UNIVERSIDADE FEDERAL DE OURO PRETO
INSTITUTO DE CIÊNCIAS EXATAS E BIOLÓGICAS
DEPARTAMENTO DE COMPUTAÇÃO

MATHEUS PEIXOTO RIBEIRO VIEIRA

PLASTICIDADE E RIGIDEZ EM MODELOS DE EMBEDDINGS GLOBAIS E MONOLÍNGUES PARA O PORTUGUÊS BRASILEIRO

Ouro Preto
2026

MATHEUS PEIXOTO RIBEIRO VIEIRA

PLASTICIDADE E RIGIDEZ EM MODELOS DE EMBEDDINGS GLOBAIS E MONOLÍNGUES PARA O PORTUGUÊS BRASILEIRO

Monografia apresentada ao Curso de Ciência da Computação da Universidade Federal de Ouro Preto como parte dos requisitos necessários para a obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Pedro Henrique Lopes Silva

Ouro Preto
2026

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE OURO PRETO
REITORIA
INSTITUTO DE CIENCIAS EXATAS E BIOLOGICAS
DEPARTAMENTO DE COMPUTACAO

FOLHA DE APROVAÇÃO

Matheus Peixoto Ribeiro Vieira

Plasticidade e Rigidez em Modelos de *Embeddings* Globais e Monolíngues para o Português Brasileiro

Monografia apresentada ao Curso de Ciência da Computação da Universidade Federal de Ouro Preto como requisito parcial para obtenção do título de Bacharel em Ciência da Computação

Aprovada em 27 de Fevereiro de 2026

Membros da banca

Pedro Henrique Lopes Silva (Orientador) - Doutor - Universidade Federal de Ouro Preto
Arthur Negrão de Faria Martins da Costa (Examinador) - Bacharel - Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Ouro Preto
Ederson Naves Fernandes Gonçalves Junior (Examinador) - Bacharel - Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Ouro Preto

Pedro Henrique Lopes Silva, orientador do trabalho, aprovou a versão final e autorizou seu depósito na Biblioteca Digital de Trabalhos de Conclusão de Curso da UFOP em 27/02/2026

Documento assinado eletronicamente por **Pedro Henrique Lopes Silva**, **PROFESSOR DE MAGISTERIO SUPERIOR**, em 27/02/2026, às 14:05, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site http://sei.ufop.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0 , informando o código verificador **1062474** e o código CRC **8089A69B**.

R. Diogo de Vasconcelos, 122, - Bairro Pilar Ouro Preto/MG, CEP 35402-163
Telefone: 3135591692 - www.ufop.br

Dedico este trabalho aos meus pais, Cláudia e Omar, e à minha irmã, Camila.

Agradecimentos

Agradeço aos meus pais, Cláudia e Omar, por todo o apoio e incentivo que me permitiram superar os mais diversos desafios desta jornada. Vocês foram a base sólida sobre a qual todo este trabalho pôde ser construído, sempre próximos e dispostos a me apoiar nos momentos em que mais tive dúvidas e incertezas.

Agradeço à minha irmã, Camila, por me lembrar de que existe vida além das tarefas e por me mostrar a importância de criar memórias que serão guardadas e revisitadas por toda a vida.

Agradeço a meus primos Caio Henrique e Luís Felipe por me mostrarem os caminhos da computação e o passos necessários para iniciar minha trajetória. Agradeço imensamente aos meus primos Thales e Thaciane, que me ensinaram, e muito, a como enfrentar desafios, cultivar relacionamentos e, sobretudo, buscar equilíbrio.

Agradeço à Sophia por me mostrar que é possível levar a vida com mais leveza e por me ensinar que não é necessário ter o controle de tudo a todo momento. Ademais, me mostrou como é importante viver um dia de cada vez e como, de fato, estar presente no momento.

Agradeço aos amigos que conheci ao longo desta jornada e que desejo levar para a vida. Muito obrigado pelas incontáveis horas de conversa, risadas, aprendizados e reflexões sobre a vida e o futuro, seja em casa, na sala de aula, no CSILab, no GAID, no RU ou em qualquer outro espaço, físico ou virtual. Não consigo imaginar o quanto essa caminhada teria sido mais desafiadora sem a presença de cada um de vocês. Por isso, deixo meus mais sinceros agradecimentos a Bruno Braga, Felipe Braz, Gabriel Saldanha, Lucas Chagas, Lucas Silva, Nicolas Mendes, Pedro Henrique, Pedro Morais e Vitor Oliveira.

