estável e previsível durante a operação normal, com as variáveis mais recorrentes mantendo um equilíbrio entre os estados ativado e desativado. Esse tipo de padrão sugere que os sistemas estão funcionando dentro de suas especificações, com os componentes ou subsistemas monitorados alternando entre ativação e desativação de acordo com as demandas operacionais.

Uma questão importante durante a EDA é analisar o impacto de cada variável nos dados utilizados, ou seja, utilizar um algoritmo que seja capaz de identificar qual(is) variável(is) apresentam a maior relação/impacto com uma classe específica, sem necessariamente serem as variáveis mais recorrentes para essa classe. Dessa forma, foi

utilizado o algoritmo Ridge para verificar quais são os eventos mais impactantes em situações prévias a *Trips*, e os resultados obtidos podem ser observados na Figura 37.

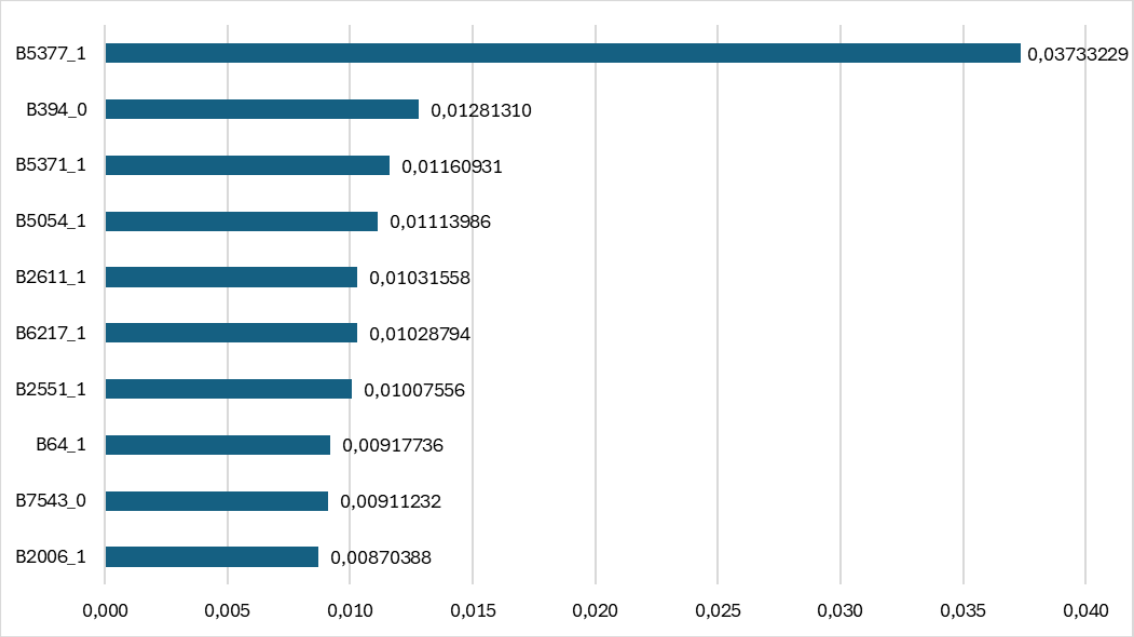

Figura 37. Impacto médio das *features* nos dados (De autoria própria)

Conforme mencionado, a análise da Figura 37 mostra as variáveis identificadas pelo algoritmo Ridge como aquelas com maior impacto em eventos de *Trip*. O gráfico exibe as variáveis (ou características) mais influentes, sendo que o eixo horizontal representa a magnitude da influência de cada variável no resultado, enquanto o eixo vertical apresenta os identificadores das variáveis e seus respectivos estados.

Observa-se que a variável *B5377_1* (Código de Liberação – Atuado) se destaca significativamente em comparação com as outras, possuindo uma magnitude muito maior de impacto em eventos de *Trip*. Essa variável ativada, com seu valor muito mais elevado que as demais, indica que o sistema ou componente relacionado a ela desempenha um papel crucial e preponderante em situações de falha. O comportamento anômalo desta variável parece estar fortemente associado à ocorrência de um *Trip*, tornando-a uma variável crítica a ser monitorada e indicativa de que o desarme do reator é iminente de ocorrer.

Outras variáveis com impacto relevante, embora menores que a *B5377_1*, incluem *B394_0* (Desligamento da Turbina – Atuada), *B5371_1* (RESA Manual – Atuado),

B5054_1 (Falha Atuador Lógico – Atuada), *B6217_1* (Transmissor – Falha), *B2611_1* (Desligamento da Turbina – Atuada), e *B2551_1* (Ativação Elemento Final – Não Atuada). As magnitudes dessas variáveis mostram que, embora elas também tenham influência em eventos de *Trip*, o impacto é menor quando comparado à variável *B5377_1*. Ainda assim, a ativação de variáveis representantes de falhas de componentes, como *B5054*, *B6217* e *B2611*, sugere que esses alarmes são ativados como parte de um processo que pode preceder ou estar envolvido em uma falha crítica do respectivo sistema.

A presença de instrumentos como a variável *B394* (Desligamento da Turbina) no estado atuado e outras variáveis ativadas, como *B2551_1* (Ativação Elemento Final) e *B5054_1* (Falha Atuador Lógico), indica que o algoritmo conseguiu identificar tanto eventos de ativação quanto de desativação de certos alarmes como fatores que contribuem para a ocorrência de *Trip*. A atuação do alarme *B394_0*, por exemplo, representa um evento de TUSA que corresponde a uma variável crítica que requer sinalização de alarme para o operador, e que está altamente correlacionada à ocorrência de *Trip*.

De modo geral, a presença tanto de variáveis ativadas quanto desativadas no gráfico mostra que o algoritmo está identificando transições de estado como elementos-chave que influenciam a ocorrência de *Trips*. O impacto desproporcional da variável *B5377_1* destaca sua importância, sugerindo que a monitorização mais rigorosa dessa variável, junto com uma análise das condições que levam à sua ativação, pode ser fundamental para prevenir futuras falhas e garantir a segurança operacional.

4.2. EXPERIMENTOS REALIZADOS PARA A IDENTIFICAÇÃO PRÉVIA DE TRIP

O desempenho do modelo de Rede Neural Transformer, LSTM e 5 modelos de algoritmos de Aprendizado de Máquina foram analisados na tarefa de identificação prévia de desarme do reator. Os modelos avaliados foram:

- **RIDGE**: Um algoritmo que apresentou bons resultados em trabalhos anteriores (HOERL, 1962; HOERL & KENNARD, 1970; HUANG *et al.*, 2020; WU & ZHAO, 2020; HUANG *et al.*, 2022; WU, 2023; WU & PAN, 2024) para problemas de classificação e regressão utilizando um modelo linear regularizado, especialmente em cenários com muitas características ou dados

ruidosos. Ele utiliza a regularização L2, que ajuda a reduzir o *overfitting* ao penalizar a magnitude dos coeficientes, tornando-o mais robusto em situações de alta dimensionalidade ou com um certo grau de multicolinearidade entre as variáveis. Em virtude disso, nesse trabalho, o algoritmo Ridge foi avaliado pela primeira vez na classificação prévia de *Trip* a fim de se verificar o comportamento de um modelo de classificação linear aplicado nesse problema.

- **FLORESTA ALEATÓRIA**: É um algoritmo amplamente utilizado para problemas de classificação devido à sua robustez e capacidade de lidar com grandes conjuntos de dados com muitas variáveis preditivas, conforme demonstrado em trabalhos anteriores (DONG *et al.*, 2014; GRAPE *et al.*, 2020; ONG *et al.*, 2022). Ele combina a capacidade de aprendizado de várias árvores de decisão, reduzindo a variância e melhorando a exatidão do modelo por meio da agregação das previsões das árvores individuais. Essa técnica também é eficaz para lidar com *overfitting*, já que a aleatoriedade introduzida na seleção de amostras e variáveis aumenta a capacidade de generalização do modelo. Dessa forma, esse algoritmo foi avaliado pela primeira vez no problema da classificação prévia de *Trip* para se verificar o impacto do modelo de votação das árvores de decisão aplicado a esse problema.
- **SGD**: É um algoritmo de otimização amplamente utilizado para ajustar os parâmetros de modelos de Aprendizado de Máquina, especialmente em grandes conjuntos de dados. Ele atualiza iterativamente os parâmetros do modelo com base em amostras individuais, o que o torna eficiente e escalável. No contexto de classificação, o SGD tem sido aplicado com sucesso na otimização de parâmetros em modelos lineares e não-lineares, como na regressão logística e em modelos SVM, conforme demonstrado em estudos anteriores (ZHANG, 2004; BOTTOU, 2010; ZHANG *et al.*, 2020). Com isso, esse método foi avaliado pela primeira vez no problema de identificação de *Trip*, sendo utilizado para otimizar os parâmetros de um modelo Linear SVM.
- **NAIVE BAYES**: É um classificador probabilístico baseado no Teorema de Bayes, que assume independência entre as variáveis. Apesar da suposição de

independência ser frequentemente irrealista na prática, o algoritmo Naive Bayes tem se mostrado eficaz em diversas aplicações, incluindo a classificação de textos e detecção de *spam*. Trabalhos anteriores (MCCALLUM & NIGAM, 1998; RENNIE *et al.*, 2003; DONG *et al.*, 2014) demonstram a aplicabilidade e eficiência deste algoritmo em tarefas de classificação e, devido a esses fatores, neste trabalho esse algoritmo foi utilizado pela primeira vez para a tarefa de identificação de *Trip*.

- **K-NEAREST NEIGHBORS**: É um algoritmo de classificação que atribui a classe de uma amostra com base nas classes das *k* amostras mais próximas no espaço das características do problema. É simples e não paramétrico, sendo útil em situações em que a distribuição dos dados é desconhecida. Estudos anteriores (COVER & HART, 1967; ALTMAN, 1992; LIN *et al.*, 1995; NAIMI *et al.*, 2022) exploram a eficácia do kNN em diferentes contextos de classificação e, dessa forma, esse algoritmo foi aplicado pela primeira vez na identificação de *Trip*.

- **LSTM**: Diversos trabalhos na literatura mostram a capacidade da LSTM em problemas de previsão de séries temporais e sequências de texto, porém, LSTMs são menos comumente aplicadas em problemas de classificação de sequências de texo. Ainda assim, dentro do escopo de trabalhos acadêmicos relacionados à área nuclear que aplicaram LSTMs para classificação (ZHA *et al.*, 2021; SAEED *et al.*, 2020; YANG *et al.*, 2020), bons resultados foram encontrados. Em virtude disso, uma LSTM foi avaliada no cenário de classificação prévia de *Trip* proposta neste trabalho.

- **TRANSFORMERS**: São modelos baseados em mecanismos de atenção que têm revolucionado o campo do Processamento de Linguagem Natural, especialmente em tarefas de classificação de texto. Sua capacidade de capturar dependências de longo alcance e contextos complexos os torna adequados para a análise de sequências textuais. No contexto de classificação de texto, estudos anteriores (VASWANI *et al.*, 2017; DEVLIN *et al.*, 2019) demonstram a eficácia dos Transformers em diversas aplicações, incluindo a identificação de padrões e anomalias em dados textuais. Além disso, trabalhos como (YI *et al.*,

2023; ZHOU *et al.*, 2024) abordaram a aplicação de Transformers em usinas nucleares, principalmente relacionados à detecção de anomalias durante a operação. Dessa forma, uma rede Transformer foi avaliada neste trabalho para a identificação prévia de *Trips* em sequências operacionais anômalas.

Deste modo, para os experimentos realizados, todos os modelos computacionais foram treinados, validados e testados com o conjunto de dados apresentado na seção 4.1.1. Para os 5 algoritmos de Aprendizado de Máquina (Ridge, Floresta Aleatória, SGD, Naive Bayes e Nearest Neighbors), foi utilizado o algoritmo TF-IDF para realizar a tokenização dos dados. Para o modelo LSTM foi utilizada a classe "Tokenizer" da biblioteca *TensorFlow* para realizar um ajuste sobre as sequências de texto disponíveis, de forma que esta classe permitiu vetorizar o *corpus* de texto disponível, transformando cada conjunto de eventos em uma sequência de valores inteiros (cada inteiro sendo o índice do respectivo *token* em um dicionário mapeado pelo tokenizador). Por padrão, toda a pontuação é removida, transformando os textos em sequências de termos separados por espaços. Essas sequências foram então divididas em um arranjo de *tokens*, sendo então indexadas ou vetorizadas para uma representação discreta com base no mapeamento realizado.

Para o modelo Transformer, foi utilizado o algoritmo de tokenização WordPiece, em conjunto com o BERT Tokenizer, desenvolvido pelo Google para melhorar a capacidade de processamento de linguagem natural em modelos como o BERT (*Bidirectional Encoder Representations from Transformers*) (DEVLIN *et al.*, 2019). A técnica WordPiece inicia com um vocabulário básico contendo caracteres individuais e *tokens* especiais e, a partir de um *corpus* de texto, o algoritmo identifica iterativamente pares de *tokens* cuja combinação maximize a probabilidade de ocorrência conjunta, ou seja, a frequência com que dois *tokens* aparecem lado a lado ou em combinação dentro do *corpus* de texto, com o objetivo de combinar esses pares de *tokens* em um único *token* composto que represente uma unidade informativa útil, considerando um equilíbrio entre frequência e informatividade. Esse processo de formação de novos *tokens* compostos expande o vocabulário do modelo com subpalavras frequentes no *corpus*, permitindo representar palavras inteiras ou subpalavras frequentes com um número menor de *tokens*. Durante o processamento de novas palavras, o WordPiece adota uma estratégia de

máxima correspondência (*longest-match-first*), ou seja, busca a sequência mais longa de caracteres que corresponda a um *token* no vocabulário. Caso a palavra não exista no vocabulário, ela é segmentada em subpalavras ou caracteres menores que estejam presentes no mesmo.

Durante a etapa de ajuste do documento de texto o algoritmo WordPiece identifica pares de palavras que ocorrem com frequência no *corpus* de treinamento e os combina para formar subpalavras. Por exemplo, a palavra "incompreensível" pode ser segmentada em ["in", "compr", "e", "ens", "ível"] ao combinar pares frequentes e, no contexto do BERT, os *tokens* resultantes poderiam ser, ["in", "##compr", "##e", "##ens", "##ível"] de forma que o prefixo "##" indica que o *token* é uma continuação de uma subpalavra anterior. Essa abordagem permite que o modelo lide com palavras desconhecidas ao decompô-las em subunidades presentes no vocabulário, melhorando a compreensão do contexto e do significado das palavras. Dessa forma, em vez de mapear todas as palavras existentes no *corpus*, o modelo pode reconhecer componentes menores como "in", "compr", "e", "ens" e "ível", associando-os a significados conhecidos e compreendendo a palavra como um todo, reconstruindo o sentido completo da palavra. O BERT utiliza o tokenizador WordPiece para converter texto em sequências de *tokens* que podem ser processadas pelo modelo, de forma que essa abordagem permite que o BERT lide eficientemente com palavras raras ou desconhecidas, decompondo-as em subunidades conhecidas, o que melhora a capacidade do modelo de entender e representar o significado contextual das palavras. Além disso, o uso de subpalavras ajuda o BERT a capturar nuances semânticas e lidar com variações morfológicas, tornando-o mais eficiente em diversas tarefas de Processamento de Linguagem Natural (DEVLIN *et al.*, 2019).

4.2.1. ARQUITETURAS E HIPERPARÂMETROS

Cada algoritmo utilizado neste trabalho apresenta características distintas que influenciam seu desempenho e aplicabilidade em diferentes contextos, de maneira que a maioria dos modelos possui uma série de pesos e parâmetros que são ajustados para otimizar seus resultados, levando em consideração tanto as especificidades do conjunto de dados quanto os objetivos da análise. As Tabelas 10 a 17 apresentam uma visão

detalhada desses parâmetros, incluindo os hiperparâmetros definidos para cada modelo: Ridge, Floresta Aleatória, SGD, Complement Naive Bayes, kNN, LSTM e Transformer. Para cada modelo, é fornecida uma explicação sobre a função e a importância de cada parâmetro, auxiliando a contextualizar as escolhas feitas durante a configuração e treinamento de cada um.

Além disso, para garantir a reprodutibilidade dos experimentos realizados, uma etapa crucial é a definição do estado aleatório. A configuração do estado aleatório, ou *seed*, foi fixada para todos os modelos, permitindo que os resultados possam ser reproduzidos, independentemente da execução do experimento em diferentes ambientes ou em momentos distintos. Isso é particularmente importante para garantir que os modelos sejam comparáveis e que as análises sejam consistentes ao longo do tempo, evitando variações que possam surgir devido a aleatoriedades no processo de treinamento e separação dos dados em treinamento, validação e teste. A fixação do estado aleatório também assegura que a avaliação dos modelos seja realizada sob condições controladas, fornecendo uma base confiável para a análise de seu desempenho.

Tabela 10. Parâmetros utilizados para o algoritmo Ridge

Parâmetro	Valor	Descrição
Alpha	1.0	Representa a força da regularização. A regularização melhora o condicionamento do problema e reduz a variância das estimativas. Valores maiores indicam uma regularização mais forte, tornando os coeficientes lineares do algoritmo Ridge menos suscetíveis a flutuações
Tolerância	10^{-4}	Indica a precisão da solução, atuando como o critério de parada do algoritmo, definido empiricamente
Técnica de solução do sistema de equações	*Sparse Conjugate Gradient*	É um método de otimização iterativo usado para resolver sistemas de equações lineares esparsos. Ele é eficiente para problemas de alta dimensionalidade, onde a regularização pode resultar em matrizes esparsas (HESTENES & STIEFEL, 1952)

Tabela 11. Parâmetros utilizados para o algoritmo Floresta Aleatória

Parâmetro	Valor	Descrição
Número de Estimadores	100	Define o número de árvores de decisão que serão criadas na floresta. Cada árvore é treinada em uma amostra aleatória do conjunto de dados de forma que, quanto maior o número de estimadores, mais robusto tende a ser o

		modelo, pois cada árvore contribui para reduzir a variância. Através de um mecanismo de voto majoritário entre todas as árvores, o algoritmo determina a classe associada às observações de entrada.
Critério	*Impureza de Gini*	É uma medida utilizada para avaliar a "pureza" dos nós nas árvores de decisão. O Critério *Gini* calcula a impureza de um nó com base nas frequências de classes das amostras. Um nó é considerado puro quando todas as amostras pertencem à mesma classe
Profundidade Máxima	N/A (Indeterminado)	Controla a profundidade de cada árvore na floresta. Árvores mais profundas têm maior capacidade de modelar complexidades dos dados (tendendo a um ajuste mais próximo). No entanto, uma profundidade excessiva pode levar ao *overfitting*, onde a árvore se adapta demais aos dados de treino e generaliza mal para novos dados. Um valor de profundidade indeterminado implica que os nós serão expandidos até que todas as folhas sejam puras
Poda por Complexidade de Custo Mínimo (*ccp_alpha*)	0,001	Controla a poda em árvores de decisão, controlando a complexidade do modelo ao eliminar divisões com baixo ganho de informação. Valores menores mantêm árvores mais profundas e complexas, enquanto valores maiores resultam em árvores mais simples e generalizáveis.

Tabela 12. Parâmetros utilizados para o algoritmo SGD

Parâmetro	**Valor**	**Descrição**
Função de Perda	*Log Loss*	A função de perda utilizada para treinar o modelo. *Log Loss* (ou perda logarítmica) é usada para problemas de classificação binária e penaliza predições incorretas, reajustando os pesos do modelo. Ela é representada pela função $L_{log}(y, p) = -(y \log(p) + (1 - y) \log(1 - p))$
Penalização	L2	Regularização que penaliza grandes valores nos coeficientes do modelo, ajudando a reduzir o *overfitting*. A regularização L2 (Ridge) adiciona uma penalidade proporcional ao quadrado da magnitude dos coeficientes, incentivando o modelo a aprender uma solução mais simples e menos propensa ao ruído dos dados
Alpha	10^{-4}	É o coeficiente de regularização que controla a força da penalidade L2. Valores mais altos de *alpha* aumentam a penalização, promovendo coeficientes menores e ajudando a reduzir a complexidade do modelo, o que é útil para evitar o *overfitting*, enquanto valores menores permitem que o modelo se ajuste melhor aos dados, limitando a capacidade de generalização
Número Máximo de Iterações	1.000	Define o número máximo de iterações (ou épocas) que o algoritmo vai executar para buscar o melhor ajuste
Tolerância	10^{-3}	Especifica o limite de tolerância para a convergência. Se a melhoria da função de perda entre uma iteração e outra

		for menor que o valor de Tolerância e satisfazer os critérios de parada, o algoritmo cessa o treinamento, considerando que atingiu uma solução satisfatória
Número de Iterações sem Mudança	100	Define o número de iterações consecutivas nas quais a função de perda precisa estar abaixo de Tolerância para que o algoritmo conclua que a convergência foi alcançada
Parada Antecipada	Sim	Habilita a parada antecipada do treinamento se a função de perda não apresentar melhorias significativas em um número específico de iterações consecutivas

Tabela 13. Parâmetros utilizados para o algoritmo Complement Naive Bayes

Parâmetro	Valor	Descrição
Alpha	10^{-2}	É um coeficiente de suavização usado para lidar com a ocorrência de atributos com valor zero nas classes, ajudando a evitar o problema de probabilidade zero. Esse problema ocorre quando uma característica (palavra ou atributo, por exemplo) não aparece em uma das classes do treinamento, levando a uma probabilidade nula que compromete o cálculo das probabilidades para essa classe. O parâmetro *alpha*, ao adicionar um valor positivo ao contador de cada característica, permite que todas as características tenham uma probabilidade mínima, evitando esse problema

Tabela 14. Parâmetros utilizados para o algoritmo kNN

Parâmetro	Valor	Descrição
Número de Vizinhos	5	Define a quantidade de vizinhos mais próximos (*k*) que o algoritmo considera para determinar a classe de uma amostra
Pesos	Uniforme	Controla a forma como o algoritmo pondera a contribuição dos vizinhos. Com pesos uniformes, cada vizinho tem o mesmo peso na votação, ou seja, todos os vizinhos contribuem igualmente para a determinação da classe da amostra
Algoritmo	*Ball Tree*	O algoritmo *Ball Tree* divide recursivamente os dados em nós definidos por um centroide e um raio, de modo que cada ponto no nó esteja contido na hiperesfera definida por esses valores. O número de pontos candidatos para uma busca de vizinhança é reduzido através do uso da desigualdade triangular. Com essa configuração, um único cálculo de distância entre um ponto de teste e o centroide é suficiente para determinar um limite inferior e superior da distância para todos os pontos dentro do nó

Tabela 15. Arquitetura da Rede Neural LSTM para a identificação de *Trip*

Camada	Formato da Saída	Descrição
Entrada (*Input*)	(, 50)	Corresponde ao formato dos dados de entrada, contendo sequências de até 50 eventos já previamente tokenizados
Token and Positional Embedding	(, 50, 128)	Uma camada de *Token Embedding* seguida de *Positional Embedding*, gerando um vetor de *embedding* que representa a soma de ambas as camadas
LSTM	(, 50, 128)	Duas camadas LSTM responsáveis por capturar as dependências de longo prazo entre os dados
LSTM	(, 64)	
Densa (ReLU)	(, 64)	Uma camada densa para proporcionar um grau de abstração adicional na rede
Densa (Sigmoide)	(, 1)	Uma camada de saída com função de ativação sigmoide para determinar qual classe a amostra de entrada representa
Total de parâmetros treináveis: 1.285.249		

Tabela 16. Hiperparâmetros da Rede Neural LSTM para a identificação de *Trip*

Parâmetro	Valor	Descrição
Tamanho do *Batch*	8	Representa o número de dados que são passados de cada vez (em lotes) ao modelo durante cada época do treinamento
Épocas de Treinamento	1.000	O número de vezes que todos os dados são passados ao modelo para reajustar os pesos e vieses neuronais
Gatilhos (*callbacks*)	*Early Stopping* (Paciência = 100), *Model Checkpoint* (Manter somente o melhor modelo)	O critério de parada previne o *overfitting* do modelo ao finalizar o treinamento previamente com base nas alterações do erro em cada época. Nesse caso, se o modelo permanecer 100 épocas seguidas em haver progresso significativo na função de perda o treinamento é interrompido. O critério de *checkpoint* mantém sempre o melhor modelo salvo ao final de cada época de treinamento
Otimizador	Adam (Taxa de Aprendizado = 10^{-3}, Função de Perda = Entropia Cruzada Binária)	O algoritmo de otimização é o método utilizado pela Rede Neural para minimizar a função de erro durante o treinamento
Padding	50 (prefixado)	Completa as sequências com menos de 50 *tokens* para terem exatamente 50 *tokens* ao acrescentar 0s à esquerda, para terem sempre um tamanho fixo a fim de serem passadas ao modelo. Sequências maiores que 50 são truncadas até atingirem o valor estabelecido

Tabela 17. Hiperparâmetros da Rede Neural Transformer

Parâmetro	Valor	Descrição
Modelo Pré-Treinado	DistilBERT (66 milhões de parâmetros)	O modelo DistilBERT mantém a arquitetura Transformer do BERT, utilizando atenção bidirecional para representar as palavras em um contexto completo, ou seja, ele considera tanto as palavras à esquerda quanto à direita de uma palavra para entender seu significado. A principal diferença é que este modelo possui 6 camadas de Transformer ao invés das 12 do modelo BERT tradicional (DEVLIN *et al.*, 2019), diminuindo consideravelmente o tempo de processamento, porém representando uma queda de performance, em média, de apenas 3% (SANH *et al.*, 2019)
Taxa de Aprendizado	$5 \cdot 10^{-5}$	Representa o fator com que os pesos neuronais são atualizados durante o treinamento
Tamanho do *Batch* para Treinamento	8	Representa o número de dados que são passados de cada vez (em lotes) ao modelo durante cada época de treinamento
Tamanho do *Batch* para Validação	8	Representa o número de dados que são passados de cada vez (em lotes) ao modelo durante cada época de validação
Estratégia de Validação	Época	Indica com que frequência é realizada a validação dos dados durante o treinamento. Ao final de cada época, todos os dados de validação são passados ao modelo para verificar o desempenho com dados nunca observados pela rede
Passos de Acumulação do Gradiente	8	É uma técnica que permite treinar a rede com tamanhos de lote maiores do que a capacidade de memória da máquina. Isso é feito acumulando gradientes ao longo de vários lotes e executando a atualização do otimizador apenas após um certo número de lotes terem sido processados
Taxa de Aquecimento	0,1	Define a proporção do treinamento a ser dedicada a um aquecimento linear, onde a taxa de aprendizado aumenta gradualmente. Isso pode ajudar a estabilizar o processo de treinamento no início (MA & YARATS, 2021)
Número de Épocas	3	O número de vezes que todos os dados são passados ao modelo para reajustar os pesos e vieses neuronais
Decaimento dos Pesos	0,01	Define a taxa de decaimento dos pesos do modelo a ser aplicada no processo de regularização, ajudando a evitar que o modelo sofra *overfitting* ao penalizar pesos elevados (KOBAYASHI *et al.*, 2024)
Coletor de Dados	*Data Collator with Padding*	Reúne uma lista de amostras em um minilote de treinamento. É responsável por preencher ou truncar as amostras de entrada para garantir que todas as amostras em um minilote tenham o mesmo comprimento
Tokenizador	WordPiece + BERT Tokenizer	Tem como objetivo dividir um texto em *tokens*, as menores unidades que o modelo pode processar. O processo de tokenização é realizado com a técnica WordPiece, que divide palavras em subunidades (subpalavras) com base em um vocabulário limitado, de forma que palavras comuns são mantidas como *tokens* inteiros, enquanto palavras raras ou desconhecidas são partidas em subpalavras que são mais frequentes no vocabulário do modelo

4.2.2. RESULTADOS

A Tabela 18 mostra a exatidão obtida por cada um dos modelos analisados para o conjunto de teste. Analisando o resultado, é possível perceber que o algoritmo *Complement Naive Bayes* (Complement NB), com sequências variando entre 20 e 50 eventos, foi o que obteve o pior resultado (destacado em vermelho). Contudo, para os demais modelos de Aprendizado de Máquina e Aprendizado Profundo, à medida que mais subsequências de n-gramas foram criadas, o desempenho dos modelos foi se tornando progressivamente melhor, com exceção das sequências de 20 a 50 eventos, onde todos os modelos apresentaram piores resultados. Esse comportamento se deve principalmente ao fato de que, ao utilizar somente 20 eventos para realizar uma classificação, os modelos tiveram dificuldade em conseguir distinguir uma sequência de *Trip* de Não-*Trip*, impactando diretamente na capacidade de generalização de cada modelo.

Embora os modelos LSTM e Transformer tenham obtido um comportamento similar, com ambos alcançando uma exatidão acima de 99,90% para sequências de 40 a 50 eventos, e a Transformer uma precisão de 99,98% para sequências de 30 a 50 eventos, ao analisar a exatidão média, na Tabela 18, é possível observar que os modelos utilizando Floresta Aleatória, SGD, LSTM e Transformer (destacados em azul) superaram a faixa de 99% de exatidão, com a Rede Transformer alcançando a maior exatidão média e menor desvio padrão, apresentando os resultados mais consistentes dentre os modelos analisados. Esse comportamento demonstra a capacidade dessa arquitetura de rede de identificar correlações de longo prazo entre as sequências de eventos mesmo sem a utilização de n-gramas e para sequências de eventos de menor comprimento.

Tabela 18. Resultados obtidos por cada modelo para o conjunto de teste para a identificação de *Trip*

SEQUÊNCIA	Ridge	Floresta Aleatória	SGD	Complement NB	kNN	LSTM	Transformer
50 (Fixa)	98,62%	98,62%	99,08%	96,56%	98,16%	99,66%	99,89%
40-50	99,49%	99,84%	99,84%	97,26%	99,22%	99,92%	99,96%
30-50	99,45%	99,90%	99,59%	96,98%	99,19%	99,93%	**99,98%**
20-50	98,13%	99,80%	99,51%	**95,87%**	98,97%	99,93%	99,95%

Média:	98,95%	**99,54%**	**99,50%**	96,66%	98,88%	**99,86%**	**99,94%**
Desvio Padrão (±):	0,58%	0,53%	0,27%	0,52%	0,43%	0,12%	0,04%

Além disso, foram analisados os diferentes tipos de erro cometidos por cada modelo, com o objetivo de identificar quais classes apresentam maior dificuldade de serem previstas para cada abordagem. Essa análise permitiu uma compreensão mais aprofundada das limitações e pontos críticos de cada modelo, proporcionando, posteriormente, uma base para sugerir melhorias e discussões que possam elevar a exatidão e a confiabilidade das previsões, como por exemplo reajustar os hiperparâmetros de cada modelo, aumentar o número de sequências e rebalancear a quantidade de dados disponível em cada classe. Os tipos de erro observados foram classificados em:

- **Falso Positivo (FP)**: ocorre quando o algoritmo classifica erroneamente uma sequência como "Trip", embora, na realidade, essa sequência não represente uma condição de falha. Esse erro resulta em um "alarme falso" por parte do modelo, o que pode acarretar potenciais interrupções indesejadas da operação.

- **Falso Negativo (FN)**: ocorre quando o algoritmo classifica erroneamente uma sequência como "Não Trip", embora a situação de fato represente uma condição de falha. Esse tipo de erro é especialmente crítico, pois leva a uma subestimação da situação, o que pode resultar em uma resposta insuficiente ou atrasada a uma condição que requer atenção imediata.

Para avaliar o desempenho dos modelos, foram utilizadas as métricas de *Recall*, *F1 Score* e *Taxa de Falso Positivo* (ou probabilidade de alarme falso), essenciais para a análise de precisão e robustez dos modelos na determinação das diferentes classes. A seguir cada uma das métrica é descrita em mais detalhes:

- ***Recall***: também conhecida como sensibilidade, essa métrica indica a proporção de verdadeiros positivos em relação ao total de instâncias que realmente pertencem à classe positiva. No contexto de detecção de falhas, o *Recall* mede a capacidade do modelo de identificar todas as falhas presentes. O cálculo desta métrica é dado pela equação (53):

$$Recall = \frac{TP}{TP + FN} \tag{53}$$

onde *TP* é a fração de Verdadeiros Positivos (*true positive*) i.e., casos em que o algoritmo identificou corretamente uma sequência de *Trip*.

- ***F1 Score***: corresponde à média harmônica entre as métricas de precisão e *Recall*, fornecendo uma única medida de desempenho que equilibra ambos os aspectos. Essa métrica é particularmente útil quando há um desequilíbrio entre as classes, pois fornece uma visão conjunta de quantas falhas foram detectadas e da precisão dessas detecções. O *F1 Score* é calculado através da equação (54):

$$F1\ Score = 2\frac{Precisão \cdot Recall}{Precisão + Recall} = \frac{2TP}{2TP + FP + FN} \tag{54}$$

- **Taxa de Falso Positivo (FPR):** essa métrica mede a proporção de instâncias que foram classificadas erroneamente como "Trip" quando, na realidade, não representavam uma condição de falha. De certo modo, é possível considerá-la como a medida de probabilidade de que, ao receber uma sequência de eventos, o modelo o categorize incorretamente como *Trip* quando na verdade seria uma sequência de eventos em operação normal. No contexto de Teste de Hipótese, a Taxa de Falso Positivo está associada ao erro do Tipo I, que ocorre quando se rejeita a hipótese nula (ausência de falha) incorretamente, gerando um "alarme falso". Esse erro é especialmente importante em sistemas de monitoramento, pois alarmes falsos podem gerar ações desnecessárias, custos adicionais e desgaste da confiança no sistema. Matematicamente, a Taxa de Falso Positivo é calculada através da equação (55):

$$FPR = \frac{FP}{N} = \frac{FP}{FP + TN} \tag{55}$$

onde *N* é a quantidade de instâncias de Não-*Trip* na população de dados utilizada no experimento e *TN* é a fração de Verdadeiros Negativos (*true negative*) i.e., casos em que o algoritmo identificou corretamente uma sequência de Não-*Trip*.

Dessa forma, para cada um dos experimentos realizados (50, 40-50, 30-50 e 20-50 eventos) foram descritos os resultados nas Tabelas 19, 20, 21 e 22, respectivamente.

As tabelas mostram, para cada modelo, a proporção de Falso Positivos e Falso Negativos, assim como as métricas de *Recall*, *F1 Score* e a Taxa de Falso Positivo (FPR), como pode ser observado a seguir:

Tabela 19. Erros obtidos para o conjunto de teste de sequências de 50 eventos

Algoritmo	Falso Positivo	Falso Negativo	Recall	F1 Score	FPR
Ridge	0,46%	0,92%	97,95%	98,46%	1,03%
Floresta Aleatória	**0,80%**	0,57%	98,70%	98,45%	**1,81%**
SGD	0,46%	0,46%	98,97%	98,97%	1,03%
Complement NB	**0,80%**	**2,64%**	**94,06%**	**96,04%**	**1,81%**
kNN	0,69%	1,15%	97,40%	97,91%	1,55%
LSTM	0,11%	0,23%	99,48%	99,61%	0,26%
Transformer	**0,00%**	**0,11%**	**99,74%**	**99,87%**	**0,00%**

A partir da Tabela 19, é possível observar que o modelo Transformer apresentou os melhores resultados, alcançando 99,74% de *Recall* e 99,87% de *F1 Score*, com taxa de falso positivo de 0,00%, indicando uma capacidade superior de prever todos os casos positivos e evitar alarmes falsos. Já a LSTM também apresentou um excelente desempenho, com *Recall* de 99,48% e *F1 Score* de 99,61%, ressaltando a alta capacidade de processamento de sequências de dados de ambas as arquiteturas. Em contraste, os piores resultados foram obtidos com o modelo Complement NB, que apresentou o maior percentual de falso negativos (2,64%) e o menor *Recall* (94,06%) e *F1 Score* (96,04%), evidenciando as dificuldades em identificar sequências de *Trip* com alta precisão em comparação com os demais modelos. Já o algoritmo kNN obteve um desempenho mediano, com *Recall* de 98,42%, mas com percentuais de falso positivo (0,69%) e falso negativo (1,15%) relativamente altos em relação aos outros modelos analisados, indicando que o algoritmo teve maior dificuldade em distinguir corretamente uma classe da outra a partir dos dados de teste.

Tabela 20. Erros obtidos para o conjunto de teste de sequências de 40-50 eventos

Algoritmo	Falso Positivo	Falso Negativo	Recall	F1 Score	FPR
Ridge	0,08%	0,43%	98,41%	99,05%	0,11%
Floresta Aleatória	0,05%	0,11%	99,60%	99,70%	0,07%

SGD	0,01%	0,15%	99,45%	99,70%	0,02%
Complement NB	**1,25%**	**1,49%**	**94,34%**	**94,78%**	**1,72%**
kNN	0,35%	0,43%	98,42%	98,57%	0,48%
LSTM	0,03%	0,05%	99,80%	99,85%	0,04%
Transformer	**0,01%**	**0,03%**	**99,90%**	**99,93%**	**0,02%**

Com a Tabela 20, é possível observar que as arquiteturas Transformer e LSTM novamente obtiveram os melhores resultados, com *Recall* e *F1 Score* acima de 99,80% e taxas de erro (FP, FN, FPR) próximas de 0,00%. Os algoritmos Floresta Aleatória e SGD também se destacaram com *Recall* e *F1 Score* superiores a 99,45%, embora com eventuais erros de classificação que, na prática, podem gerar complicações devido à classificação errônea do tipo de sequência. O modelo Complement NB novamente apresentou os piores resultados, com *Recall* de 94,34% e a maior taxa de falso positivo (1,72%) dos modelos avaliados, porém, com a utilização da técnica de n-gramas, apresentou resultados melhores do que com a utilização de janelas de eventos fixas. Além disso, o modelo kNN mostrou uma performance inferior em relação a modelos como Floresta Aleatória e SGD, com *Recall* de 98,41%. Dessa forma, com a introdução de n-gramas variando entre 40 e 50 eventos, houve uma melhoria de desempenho em todos os modelos, impactando diretamente em suas capacidades de generalização.

Tabela 21. Erros obtidos para o conjunto de teste de sequências de 30-50 eventos

Algoritmo	**Falso Positivo**	**Falso Negativo**	**Recall**	**F1 Score**	**FPR**
Ridge	0,16%	0,39%	98,75%	99,12%	0,23%
Floresta Aleatória	0,04%	0,06%	99,81%	99,84%	0,06%
SGD	0,19%	0,22%	99,31%	99,35%	0,28%
Complement NB	**1,54%**	**1,49%**	**95,10%**	**95,02%**	**2,25%**
kNN	0,36%	0,45%	98,59%	98,72%	0,53%
LSTM	0,03%	0,03%	99,90%	99,90%	0,05%
Transformer	**0,00%**	**0,02%**	**99,94%**	**99,97%**	**0,00%**

A partir da Tabela 21, pode-se observar que o modelo Transformer novamente manteve os melhores resultados, apresentando *Recall* e *F1 Score* acima de 99,94%, e com taxas de erro bem próximas de 0,00%. A rede LSTM também mostrou alta eficiência,

com *Recall* e *F1 Score* de 99,90%, reforçando a capacidade de capturar as dependências de longo alcance dos modelos de aprendizado profundo. O algoritmo Floresta Aleatória obteve um *Recall* de 99,81% e *F1 Score* de 99,84%, ressaltando a capacidade de generalização do modelo baseado em árvores de decisão para a tarefa de classificação. O modelo Complement NB apresentou o pior desempenho novamente, com *Recall* de 95,10%, *F1 Score* de 95,02% e taxas de erro relativamente elevadas (FP: 1,54%, FN: 1,49%). O algoritmo SGD obteve uma leve piora em comparação com a utilização de n-gramas de 40-50 eventos, mas ainda apresentando resultados bem satisfatórios, acima de 99%, com *Recall* de 99,31% e *F1 Score* de 99,35%. Dessa forma, o uso de janelas menores de n-gramas mostrou um indicativo de beneficiar os modelos mais complexos, mantendo a consistência nos resultados, porém em alguns modelos houve piora da capacidade de generalização em comparação com os resultados da Tabela 20.

Tabela 22. Erros obtidos para o conjunto de teste de sequências de 20-50 eventos

Algoritmo	Falso Positivo	Falso Negativo	Recall	F1 Score	FPR
Ridge	0,38%	1,49%	95,62%	97,20%	0,58%
Floresta Aleatória	0,07%	0,13%	99,62%	99,70%	0,11%
SGD	0,23%	0,26%	99,22%	99,26%	0,35%
Complement NB	**2,00%**	**2,14%**	**93,56%**	**93,76%**	**3,03%**
kNN	0,39%	0,64%	98,13%	98,49%	0,59%
LSTM	0,03%	0,04%	99,89%	99,90%	0,05%
Transformer	**0,02%**	**0,03%**	**99,92%**	**99,93%**	**0,03%**

Por fim, para a Tabela 22, os modelos Transformer e LSTM mais uma vez se destacaram, com *Recall* e *F1 Score* acima de 99,89%, reforçando a capacidade de generalização desses modelos mesmo com sequências menores a partir de 20 eventos. O modelo Floresta Aleatória apresentou desempenho considerável, com *Recall* de 99,62% e *F1 Score* de 99,70%, mostrando sua adaptabilidade no cenário apresentado. Seguindo os experimentos anteriores, o Complement NB obteve o pior desempenho dentre os modelos treinados, com *Recall* de 93,56% e *F1 Score* de 93,76%, além de apresentar as maiores taxas de erro (FP: 2,00%, FN: 2,14%). O kNN manteve-se abaixo da média, com *Recall* de 98,85% e *F1 Score* de 98,49%, ficando aquém de modelos como Floresta Aleatória e SGD. Com isso, apesar do aumento na complexidade das janelas de n-gramas

e a introdução de uma maior dificuldade em separar uma sequência de *Trip* de Não-*Trip*, os modelos Transformer e LSTM permaneceram consistentes entre todos os experimentos.

A partir dos resultados apresentados, o modelo Transformer obteve o melhor desempenho em todos os experimentos realizados, demonstrando a capacidade desta arquitetura em classificar corretamente a grande maioria dos casos em que de fato houve desarme do reator, com 99,94% de exatidão média. É possível observar que a abordagem com o maior percentual em que o modelo Transformer classificou incorretamente sequências de *Trip* como operação normal (FN ≠ 0,00%) foi no experimento em que não houve a utilização de n-gramas, reforçando a utilização desse método e outros similares para a ampliação do conjunto de dados de sequências de eventos e geração de janelas de eventos sobrepostas, com o objetivo de auxiliar os modelos a diferenciarem uma classe da outra e apresentarem uma maior capacidade de generalização.

Além do Transformer, a LSTM demonstrou um desempenho consistentemente alto, alcançando *Recall* e *F1 Score* acima de 99,40% em todos os experimentos, além de uma exatidão média de 99,86%, o que reforça a robustez dessa arquitetura em cenários de dados sequenciais, mesmo em condições menos ideais, como nos casos em que houve a necessidade do modelo classificar sequências a partir de apenas 20 eventos, ou nos casos em que ela precisou analisar as dependências de longo prazo para sequências de 50 eventos. Ambas as arquiteturas, portanto, apresentaram a maior capacidade de generalização e precisão em classificações complexas de sequências de eventos, dentre os modelos avaliados.

Os modelos tradicionais, como SVM+SGD, sendo uma variação de um modelo linear, e Floresta Aleatória, baseado em árvores de decisão, apresentaram resultados sólidos em termos de precisão e *Recall*, especialmente em experimentos com a aplicação da técnica de n-gramas. No entanto, eles foram superados pelos modelos Transformer e LSTM, que mostraram maior adaptabilidade e capacidade de capturar relações mais complexas nos dados.

Por outro lado, modelos como Ridge, Complement NB e kNN apresentaram desempenho inferior, com maiores taxas de falsos positivos e falsos negativos, sendo o

Complement NB o algoritmo que apresentou os piores resultados em todos os experimentos, indicando limitações significativas na tarefa de classificação de dados sequenciais. Empiricamente, pode-se analisar que a limitação da aplicabilidade deste método nesse trabalho está relacionada com a suposição de que o método Naive Bayes considera que todas as características do problema são condicionalmente independentes entre si, o que nesse caso provou-se menos plausível de ser verdade para todos os casos. Isso destaca a importância de se utilizar modelos mais sofisticados em aplicações críticas como as analisadas neste trabalho.

Além disso, a Figura 38 ilustra um comparativo entre cada um dos 7 algoritmos analisados ao variar o comprimento da sequência para valores entre 20 e 50 eventos, de forma que cada valor no gráfico indica a precisão alcançada utilizando no mínimo o número de eventos indicado no eixo das abcissas. Com isso, é possível observar que a precisão máxima de cada algoritmo se encontra entre no mínimo 30 e 40 eventos de entrada e, conforme a sequência mínima foi tendendo a 20, na maioria dos casos a precisão também diminuiu. Para sequências próximas de 50, a técnica de n-gramas foi tendendo ao caso base de sequências fixas de 50 eventos, fazendo com que os modelos tivessem maior dificuldade de identificar padrões nos dados para realizar previsões mais confiáveis.

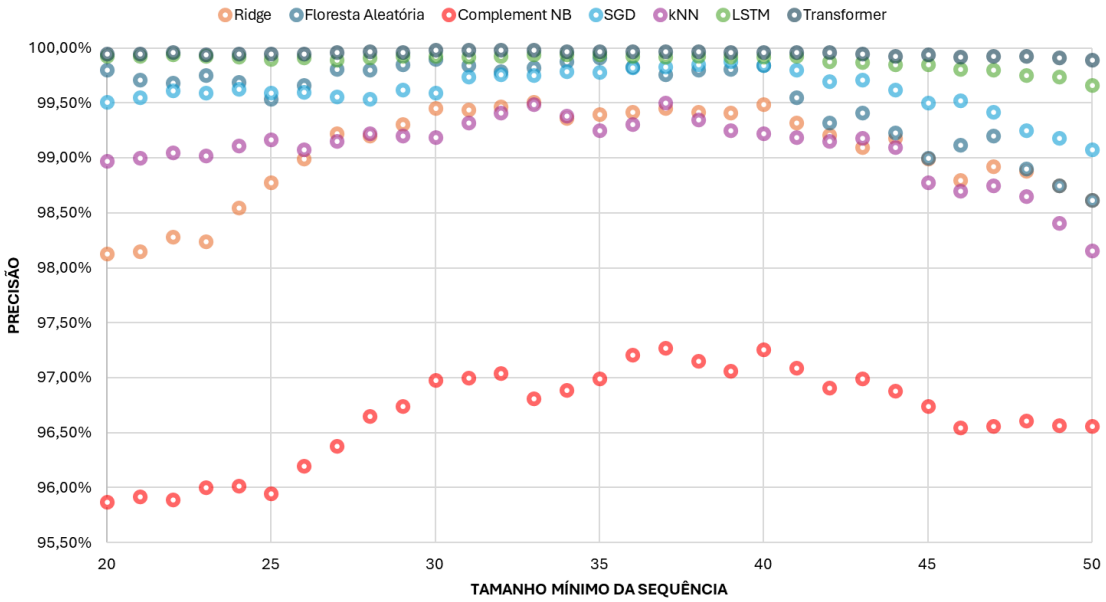

Figura 38. Comparativo entre os algoritmos utilizados para a identificação de *Trip* para diferentes comprimentos de sequência (De autoria própria)

Por fim, a utilização de n-gramas mostrou-se essencial para melhorar a performance geral dos modelos, especialmente ao reduzir erros de classificação em arquiteturas mais avançadas como Transformers e LSTMs. Esses resultados reforçam a necessidade de estratégias de enriquecimento de dados e pré-processamento para maximizar a eficácia de modelos de aprendizado profundo em cenários de alta criticidade. Além disso os resultados obtidos são altamente promissores para aplicações em usinas nucleares, como a previsão de falhas e o suporte à tomada de decisão em cenários de alta criticidade. A capacidade de identificar padrões anômalos antes que eles se desenvolvam em situações críticas pode melhorar significativamente a segurança operacional, o tempo de resposta e a eficiência da planta, reduzindo custos associados a paradas não planejadas.

4.3. EXPERIMENTOS REALIZADOS PARA A PREVISÃO DO PRÓXIMO EVENTO

O desempenho do modelo de Rede Neural Transformer e LSTM foi analisado na tarefa de previsão do próximo evento de uma sequência, devido à sua capacidade de lidar com longas sequências de dados, extraindo padrões complexos e considerando dependências de longo alcance entre as variáveis de entrada. Os modelos avaliados foram:

- **LSTM**: As LSTMs foram desenvolvidas especificamente para superar os desafios encontrados nas RNNs tradicionais, como o desaparecimento e a explosão do gradiente. Com sua estrutura recorrente baseada em células de memória, as LSTMs são capazes de capturar informações relevantes de longo prazo em dados sequenciais. Essa característica é especialmente valiosa para prever o próximo evento em sequências de eventos na operação de uma usina nuclear, onde eventos anteriores podem ter impacto direto em eventos futuros. Trabalhos prévios confirmam a eficácia das LSTMs na modelagem de sequências temporais em domínios semelhantes (CHOI & LEE, 2020; CHEN *et al.*, 2021; CHEN *et al.*, 2023), alcançando resultados precisos e satisfatórios em sistemas complexos. Dessa forma, uma rede LSTM foi avaliada neste trabalho para prever o próximo evento de sequências de eventos.

- **TRANSFORMERS**: As redes Transformers representam um avanço significativo em modelos de processamento de linguagem natural, eliminando a necessidade de processamento sequencial característico das redes *feedforward* e LSTMs. Por meio do mecanismo de atenção, os Transformers atribuem pesos diferentes a cada elemento da sequência, permitindo identificar as partes mais relevantes para a previsão do próximo evento, sendo essa abordagem particularmente eficiente em cenários onde a relação entre eventos pode ser complexa e não necessariamente linear. Na literatura, os Transformers têm sido amplamente aplicados em domínios envolvendo processamento de linguagem natural, geração e tradução de textos e previsão de séries temporais. Trabalhos anteriores (QI *et al.*, 2024; OH *et al*., 2024; ZHA *et al.*, 2021; XIAN *et al.*, 2024; XIAO *et al.*, 2024) exploraram Transformers e LLMs para a previsão e geração de sequências de texto no formato seq2seq no contexto da operação de usinas nucleares, demonstrando a ampla aplicabilidade dessa arquitetura para tarefas de processamento de linguagem natural em sistemas complexos.

Deste modo, para os experimentos realizados, todos os modelos computacionais foram treinados, validados e testados em conformância com o conjunto de dados apresentado na seção 4.1.1, porém neste caso descartando a coluna de rótulos. Além disso, a fim de se verificar as limitações das dependências de longo prazo das redes LSTM e o impacto do mecanismo de atenção nas redes Transformer para grandes documentos de texto, o conjunto de dados foi ampliado para 4 subconjuntos no total, seguindo a distribuição a seguir:

- Sequências de 20 a 50 eventos.
- Sequências de 20 a 100 eventos.
- Sequências de 20 a 500 eventos.
- Sequências de 20 a 1.000 eventos.

Dessa forma, as tabelas de resultados e as análises irão refletir a performance de cada modelo de Aprendizado Profundo ao variar o comprimento das sequências de eventos. Além disso, para os modelos de rede LSTM foram desenvolvidas arquiteturas

com e sem a utilização de uma camada inicial de *Token and Position Embedding*, responsável por gerar um nível adicional de abstração ao combinar uma camada de *embedding* de *tokens* e outra de *embedding* posicional e somando as saídas de cada uma dessas camadas. O resultado esperado é que a utilização desta camada permita ao modelo LSTM aprender características semânticas e posicionais dos dados de sequência de eventos, assim como permitir que o modelo compreenda relações entre eventos, como similaridades e contextos de uso. Sendo assim, esses resultados foram comparados com uma rede LSTM convencional e uma rede Transformer.

Assim como realizado no problema da identificação prévia de *Trips*, para construir os *datasets* com sequências de 50, 100, 500 e 1000 eventos, foi empregada a técnica dos n-gramas, que também contribuiu para o aumento do número de dados disponíveis (*data augmentation*). Essa técnica consistiu em definir uma janela de tamanho fixo que percorre a sequência de dados original, gerando novas amostras a partir de subsequências parcialmente sobrepostas. Ao deslizar a janela ao longo dos eventos, é possível criar novas instâncias de sequências, cada uma contendo informações deslocadas em relação à anterior, mas mantendo o contexto sequencial dos dados, de maneira que, a partir de cada amostra de comprimentos mínimo e máximo, N_{min} e $N_{máx}$, respectivamente, foram gerados n-gramas contendo até $N_{máx} - 1$ eventos de entrada (o restante do vetor de eventos é preenchido com zeros* ou com o *token* de *padding* utilizado pelo tokenizador até atingir $N_{máx}$ termos), sendo o enésimo evento correspondente à saída esperada do modelo. Um exemplo simples pode ser observado a seguir onde, a partir de uma sequência de eventos original, foram obtidos diferentes n-gramas:

	X (Input)						**y (Evento previsto)**
Sequência original:	**1**	**2**	**3**	**4**	**5**	**6**	**-**
1º n-grama:	**1**	**2**	0	0	0	0	3
2º n-grama:	**1**	**2**	**3**	0	0	0	4
3º n-grama:	**1**	**2**	**3**	**4**	0	0	5
4º n-grama:	**1**	**2**	**3**	**4**	**5**	0	6

*Os *tokens* de *padding* (0 neste exemplo) são utilizados pelo modelo para determinar qual deve ser a máscara de atenção, ou seja, quais *tokens* o modelo deve ignorar na sequência de entrada

Dessa forma, a partir de uma sequência original de 6 eventos, foi possível obter 4 sequências compostas por 2 (N_{min}) a 5 ($N_{máx} - 1$) eventos, contribuindo para a diversificação do conjunto de amostras de treinamento e validação dos modelos.

Esse processo é especialmente vantajoso em modelos que se beneficiam de um volume maior de dados, como as arquiteturas LSTM e Transformer, pois permite aumentar a diversidade e a representatividade das sequências de treinamento sem necessidade de adquirir novos dados. Dessa forma, ao aplicar a técnica de n-gramas, os *datasets* com diferentes comprimentos de sequência (50, 100, 500 e 1000 eventos) foram enriquecidos com uma variedade maior de padrões de eventos, o que se espera contribuir para a melhoria da generalização e robustez dos modelos ao lidar com diferentes contextos e transientes operacionais.

4.3.1. ARQUITETURAS E HIPERPARÂMETROS

Os algoritmos utilizados neste problema possuem características distintas que impactam diretamente seu desempenho e aplicabilidade em diferentes cenários, onde cada modelo empregado de rede LSTM e Transformer conta com um conjunto específico de parâmetros que são ajustados de forma a otimizar os resultados obtidos, de maneira que esses ajustes levam em consideração tanto as particularidades do conjunto de dados analisado quanto os objetivos específicos do trabalho. Nas Tabelas 23, 24 e 25, são apresentados detalhes abrangentes sobre esses parâmetros, incluindo seus valores definidos para cada modelo. Além disso, é fornecida uma explicação sobre a função e a relevância de cada parâmetro, destacando o papel que desempenham na configuração e no treinamento dos algoritmos. Com isso, essa abordagem visa oferecer um panorama claro e fundamentado das decisões tomadas durante o processo de modelagem, permitindo compreender como essas escolhas influenciam o desempenho final dos modelos avaliados.

Além disso, assim como nos experimentos voltados à identificação de *Trip*, a definição do estado aleatório dos pesos e vieses neuronais foi realizada para assegurar a reprodutibilidade dos resultados obtidos. A configuração do estado aleatório foi padronizada em todos os modelos, possibilitando a reprodução dos resultados mesmo quando os experimentos forem executados em diferentes ambientes ou momentos, de

maneira que essa abordagem é essencial para garantir a comparabilidade entre os modelos e a consistência das análises ao longo do tempo, minimizando variações decorrentes de fatores aleatórios durante o treinamento e proporcionando condições controladas para a avaliação dos modelos, oferecendo uma base sólida e confiável para a análise do desempenho de cada um.

Tabela 23. Arquitetura da Rede Neural LSTM para a previsão de eventos

Camada	Formato da Saída	Descrição
Entrada (Input)	(, X*)	Corresponde ao formato dos dados de entrada, contendo sequências de até *X* eventos já previamente tokenizados
Token and Positional Embedding**	(, X, 128)	Uma camada de *Token Embedding* seguida de *Positional Embedding*, gerando um vetor de *embedding* que representa a soma de ambas as camadas. É uma camada responsável por capturar relações semânticas e contextuais dos dados da camada de entrada
LSTM	(, X, 64)	Duas camadas LSTM responsáveis por capturar as dependências de longo prazo entre os dados
LSTM	(, 64)	
Densa (ReLU)	(, 64)	Uma camada densa para proporcionar um grau de abstração adicional na rede
Densa (Softmax)***	(, 1320)	Uma camada de saída com função de ativação *softmax* para determinar qual é o evento com a maior probabilidade de ser o próximo da sequência

* Representa o comprimento do vetor de entrada para cada experimento, variando entre 50, 100, 500 e 1.000

****** Foram desenvolvidas arquiteturas com e sem a camada de *Token and Positional Embedding*, influenciando o número total de parâmetros treináveis do modelo

*** O número de dimensões dessa camada representa a quantidade de eventos distintos observados no conjunto de dados

Tabela 24. Hiperparâmetros da Rede Neural LSTM para a previsão de eventos

Parâmetro	Valor	Significado
Tamanho do Batch	8	Representa o número de dados que são passados de uma vez (em lotes) ao modelo durante cada época do treinamento
Épocas	500	O número de vezes que todos os dados são passados ao modelo para reajustar os pesos neuronais
Gatilhos (*callbacks*)	*Early Stopping* (Paciência = 100) *Model Checkpoint* (Manter somente o melhor modelo)	O critério de parada previne o *overfitting* do modelo ao finalizar o treinamento previamente com base nas alterações do erro

		em cada época. Se o modelo permanecer 100 épocas seguidas em haver progresso significativo na função de perda o treinamento é interrompido
Otimizador	Adam (Taxa de Aprendizado = 10^{-3}, Função de Perda = Entropia Cruzada Categórica Esparsa)	O algoritmo de otimização é a função utilizada pela Rede Neural para minimizar o erro durante o treinamento
Padding	Dependente do conjunto de dados, sendo 50, 100, 500 e 1.000 para cada experimento	Completa as sequências com menos de X *tokens* para terem exatamente X *tokens* ao acrescentar 0s à esquerda, para terem sempre um tamanho fixo a fim de serem passadas ao modelo. Sequências maiores que X são truncadas até atingirem o valor determinado

A utilização de uma função *softmax* na camada de saída da Rede Neural LSTM foi utilizada como abordagem para a previsão do próximo elemento de uma determinada sequência pois a função *softmax* transforma os valores de saída da rede em uma distribuição de probabilidade sobre as possíveis classes ou estados seguintes, facilitando a escolha do próximo elemento da sequência entre várias alternativas, semelhante ao observado na saída do bloco de decodificação da rede Transformer. Esse aspecto é particularmente relevante em aplicações como previsão de palavras em sequências de texto, onde o modelo precisa atribuir uma probabilidade a cada possível próximo elemento, selecionando aquele com maior probabilidade de acordo com essa distribuição. Outro ponto importante é que a *softmax* ajuda a reduzir a incerteza nas previsões da LSTM pois, ao invés de retornar valores arbitrários para cada classe, a rede neural gera uma lista de probabilidades que representam o grau de confiança em cada possível saída, o que se traduz em previsões mais interpretáveis. Essa representação probabilística é útil tanto para avaliar o desempenho do modelo quanto para refinar sua otimização, já que a confiança expressa em cada previsão ajuda a sinalizar a precisão relativa do modelo.

Além disso, o uso de *softmax* ajuda a LSTM a ajustar dinamicamente seus pesos em resposta aos padrões encontrados nas sequências de entrada. Como a saída de uma célula LSTM é influenciada pelos seus estados ocultos anteriores, a *softmax* orienta o modelo a ajustar seus pesos, melhorando as previsões futuras conforme os padrões da sequência são identificados. Esse mecanismo se torna ainda mais valioso em aplicações que requerem precisão e contexto, onde a dependência de longo prazo dos dados pode variar.

Por fim, a *softmax* oferece uma interpretação intuitiva dos resultados ao fornecer uma distribuição de probabilidade entre 0 e 1 para cada possível próximo elemento da sequência, com a soma totalizando 1. Essa representação torna os resultados do modelo mais interpretáveis, permitindo escolher diretamente o conjunto variável-estado mais provável.

No contexto de previsão do próximo evento de uma sequência foi optado por utilizar o modelo GPT-2 *Medium* para realizar o ajuste e previsão dos dados, de forma que, para este modelo, foi utilizado o tokenizador BPE + ByteLevel Tokenizer. A técnica Byte Pair Encoding (BPE) (GAGE, 1994), utilizada em modelos Transformer como o GPT (RADFORD *et al.*, 2018) e RoBERTa (LIU *et al.*, 2019), é uma técnica de tokenização que equilibra a divisão de texto entre caracteres individuais e palavras inteiras. O BPE começa tratando cada caractere como um *token* separado e, iterativamente, combina os pares de *tokens* adjacentes mais frequentes para formar novos tokens compostos. Esse processo continua até que o vocabulário atinja um tamanho predefinido, permitindo que *tokens* comuns sejam representados como unidades únicas, enquanto palavras raras são decompostas em subunidades menores. Uma extensão dessa abordagem é o Byte-Level BPE, que opera diretamente no nível dos *bytes* em vez de caracteres ou subpalavras. Isso significa que o texto é inicialmente dividido em *bytes* individuais, permitindo que o tokenizador lide com qualquer caractere Unicode sem a necessidade de um vocabulário inicial específico, sendo essa técnica particularmente útil para lidar com caracteres raros ou desconhecidos e outros símbolos que podem não estar presentes em vocabulários baseados em caracteres. Modelos como o GPT-2 e o RoBERTa utilizam o Byte-Level BPE para garantir uma cobertura abrangente de caracteres e símbolos, independentemente de sua frequência no *corpus* de treinamento.

Tabela 25. Hiperparâmetros da Rede Neural Transformer para a previsão de eventos

Parâmetro	Valor	Significado
Modelo Pré-Treinado	GPT-2 Medium (355 milhões de parâmetros)	A principal modificação em relação ao GPT-2 base é o aumento do número de parâmetros, possuindo 24 camadas de Transformer ao invés das 12 do modelo GPT-2, 1024 unidades ocultas ao invés das 768 e 16 cabeças de atenção ao invés das 12 (RADFORD *et al*., 2019). Foi projetado

		para realizar tarefas de modelagem de linguagem autoregressiva, onde a geração de cada *token* depende dos *tokens* anteriores na sequência (auto-atenção causal)
Taxa de Aprendizado	$5 \cdot 10^{-5}$	Representa o fator com que os pesos neuronais são atualizados durante o treinamento
Tamanho do Batch para Treinamento	8	Representa o número de dados que são passados de uma vez (em lotes) ao modelo durante cada época do treinamento
Tamanho do Batch para Validação	8	Representa o número de dados que são passados de uma vez (em lotes) ao modelo durante cada época da validação
Estratégia de Validação	Época	Indica com que frequência é realizada a validação dos dados durante o treinamento. Neste caso, ao final de cada época todos os dados de validação são passados ao modelo para verificar a exatidão com dados nunca observados pela Rede
Passos de Acumulação do Gradiente	8	É uma técnica que permite treinar com tamanhos de lote maiores do que a capacidade de memória da máquina. Isso é feito acumulando gradientes ao longo de vários lotes e executando a atualização do otimizador apenas após um certo número de lotes terem sido processados.
Taxa de Aquecimento	0,1	Define a proporção do treinamento a ser dedicada a um aquecimento linear, onde a taxa de aprendizado aumenta gradualmente. Isso pode ajudar a estabilizar o processo de treinamento no início.
Número de Épocas	3	O número de vezes que todos os dados são passados ao modelo para reajustar os pesos neuronais
Decaimento dos Pesos	0,01	Defina a taxa de decaimento de peso para aplicar na regularização. Isso ajuda a evitar que o modelo sofra overfitting ao penalizar pesos elevados.
Tokenizador	BPE + Byte-Level Tokenizer	Tem como objetivo dividir um texto em *tokens*, unidades menores que o modelo pode processar. O *Byte Pair Encoding* (BPE) utilizado no GPT-2 é um algoritmo de tokenização que divide palavras em subunidades menores com base na frequência de pares de caracteres no texto. Esse processo permite que o modelo trabalhe com um vocabulário menor e mais eficiente, ao mesmo tempo em que mantém a flexibilidade para lidar com palavras raras ou desconhecidas.

4.3.2. RESULTADOS

A Tabela 26 apresenta a exatidão obtida por cada um dos modelos testados em relação ao conjunto de teste para a previsão do evento seguinte. Ao observar os resultados, nota-se que o modelo LSTM, configurado com 1.000 eventos de entrada e sem a utilização da camada inicial de *embedding* de *tokens* e posições (T&PE), apresentou o

pior desempenho entre os modelos analisados, seguida pela mesma arquitetura de rede LSTM, porém com a utilização dessa camada. Esse comportamento pode ser explicado pela dificuldade da LSTM em lidar com sequências de eventos muito longas, de maneira que, à medida que o tamanho das sequências aumenta, a capacidade da LSTM de reter informações do início da sequência torna-se limitada, devido, em grande parte, ao problema do desaparecimento do gradiente. Esse fenômeno ocorre porque o gradiente diminui progressivamente durante o processo de retropropagação em redes recorrentes, tornando-se imperceptível para eventos distantes do instante de tempo atual. A utilização dos portões lógicos das células LSTM se provou ser uma solução para atenuar o problema do desaparecimento do gradiente nas RNNs, porém, devido principalmente à utilização de funções como tangente hiperbólica e sigmoide como ativação dos portões, as suas respectivas derivadas para os cálculos realizados durante a retropropagação acabam por tornar o gradiente muito pequeno para arquiteturas muito profundas e para processamento de documentos muito longos, além de demandar uma alta capacidade computacional e de armazenamento em memória das informações. Como resultado, as LSTMs acabam apresentando uma certa dificuldade em capturar dependências de longo prazo para documentos muito grandes, impactando negativamente sua capacidade de generalização.

Em contrapartida, o modelo Transformer se mostrou resiliente ao problema do desaparecimento do gradiente, mesmo com o aumento do comprimento das sequências. Graças ao mecanismo de atenção, o Transformer consegue atribuir pesos diferentes a cada evento na sequência, garantindo que a relevância de informações mais antigas seja mantida ao longo do processamento. Essa arquitetura possibilita ao Transformer preservar relações contextuais ao longo de longas sequências, resultando em uma alta exatidão que se mantém consistente mesmo com o crescimento do tamanho das sequências. Esse diferencial torna o Transformer uma alternativa promissora para tarefas com dados sequenciais extensos, especialmente em contextos em que a retenção de informações ao longo da sequência é crucial para a precisão do modelo.

Tabela 26. Resultados obtidos por cada modelo para o conjunto de teste para a previsão do próximo evento de uma sequência

SEQUÊNCIA	LSTM (sem T&PE)	LSTM (com T&PE)	Transformer
50	99,45%	99,56%	99,76%

100	99,21%	99,25%	**99,90%**
500	97,16%	98,11%	99,83%
1000	**96,73%**	97,58%	99,77%
Média:	98,14%	98,63%	**99,82%**
Desvio Padrão (±):	1,21%	0,81%	0,06%

A incorporação da camada de *embedding* de *tokens* e posições em uma arquitetura de rede LSTM demonstrou-se uma abordagem promissora e inovadora para aprimorar o processamento da informação de entrada, sendo ela composta por sequências temporais ou textuais. Essa camada, essencial na arquitetura das redes Transformer, no caso da LSTM cumpre duas funções principais: primeiro, gera *embeddings* que capturam informações semânticas e sintáticas dos *tokens* de entrada; e segundo, cria codificações posicionais que indicam a posição de cada evento na sequência, proporcionando que a LSTM capture relações posicionais entre termos de sequências muito longas. Essa combinação proporcionou um nível adicional de abstração às representações de entrada, permitindo que as camadas LSTM subsequentes aprendessem não apenas as relações contextuais entre os eventos, mas também a ordem em que eles ocorrem, o que é crucial para o entendimento de dependências temporais ou contextuais que definem o significado global da sequência. Esse comportamento foi observado através dos resultados obtidos na Tabela 26 onde, para todos os experimentos realizados variando o comprimento da sequência, foi possível constatar que a arquitetura LSTM com a utilização da camada de *embedding* impactou positivamente os resultados sem comprometer o tempo de inferência e a complexidade do modelo.

Esse aprimoramento no processamento das sequências potencializa a eficácia da arquitetura LSTM em tarefas que exigem sensibilidade ao contexto, como tradução automática, análise de séries temporais, processamento de linguagem natural, classificação de textos e predição baseada em eventos. Além disso, essa integração também pode facilitar a convergência do treinamento, ao fornecer ao modelo informações estruturais mais ricas na etapa inicial de processamento, promovendo um aprendizado mais eficiente para as camadas mais profundas.

Outrossim, a seguir estão apresentados os resultados médios obtidos para a previsão realizada por cada modelo dos próximos *n* eventos a partir dos experimentos realizados para os diferentes tamanhos máximos de sequência de entrada. Em um primeiro momento, a Figura 39 contém a frequência de erros obtidos pelo modelo LSTM sem a camada de *embeddings* de *tokens* e posições para a previsão dos próximos 100 eventos, dadas sequências de entrada variando de 20 a 1000 eventos.

A partir de uma análise do gráfico, é possível observar que cada hexágono agrupa múltiplos pontos (erros) em uma determinada região do espaço (índice do evento previsto × tamanho da sequência de entrada), de forma que a cor dos hexágonos indica quantos erros ocorreram naquela região compreendida pelo hexágono. Dessa forma, as cores mais claras indicam uma maior frequência de erros, enquanto cores mais escuras indicam uma menor quantidade de erros concentrados naquela região.

Com isso, é observado que o modelo LSTM teve maior dificuldade em prever a partir do 20° evento, principalmente para comprimentos de sequência de entrada até 150 eventos. Conforme o tamanho da sequência de entrada aumentou, a taxa de erros foi diminuindo, permitindo que o modelo fosse capaz de prever até 15 eventos, em média, com uma boa precisão, incluindo a variável e seu estado binário. Para conseguir prever uma maior quantidade de eventos, o modelo precisou de mais eventos de entrada para ser capaz de identificar determinados padrões de ordem e posição dos eventos na sequência.

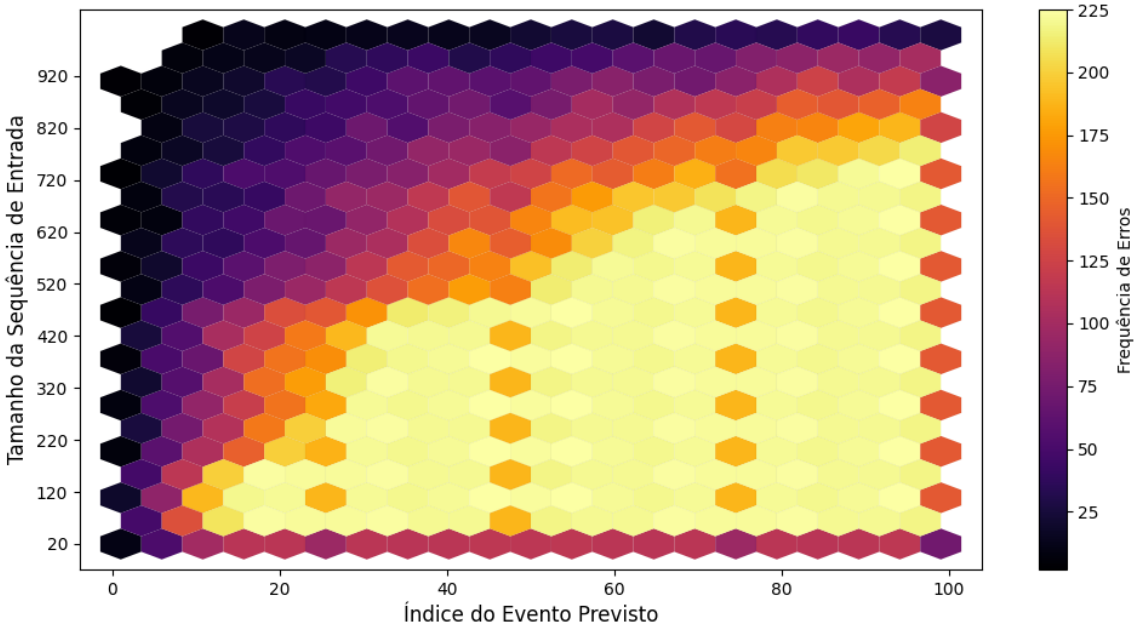

Figura 39. Frequência de erros obtidos para a previsão do enésimo evento ao variar o tamanho da sequência de entrada para a rede LSTM sem T&PE (De autoria própria)

Para o modelo LSTM com a camada inicial de *embedding* de *tokens* e posições, houve uma melhoria considerável na capacidade de generalização do modelo, principalmente para sequências de evento de comprimento intermediário, entre 120 e 620 eventos, conforme pode ser observado na Figura 40. A partir dessa imagem, nota-se uma redução da frequência de erros para tamanhos de sequência menores e eventos mais próximos, indicando que os *embeddings* ajudaram na representação das relações entre variáveis e posições na sequência. Porém, ainda há uma frequência de erros que aparecem de maneira significativa para eventos mais distantes, o que sugere que, apesar da melhoria, a LSTM ainda apresenta limitações no aprendizado de longas dependências temporais.

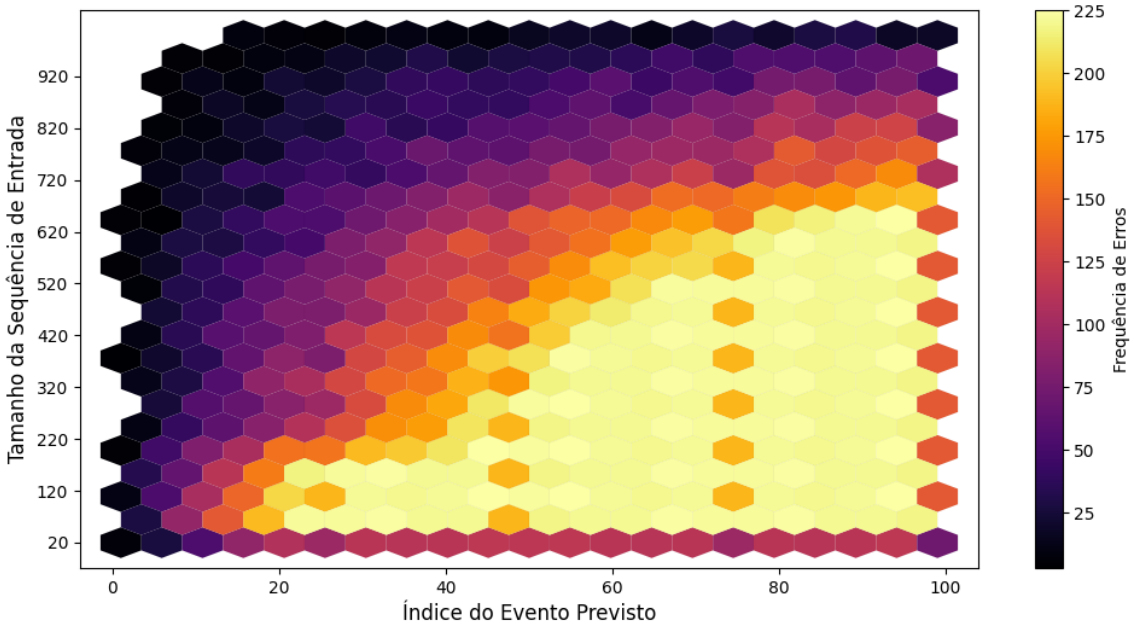

Figura 40. Frequência de erros obtidos para a previsão do enésimo evento ao variar o tamanho da sequência de entrada para a rede LSTM com T&PE (De autoria própria)

Por fim, os resultados obtidos com o modelo Transformer indicaram a capacidade dessa arquitetura em identificar relações e padrões entre os eventos de entrada e, ao utilizar o mecanismo de atenção em vez da recorrência das redes LSTM, o Transformer foi capaz de capturar dependências de longo alcance de maneira mais eficaz. Conforme

pode ser observado na Figura 41, os erros obtidos são menos dispersos e ocorrem predominantemente nos eventos mais distantes, mas de forma mais controlada em comparação com as LSTMs. Além disso, a tendência de erro se concentra em regiões específicas e não se espalha tanto ao longo do espaço de entrada, sugerindo que o modelo Transformer conseguiu aprender melhor as relações estruturais da sequência, permitindo prever eventos para sequências de entrada com maior variabilidade de comprimento.

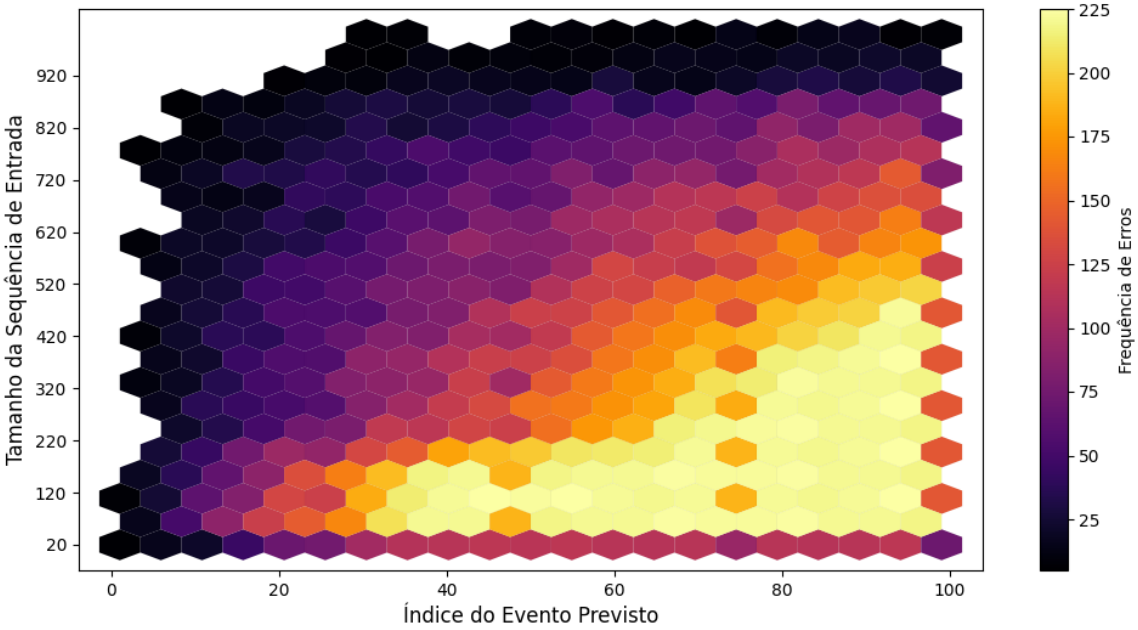

Figura 41. Frequência de erros obtidos para a previsão do enésimo evento ao variar o tamanho da sequência de entrada para a rede Transformer (De autoria própria)

CAPÍTULO 5
CONCLUSÕES

O presente trabalho analisou a aplicação de Redes Neurais Transformers, outros métodos de Deep Learning e algoritmos clássicos de Aprendizado de Máquina na identificação prévia de *Trip* em reatores nucleares do tipo PWR, assim como na previsão de sequências de eventos em situações de transientes operacionais. Os resultados obtidos demonstram a relevância e a viabilidade de soluções baseadas em inteligência artificial para aprimorar a segurança e a eficiência operacional de usinas nucleares.

O estudo, fundamentado em uma base de dados de sequência de eventos de 10 anos de operação da usina nuclear Angra 2, explorou algoritmos avançados de RNAs como LSTM e Transformers, configurando-os para capturar padrões temporais e realizar previsões com precisão significativa, de maneira que os modelos desenvolvidos alcançaram resultados notáveis, com taxas de exatidão média de até 99,94% na identificação de eventos de Trip e 99,82% na previsão de eventos subsequentes. Tais resultados superam abordagens tradicionais e confirmam o potencial de métodos baseados em Aprendizado Profundo e Mecanismo de Atenção para lidar com cenários de alta complexidade e criticidade.

Além do desenvolvimento técnico, a pesquisa destacou a importância de etapas como a análise exploratória de dados, a seleção de características relevantes e o ajuste de hiperparâmetros para um melhor desempenho dos modelos, fatores que impactaram diretamente a capacidade de generalização e tempo de convergência dos algoritmos analisados.

Este trabalho não apenas oferece uma ferramenta para aplicação prática, mas também contribui para o avanço do conhecimento científico na interface entre Engenharia Nuclear e Inteligência Artificial. Dessa forma, é importante destacar algumas limitações e oportunidades para estudos futuros, como por exemplo: apesar dos resultados promissores, a generalização dos modelos para outras usinas ou cenários operacionais deve ser explorada, de forma que o uso de técnicas de transferência de aprendizado (*transfer learning*) e *fine-tuning* pode ser um caminho para adaptar os modelos a novas

condições com menor custo computacional. Além disso, a implementação em tempo real do sistema e a integração com sistemas de controle e monitoramento, como a rede SICA, representam um desafio técnico e organizacional que requer estudos adicionais e uma análise de aplicabilidade, principalmente no que se diz respeito ao tempo de inferência de modelos mais complexos, como as redes Transformers e LLMs.

Além da eficácia dos modelos, a explicabilidade das decisões realizadas por eles é fundamental em aplicações críticas. Métodos como SHAP (do inglês, *SHapley Additive exPlanations*) (LUNDBERG & LEE, 2017) ou LIME (do inglês, *Local Interpretable Model-agnostic Explanations*) (RIBEIRO *et al.*, 2016) podem ser usados para aumentar a transparência e a confiança na adoção de tais ferramentas. O SHAP um método baseado na teoria dos valores de Shapley, oriunda da teoria dos jogos, para explicar as previsões de modelos de aprendizado de máquina. Ele atribui a cada *feature* uma contribuição quantitativa para a previsão final, calculando o impacto marginal de cada feature ao ser adicionada ao modelo (LUNGBERG *et al.*, 2020). O LIME é um método que fornece explicações locais para previsões individuais, criando um modelo simplificado que aproxima o comportamento do modelo original em torno de uma instância específica. Ele funciona perturbando os dados de entrada e avaliando como o modelo responde, permitindo identificar quais variáveis foram mais importantes em uma previsão específica (RIBEIRO *et al.*, 2016). Dessa forma, a utilização destes métodos pode ser uma alternativa interessante para explicar o porquê determinados eventos e a ordem em que eles acontecem ocasionam o desarme do reator ou são possíveis causas de transientes operacionais.

Como demais propostas de trabalhos futuros, pode-se mencionar a utilização de algoritmos não-supervisionados para identificação de *Trip* e previsão de eventos, como algoritmos de clusterização, ou outras técnicas de Aprendizado Profundo, como VAEs (do inglês, *Variational Autoencoders*) ou GANs (do inglês, *Generative Adversarial Networks*). Também pode-se mencionar a implementação de modelos de Transformer e demais modelos de Inteligência Artificial apropriados ao âmbito de Processamento de Linguagem Natural que sejam capazes de entregar uma resposta, comentário ou descrever a linha de raciocínio utilizada durante a inferência para o operador, como é observado em modelos atuais de LLM com capacidade de raciocínio, tais quais os modelos OpenAI o1

e *Deep Seek* (GUO *et al.*, 2025). Outro ponto a ser mencionado é a utilização de técnicas de *Grid Search* ou algoritmos de otimização (Computação Evolucionária ou Inteligência de Enxame, por exemplo) para encontrar conjuntos ótimos de hiperparâmetros para os diferentes algoritmos de ML utilizados neste trabalho. Além disso, pode-se mencionar a utilização de arquiteturas emergentes de RNAs para processamento de linguagem natural e sequências de texto, como redes xLSTM (BECK *et al.*, 2024) e redes Kolmogorov-Arnold (LIU *et al.*, 2024a). Por fim, trabalhos futuros podem verificar o impacto de diferentes técnicas de otimização do uso de memória nas Redes Transformers, principalmente no que diz respeito ao uso de um grande volume de dados, matrizes de alta dimensionalidade e uma elevada quantidade de parâmetros treináveis (bilhões ou trilhões de parâmetros).

Conclui-se, portanto, que este trabalho fornece uma base sólida para futuras aplicações de Aprendizado Profundo na operação de usinas nucleares, promovendo avanços em segurança, eficiência e capacidade preditiva. Ao mesmo tempo, reforça a necessidade de colaboração interdisciplinar e contínua inovação tecnológica para enfrentar os desafios do setor nuclear no século XXI ao incentivar o uso de modelos emergentes de Inteligência Artificial para auxiliar na tomada de decisão em cenários de alta complexidade.

REFERÊNCIAS BIBLIOGRÁFICAS

ABADI, M. et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

AINSLIE, J. et al. Gqa: Training generalized multi-query transformer models from multi-head checkpoints. arXiv:2305.13245, 2023.

ALTMAN, N. S. An Introduction to Kernel and Nearest Neighbor Nonparametric Regression. The American Statistician, 46, 175-185, 1992.

ALVARENGA, M. A. B. et al. Adaptative Vector Quantization Optimized by Genetic Algorithm for Real Time Diagnosis Through Fuzzy Sets. Nuclear Technology, v. 120, n.3, p. 188-197, 1997.

ALVES, A. C. P. D. Um Sistema de Análise de "Trip" em Reatores PWR Usando Redes Neuronais. 1993. 147 f. (Tese - Doutorado em Engenharia Nuclear) Universidade Federal do Rio de Janeiro.

ANALA, S. A guide to word embeddings. Towards Data Science, 26 out. 2020. Disponível em: https://towardsdatascience.com/a-guide-to-word-embeddings-8a23817ab60f. Acesso em: 5 dez. 2024.

ANKIT, U. Transformer neural networks: A step-by-step breakdown. Built In. Disponível em https://builtin.com/artificial-intelligence/transformer-neural-network, Acesso em: 6 dez. 2024.

AUGUSTO, J. P. S. C. Módulos de diagnóstico do sistema inteligente de suporte ao diagnóstico de Angra I. 2017. 66 f. (Trabalho de Conclusão de Curso - Graduação em Engenharia Nuclear) Universidade Federal do Rio de Janeiro.

BA, J. L. et al. Layer normalization. arXiv:1607.06450, 2016.

BAHDANAU, D. et al. Neural machine translation by jointly learning to align and translate. arXiv:1409.0473, 2014.

BAILLIE M., et al. for the Topic Group "Initial Data Analysis" of the STRATOS Initiative (2022). Ten simple rules for initial data analysis. PLoS Comput Biol 18(2): e1009819. https://doi.org/10.1371/journal.pcbi.1009819, 2022.

BAUM, E. B., WILCZEK, F. Supervised learning of probability distributions by neural networks. in D.Z. Anderson, ed., pp. 52-61, New York: American Institute of Physics, 1988.

BECK, M. et al. xLSTM: Extended Long Short-Term Memory. arXiv:2405.04517, 2024.

BECKER, S. Unsupervised learning procedures for neural networks, International Journal of Neural Systems, vol. 2, pp. 17-33, 1991.

BECKER, S., LECUN, Y. Improving the convergence of back-propagation learning with second order methods. in D. Touretzky, G.E. Hinton, and T.J. Sejnowski, eds., Proceedings of the 1988 Connectionist Models Summer School, pp. 29-37, San Fransisco: Morgan Kaufmann, 1989.

BIRD, S. et al. Natural Language Processing with Python. O'Reilly Media. 2009.

BOTTOU, L. Large-Scale Machine Learning with Stochastic Gradient Descent. In: Lechevallier, Y., Saporta, G. (eds) Proceedings of COMPSTAT'2010. Physica-Verlag HD. https://doi.org/10.1007/978-3-7908-2604-3_16, 2010.

BOTTOU, L., BOUSQUET, O. The tradeoffs of large-scale learning. Optimization for Machine Learning, pp. 351-368. MIT Press, 2011, doi: 10.7551/mitpress/8996.003.0015.

BRANDON, W. et al. Reducing Transformer Key-Value Cache Size with Cross-Layer Attention. arXiv:2405.12981, 2024.

BREIMAN, L. Random Forests. Machine Learning 45, 5-32, 2001, doi: 10.1023/A:1010933404324.

BREIMAN, L. et al. Classification and Regression Trees. Boca Raton, FL: Chapman & Hall, 1984.

BUCILUǍ, C. et al. Model compression. In: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining. 2006. p. 535-541.

CARUANA, R., NICULESCU-MIZIL, A. An Empirical Comparison of Supervised Learning Algorithms. Proceedings of the 23rd international conference on Machine learning - ICML '06. 2006, 161-168, 2006, doi: 10.1145/1143844.1143865.

CHATFIELD, C. Problem Solving: A statistician's guide, Second edition (2nd ed.). Chapman and Hall/CRC. https://doi.org/10.1201/b15238, 1995.

CHECHIK, G. et al. Large-Scale Online Learning of Image Similarity Through Ranking. Journal of Machine Learning Research. 11. 1109-1135, 2010.

CHEN, H. et al. Automatic scram forecasting during abnormal conditions of nuclear power plants based on LSTM. In: International Conference on Nuclear Engineering. American Society of Mechanical Engineers, 2022. p. V003T03A010.

CHEN, H. et al. Prediction of Automatic Scram during Abnormal Conditions of Nuclear Power Plants Based on Long Short-Term Memory (LSTM) and Dropout. Sci. Technol. Nucl. Install. 2023, doi: 10.1155/2023/2267376.

CHEN, Y. et al. Research on Simulation and State Prediction of Nuclear Power System Based on LSTM Neural Network. Science and Technology of Nuclear Installations. 1-11; 2021, doi: 10.1155/2021/8839867.

CHO, K. et al. Learning phrase representations using RNN encoder-decoder for statistical machine translation. arXiv:1406.1078, 2014.

CHOI, J, Lee S. J. Consistency Index-Based Sensor Fault Detection System for Nuclear Power Plant Emergency Situations Using an LSTM Network. Sensors. 2020; 20(6):1651. https://doi.org/10.3390/s20061651.

CHURCHLAND, P. S., SEJNOWSKI T.J. The Computational Brain, Cambridge, MA: MIT Press, 1992.

CLEVERT, D. A. et al. Fast and accurate deep network learning by exponential linear units (elus). arXiv preprint arXiv:1511.07289, 2015.

CORTES, C., VAPNIK, V. Support-vector networks. Mach Learn 20, 273-297. https://doi.org/10.1007/BF00994018, 1995.

COVER, T., HART, P. Nearest neighbor pattern classification, in IEEE Transactions on Information Theory, vol. 13, no. 1, pp. 21-27, 1967, doi: 10.1109/TIT.1967.1053964.

CULLBERT, C. et al. State-of-the-practice in knowledge-based systems verification and validation. Expert Systems with Applications 3: 403-410, 1991, doi: 10.1016/0957-4174(91)90167-D.

CYBENKO, G. Approximations by superpositions of sigmoidal functions, Mathematics of Control, Signals, and Systems, 2 (4), 303-314, 1989.

DATACAMP. What is Tokenization? Types, Use Cases, Implementation. DataCamp Blog, 22 nov. 2024. Disponível em: https://www.datacamp.com/pt/blog/what-is-tokenization. Acesso em: 05 dez. 2024.

DELLER, J., HANSEN, J. Methods, Models, and Algorithms for Modern Speech Processing. 10.1016/B978-012170960-0/50063-3, 2005.

DESTERRO, F. S. M. et al. Prediction of LOCA's break size and location based on random forest and Multi-Tasking Deep Neural Network. NUCLEAR ENGINEERING AND DESIGN, v. 415, p. 112711, 2023.

DEVLIN, J. et al. Bert: Pre-training of deep bidirectional transformers for language understanding. In: Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers). 2019. p. 4171-4186.

DONG, L. et al. Nonlinear Methodologies for Identifying Seismic Event and Nuclear Explosion Using Random Forest, Support Vector Machine, and Naive Bayes Classification. Abstract and Applied Analysis. 2014. 1-8. 10.1155/2014/459137, 2014.

DU, K. L. Clustering: A neural network approach. Neural networks: the official journal of the International Neural Network Society. 23. 89-107, 2010, http://dx.doi.org/10.1016/j.neunet.2009.08.007.

ELETRONUCLEAR. Curso de Formação de Operadores Licenciáveis. 2001.

ELETRONUCLEAR. Informações de Angra 2. Disponível em: https://www.eletronuclear.gov.br/Nossas-Atividades/Paginas/Informacoes-de-Angra-2.aspx. Acesso em: 6 dez. 2024.

ESTER, M. et al. A density-based algorithm for discovering clusters in large spatial databases with noise. Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96). AAAI Press. pp. 226-231, 1996.

FERGUSON, T. S. An Inconsistent Maximum Likelihood Estimate. Journal of the American Statistical Association, 77(380), 831-834, 1982, doi: 10.2307/2287314.

FISCHLER, M. A., FIRSCHEIN, O. Intelligence: the eye, the brain, and the computer. Addison-Wesley Longman Publishing Co., Inc., USA, 1987.

FIX, E., HODGES, J. L. Discriminatory Analysis, Nonparametric Discrimination: Consistency Properties. Technical Report 4, USAF School of Aviation Medicine, Randolph Field, 1951.

FIX, E., HODGES, J. L. Discriminatory Analysis. Nonparametric Discrimination: Consistency Properties. International Statistical Review / Revue Internationale de Statistique, 57(3), 238-247, 1989, https://doi.org/10.2307/1403797.

FREEMAN, W. J. Mass Action in the Nervous System. Elsevier Science & Technology Books, New York, 1975.

GAGE, P. A new algorithm for data compression. The C Users Journal, v. 12, n. 2, p. 23-38, 1994.

GERS, F. et al. Learning to Forget: Continual Prediction with LSTM. Neural computation. 12. 2451-71. 10.1162/089976600300015015, 2000.

GRAPE, S. et al. Determination of spent nuclear fuel parameters using modelled signatures from non-destructive assay and Random Forest regression. Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment. 969. 163979. 10.1016/j.nima.2020.163979, 2020.

GRAVES, A. et al. Speech recognition with deep recurrent neural networks, 2013 IEEE International Conference on Acoustics, Speech and Signal Processing, Vancouver, BC, Canada, pp. 6645-6649, doi: 10.1109/ICASSP.2013.6638947, 2013.

GUO, D. et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. arXiv preprint arXiv:2501.12948, 2025.

HAND, D. J., YU, K. Idiot's Bayes: Not So Stupid after All? International Statistical Review / Revue Internationale de Statistique, 69(3), 385-398, 2001, doi: 10.2307/1403452.

HARRIS, Z. Distributional Structure. Word, 10(2-3), 146-162, 1954.

HAYKIN, S. S. Neural Networks and Learning Machines. Third Edition, Pearson Education, Inc., McMaster University, Hamilton, 2009.

HAYKIN, S. S. Neural Networks: A Comprehensive Foundation. 2nd Edition, Prentice-Hall, Englewood Cliffs, NJ, 1999.

HE, K. et al. Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778), 2016.

HE, K. et al. Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. In Proceedings of the IEEE international conference on computer vision (pp. 1026-1034), 2015.

HENNER, K. An intuitive introduction to text embeddings. Stack Overflow Blog, 9 nov. 2023. Disponível em: https://stackoverflow.blog/2023/11/09/an-intuitive-introduction-to-text-embeddings/. Acesso em: 5 dez. 2024.

HESTENES, M. R., STIEFEL, E. Methods of conjugate gradients for solving linear systems. Journal of research of the National Bureau of Standards, 49, 409-435, 1952.

HILT, D. E., SEEGRIST, D. W. Ridge, a computer program for calculating ridge regression estimates. U.S. Department of Agriculture, National Agricultural Library, 1977, doi: 10.5962/bhl.title.68934.

HINKELMANN, K. Neural Networks. University of Applied Sciences Northwestern Switzerland, 2018.

HINTON, G. et al. Distilling the knowledge in a neural network. arXiv preprint arXiv:1503.02531, 2015.

HINTON, G., ROWEIS, S. Stochastic neighbor embedding. Neural Information Processing Systems, 2002.

HO, T. K. Random decision forests. Proceedings of 3rd International Conference on Document Analysis and Recognition, Montreal, QC, Canada, 1995, pp. 278-282 vol.1, doi: 10.1109/ICDAR.1995.598994.

HOCHREITER, S., SCHMIDHUBER, J. Long Short-term Memory. Neural computation. 9. 1735-80, 1997, doi: 10.1162/neco.1997.9.8.1735.

HOERL, A. E. Application of Ridge Analysis to Regression Problems. Chemical Engineering Progress, 58, 54-59, 1962.

HOERL, A. E., KENNARD, R. W. Ridge Regression: Applications to Nonorthogonal Problems. Technometrics. 12 (1): 69-82, 1970, doi: 10.2307/1267352.

HOPFIELD J. J. Neural networks and physical systems with emergent collective computational abilities. Proceedings of the National Academy of Sciences of the United States of America, 79(8), 2554-2558; 1982, doi: 10.1073/pnas.79.8.2554.

HOSSAIN, M. Z. et al. A comprehensive survey of deep learning for image captioning. ACM Computing Surveys (CsUR), 51(6), 1-36, 2019.

HUANG, S. T. et al. Tuning-free ridge estimators for high-dimensional generalized linear models. Computational Statistics & Data Analysis, 159, 107205, 2020.

HUANG, T. et al. Application of kernel ridge regression in predicting neutron-capture reaction cross-sections. Communications in Theoretical Physics. 74. 10.1088/1572-9494/ac763b, 2022.

IAEA. Safety of Nuclear Power Plants: Design. Specific Safety Requirements No. SSR-2/1 (Rev. 1), 2016.

JANG, K. B. et al. Risk analysis of nuclear power plant (NPP) operations by artificial intelligence (AI) in robot. Journal of Robotics and Control (JRC), v. 3, n. 2, p. 153-159, 2022.

JOULIN, A. et al. Bag of tricks for efficient text classification. arXiv:1607.01759, 2016.

JOYCE, J. P.; LAPINSKY, G. W. A history and overview of the safety parameter display system concept. IEEE Transactions on Nuclear Science, v. 30, n. 1, p. 744-749, 1983.

JURAFSKY, D., MARTIN, J. Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition, 2000.

KARPATHY, A. et al. Visualizing and understanding recurrent networks. arXiv preprint arXiv:1506.02078, 2015.

KAUFMAN, L., ROUSSEEUW, P. J. Partitioning Around Medoids (Program PAM), Wiley Series in Probability and Statistics, Hoboken, NJ, USA: John Wiley & Sons, Inc., pp. 68-125, 1990.

KAZEMNEJAD, A. Transformer Architecture: The Positional Encoding. Disponível em: https://kazemnejad.com/blog/transformer_architecture_positional_encoding/. 2019, Acesso em: 4 dez. 2024.

KIM, C. H. et al. Probabilistic neural networks for improved analyses with phenomenological models. arXiv preprint arXiv:2305.02623, 2023.

KINGMA, D. P., BA, J. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2014.

KOBAYASHI, S. et al. Weight decay induces low-rank attention layers. arXiv preprint arXiv:2410.23819, 2024.

KOHONEN, T. Self-organized formation of topologically correct feature maps. Biol. Cybern. 43, 59–69 (1982). https://doi.org/10.1007/BF00337288

KOLEN, J. F., KREMER, S. C. Gradient Flow in Recurrent Nets: The Difficulty of Learning Long-term Dependencies," in A Field Guide to Dynamical Recurrent Networks, IEEE, 2001, pp.237-243, doi: 10.1109/9780470544037.ch14.

KOO, Y. et al. Identification of Initial Events in Nuclear Power Plants Using Machine Learning Methods. In: Transactions of the Korean Nuclear Society Virtual Spring Meeting. 2020. p. 1-4.

KRIZHEVSKY, A. et al. ImageNet Classification with Deep Convolutional Neural Networks. Neural Information Processing Systems. 25. 10.1145/3065386, 2012.

LECUN, Y. et al. Deep learning. Nature 521, 436-444 (2015). https://doi.org/10.1038/nature14539.

LEWIS, H. W. et al. Risk assessment review group report to the US Nuclear Regulatory Commission IEEE Trans Nucl Sci, 26 (5), p. 4686-4690, 1979.

LI, C. et al. Deep speaker: an end-to-end neural speaker embedding system. arXiv preprint arXiv:1705.02304, 2017.

LI, J., LIU, H. Challenges of feature selection for big data analytics. IEEE Intelligent Systems, 32(2), 9-15, 2016.

LIN, J. et al. Predicting the Severity of Nuclear Power Plant Transients Using Nearest Neighbors Modeling Optimized by Genetic Algorithms on a Parallel Computer. Nuclear Technology, 111(1), 46-62. https://doi.org/10.13182/NT95-A35143, 1995.

LIU, Y. et al. Roberta: A robustly optimized bert pretraining approach. arXiv preprint arXiv:1907.11692, 364, 2019.

LIU, Z. et al. Kan: Kolmogorov-Arnold Networks. arXiv preprint arXiv:2404.19756, 2024a.

LIU, Z. et al. Kivi: A tuning-free asymmetric 2bit quantization for kv cache. arXiv preprint arXiv:2402.02750, 2024b.

LUNDBERG, S. LEE, S. I. A unified approach to interpreting model predictions. arXiv preprint arXiv:1705.07874, 2017.

LUONG, M. T. et al. Effective approaches to attention-based neural machine translation. arXiv preprint arXiv:1508.04025, 2015.

MA, J., YARATS, D. On the adequacy of untuned warmup for adaptive optimization. In Proceedings of the AAAI Conference on Artificial Intelligence (Vol. 35, No. 10, pp. 8828-8836), 2021.

MAAS, A. L. et al. Rectifier Nonlinearities Improve Neural Network Acoustic Models. In Proc. icml (Vol. 30, No. 1, p. 3), Computer Science Department, Stanford University, CA 94305 USA, 2013.

MACQUEEN, J. B. Some Methods for classification and Analysis of Multivariate Observations. Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability. Vol. 1. University of California Press. pp. 281-297, 1967.

MANNING, C. D. et al. Introduction to Information Retrieval. Cambridge University Press, Cambridge. https://doi.org/10.1017/CBO9780511809071, 2008.

MCCALLUM, A., NIGAM, K. A Comparison of Event Models for Naive Bayes Text Classification. Proceedings in Workshop on Learning for Text Categorization, AAAI'98, 41-48, 1998.

MICIKEVICIUS, P. et al. Mixed precision training. arXiv preprint arXiv:1710.03740, 2017.

MIKOLOV, T. et al. Efficient estimation of word representations in vector space. arXiv:1301.3781, 3781, 2013.

NAIMI, A. et al. Fault Detection and Isolation of a Pressurized Water Reactor Based on Neural Network and K-Nearest Neighbor, in IEEE Access, vol. 10, pp. 17113-17121, doi: 10.1109/ACCESS.2022.3149772, 2022.

NICOLAU, A. S. et al. Accident diagnosis system based on real-time decision tree expert system, AIP Conference Proceedings, 1836, 0200017, 2017.

NICOLAU, A. et al. Deep neural networks for estimation of temperature values for thermal ageing evaluation of nuclear power plant equipment. Progress in Nuclear Energy, 156(3):104542, 2023, doi: 10.1016/j.pnucene.2022.104542.

NRC, NUREG-0880, Revision 1 For Comment, Safety Goals for Nuclear Power Plant Operation, 1983.

NRC, NUREG/CR 6953 Vol 1, Review of NUREG 0654, Supplement 3, Criteria for Protective Actions Recommendations for Severe Accidents, Guidance for Protective Action Strategy, 2007.

OH, S. W. et al. Development of an AI-based remaining trip time prediction system for nuclear power plants, Nuclear Engineering and Technology, vol. 56, no. 8, pp. 3167-3179, 2024, doi: 10.1016/j.net.2024.03.017.

OH, S. W. et al. Remaining Trip Time Prediction Using Light Gradient Boosting Machine in Nuclear Power Plants. In: 2022 International Conference on Computational Science and Computational Intelligence (CSCI). IEEE, 2022. p. 302-305.

OMOHUNDRO, S. M. Five balltree construction algorithms, 1989.

ONG, A. K. et al. Utilization of random forest classifier and artificial neural network for predicting the acceptance of reopening decommissioned nuclear power plant. Annals of Nuclear Energy. 175. 109188. 10.1016/j.anucene.2022.109188, 2022.

PEARSON, K. On Lines and Planes of Closest Fit to Systems of Points in Space. Philosophical Magazine. 2 (11): 559-572, 1901, doi:10.1080/14786440109462720.

PENNINGTON, J. et al. GloVe: Global Vectors for Word Representation. Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), Doha, Qatar, 25-29 October 2014, 1532-1543, 2014.

QI, B. et al. Multimodal learning using large language models to improve transient identification of nuclear power plants. Progress in Nuclear Energy, 177, 105421. https://doi.org/10.1016/j.pnucene.2024.105421, 2024.

RADFORD, A. et al. Improving Language Understanding by Generative Pre-Training, 2018.

RADFORD, A. et al. Language models are unsupervised multitask learners. OpenAI blog, v. 1, n. 8, p. 9, 2019.

RAJARAMAN, A., ULLMAN J. D. Data Mining. In: Mining of Massive Datasets. Cambridge University Press; 2011:1-17.

RAMÍREZ-GALLEGO, S. et al. An information theory-based feature selection framework for big data under apache spark. IEEE Transactions on Systems, Man, and Cybernetics: Systems, 48(9), 1441-1453, 2017.

RANKIN, D., JIANG, J. Predictive Trip Detection for Nuclear Power Plants, IEEE Transactions on Nuclear Science, vol. 63, no. 4, pp. 2352-2362, 2016, doi: 10.1109/TNS.2016.2582467.

RENNIE, J. D. M. et al. Tackling the Poor Assumptions of Naive Bayes Text Classifiers. Artificial Intelligence Laboratory; Massachusetts Institute of Technology; Cambridge, MA 02139, 2003.

RIBEIRO, M. T. et al. Why should i trust you? Explaining the predictions of any classifier. In Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining (pp. 1135-1144), 2016.

SAEED, H. A. et al. Novel fault diagnosis scheme utilizing deep learning networks. Progress in Nuclear Energy, v. 118, p. 103066, 2020.

SANH, V. et al. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. arXiv preprint arXiv:1910.01108, 2019.

SANTOS, M. C. et al. A multiple-architecture deep learning approach for nuclear power plants accidents classification including anomaly detection and “don’t know” response. ANNALS OF NUCLEAR ENERGY, v. 162, p. 108521, 2021.

SCHIRRU, R., PEREIRA, C.M.N.A. A Real-Time Artificially Intelligent Monitoring System for Nuclear Power Plants Operators Support. Real-Time Systems 27, 71-83, 2004. https://doi.org/10.1023/B:TIME.0000019127.50572.9b.

SCHMIDHUBER, J. Annotated History of Modern AI and Deep Learning. arXiv:2212.11279, 2022.

SCHNEIDER, M. et al. The World Nuclear Industry Status Report 2024. Paris: Mycle Schneider Consulting, 2024. Disponível em: https://www.worldnuclearreport.org/IMG/pdf/wnisr2024-v2.pdf. Acesso em: 05 dez. 2024.

SCHULZ, H., BEHNKE, S. Deep Learning. Künstl Intell 26, 357-363 (2012). https://doi.org/10.1007/s13218-012-0198-z.

SENNRICH, R. et al. Neural machine translation of rare words with subword units. arXiv preprint arXiv:1508.07909, 2015.

SHAHID, N. Comparison of hierarchical clustering and neural network clustering: an analysis on precision dominance. Sci Rep 13, 5661 (2023). https://doi.org/10.1038/s41598-023-32790-3.

SHAZEER, N. Fast transformer decoding: One write-head is all you need. arXiv:1911.02150, 2019.

SINGH, B. et al. Layer-Specific Adaptive Learning Rates for Deep Networks. 2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA), 364-368. (2015). https://doi.org/10.1109/ICMLA.2015.113.

SPANGHER, L. et al. Autoregressive Transformers for Disruption Prediction in Nuclear Fusion Plasmas. arXiv preprint arXiv:2401.00051, 2023.

STERRATT, D. et al. Principles of Computational Modelling in Neuroscience, Cambridge, U.K.: Cambridge University Press, 2011.

SUTSKEVER, I. et al. Sequence to Sequence Learning with Neural Networks. arXiv:1409.3215, 2014.

SUYKENS, J., VANDEWALLE, J. Least Squares Support Vector Machine Classifiers. Neural Processing Letters 9, 293-300. https://doi.org/10.1023/A:1018628609742, 1999.

TIELEMAN, T., HINTON, G. Lecture 6.5-rmsprop: divide the gradient by a running average of its recent magnitude. COURSERA: Neural networks for machine learning, 4(2), 26-31, 2012.

TURNER, R. E. An introduction to transformers. arXiv:2304.10557, 2023.

VAN DER MAATEN, L., HINTON, G. Visualizing Data Using t-SNE. Journal of Machine Learning Research. 9: 2579-2605, 2008.

VASWANI, A. et al. Attention Is All You Need. arXiv:1706.03762, 2017.

WANG, C. et al. Neural machine translation with byte-level subwords. In: Proceedings of the AAAI conference on artificial intelligence. 2020. p. 9154-9160.

WOLF, T. et al. Huggingface's transformers: State-of-the-art natural language processing. arXiv preprint arXiv:1910.03771, 2019.

WOODS, W. A. Important issues in knowledge representation. Proceedings of the Institute of Electrical and Electronics Engineers, vol. 74, pp. 1322-1334, 1986.

WU, X. H. Studies of different kernel functions in nuclear mass predictions with kernel ridge regression. Frontiers in Physics. 11. 10.3389/fphy.2023.1061042, 2023.

WU, X. H., PAN, C. Nuclear mass predictions with anisotropic kernel ridge regression. Physical Review C, 110(3), 034322, 2024.

WU, X. H., ZHAO, P. Predicting nuclear masses with the kernel ridge regression. Physical Review C. 101. 10.1103/PhysRevC.101.051301, 2020.

XIAN, M. et al. A Knowledge-Informed Large Language Model Framework for US Nuclear Power Plant Shutdown Initiating Event Classification for Probabilistic Risk Assessment. arXiv preprint arXiv:2410.00929, 2024.

XIAO, X. et al. A Text Intelligence-Based Approach for Automatic Generation of Fault Trees in Nuclear Power Plants. Proceedings of the 2024 31st International Conference on Nuclear Engineering. Volume 10: Risk Assessments and Management; Computer Code Verification and Validation; Nuclear Education and Public Acceptance. Prague, Czech Republic. August 4-8, 2024. V010T12A004. ASME. https://doi.org/10.1115/ICONE31-134226.

YANG, J. et al. Nuclear Power Plant Accident Diagnosis Algorithm Including Novelty Detection Function Using LSTM. In: Ahram T. (eds) Advances in Artificial Intelligence, Software and Systems Engineering. AHFE 2019. Advances in Intelligent Systems and Computing, v. 965, 2020.

YATES, R. B., NETO, B. R. Recuperação de informação: conceitos e tecnologia das máquinas de busca. 2. ed. Porto Alegre: Bookman, 2013.

YI, S. et al. Robust transformer-based anomaly detection for nuclear power data using maximum correntropy criterion. Nuclear Engineering and Technology. 56. 10.1016/j.net.2023.11.033, 2023.

YOO, K. H. et al. Smart support system for diagnosing severe accidents in nuclear power plants. Nuclear Engineering and Technology, v. 50, n. 4, p. 562-569, 2018.

ZHA, B. et al. Deep transformer networks for time series classification: the NPP safety case. arXiv preprint arXiv:2104.05448, 2021.

ZHANG, C. et al. Fault Diagnosis of Nuclear Power Plant Based on Sparrow Search Algorithm Optimized CNN-LSTM Neural Network. Energies. 16(6):2934, 2023a, doi: 10.3390/en16062934.

ZHANG, D. et al. Interpretable Time Series Data Prediction for Severe Nuclear Accidents. In: 2023 3rd International Conference on New Energy and Power Engineering (ICNEPE). IEEE, 2023b. p. 574-580.

ZHANG, J. et al. Why are adaptive methods good for attention models? Advances in Neural Information Processing Systems, 33, 15383-15393, 2020.

ZHANG, S. et al. Opt: Open pre-trained transformer language models. arXiv preprint arXiv:2205.01068, 2022.

ZHANG. T. Solving large scale linear prediction problems using stochastic gradient descent algorithms. In Proceedings of the twenty-first international conference on Machine learning (ICML '04). Association for Computing Machinery, New York, NY, USA, 116. https://doi.org/10.1145/1015330.1015332, 2004.

ZHOU, G. et al. A Novel Transformer-Based Anomaly Detection Model for the Reactor Coolant Pump in Nuclear Power Plants. Science and Technology of Nuclear Installations. 2024. 10.1155/stni/9455897, 2024.

ZUHRI, Z. M. K. et al. MLKV: Multi-Layer Key-Value Heads for Memory Efficient Transformer Decoding. arXiv:2406.09297, 2024.