Agradeço ao meu orientador, Prof. Dr. Pedro Silva, por todo o suporte e ensinamento ao longo destes anos, e por me permitir ingressar em uma área de estudos pela qual sempre tive curiosidade, mas que antes parecia difícil, distante e nebulosa. Hoje, graças à sua orientação, essa área se tornou mais leve, clara e inspiradora para o futuro.

Por fim, gostaria de agradecer a todos os professores do DECOM que fizeram parte da minha trajetória, tanto àqueles com quem tive o privilégio de aprender em sala de aula, quanto àqueles que conheci e com quem conversei apenas nos corredores do ICEB, pela UFOP e em eventos.

"É justamente a possibilidade de realizar um sonho que torna a vida interessante"

Coelho (2001)

Resumo

Este trabalho investiga comparativamente modelos de *embeddings* globais e monolíngues aplicados ao português brasileiro, analisando o equilíbrio entre plasticidade e rigidez das representações semânticas sob diferentes regimes de uso. São avaliadas sete famílias de modelos em quatro tarefas de processamento de linguagem natural (classificação, clusterização, inferência textual e similaridade semântica textual) considerando tanto o regime de *linear probing* quanto a adaptação supervisionada via *fine-tuning* eficiente de parâmetros com LoRA. Os resultados indicam que, embora modelos globais apresentem desempenho competitivo em *linear probing*, modelos monolíngues tendem a demonstrar maior estabilidade e ganhos mais consistentes após a adaptação, especialmente em tarefas sensíveis à geometria do espaço de *embeddings*, como STS. Adicionalmente, análises quantitativas e qualitativas de *tokenização* revelam que somente vocabulários especializados não garantem maior eficiência ou desempenho, evidenciando um dilema entre alinhamento morfológico, robustez a empréstimos linguísticos e compactação das representações. As descobertas contribuem para uma compreensão mais profunda dos *trade-offs* envolvidos na escolha e adaptação de modelos de *embeddings* para aplicações em português brasileiro, oferecendo subsídios práticos para decisões em cenários reais de PLN.

Palavras-chave: Embeddings, tokens, transformer, modelo multilíngue, modelo monolíngue, plasticidade, rigidez.

Abstract

This work presents a comparative investigation of global and monolingual embedding models applied to Brazilian Portuguese, analyzing the balance between plasticity and rigidity of semantic representations under different usage regimes. Seven families of models are evaluated across four natural language processing tasks (classification, clustering, natural language inference and semantic textual similarity) considering both linear probing and supervised adaptation via parameter-efficient fine-tuning with LoRA. The results indicate that, although global models achieve competitive performance under linear probing, monolingual models tend to exhibit greater stability and more consistent gains after adaptation, particularly in tasks sensitive to the geometry of the embedding space, such as STS. In addition, quantitative and qualitative analyses of tokenization reveal that specialized vocabularies do not inherently guarantee higher efficiency or performance, highlighting a trade-off between morphological alignment, robustness to lexical borrowings, and representation compactness. These findings contribute to a deeper understanding of the trade-offs involved in selecting and adapting embedding models for Brazilian Portuguese, providing practical insights for decision-making in real-world NLP applications.

Keywords: Embeddings, tokens, transformer, multilingual model, moonolingual model, plasticity, rigidity

Lista de Figuras

Figura 2.1 – Exemplo de um perceptron proposto por Rosenblatt (1958). 8
Figura 2.2 – MLP com uma camada de entrada, duas ocultas e uma de saída 9
Figura 2.3 – Função ReLU 11
Figura 2.4 – Representação da computação do erro de uma rede neural. W indica os pesos da camada, b o bias, a o resultado após ativação, z a soma ponderada antes da ativação e Y o valor real. 15
Figura 2.5 – Fluxo de computação de uma rede neural camada a camada até a obtenção do erro 16
Figura 2.6 – Espaço de *embeddings* arbitrário para as palavras *king* (rei), *queen* (rainha), *man* (homem) e *woman* (mulher), mostrando que as diferenças vetoriais dos termos masculinos em relação aos femininos são semelhantes. 19
Figura 2.7 – Arquitetura do modelo de um transformer. 21
Figura 2.8 – Exemplo de clusterização via K-médias com $k = 3$ e exibição dos centroides. 27
Figura 5.1 – Mapa de calor dos Δ's de variação da avaliação *linear-probing* e *fine-tuning* 43
Figura 5.2 – Delta de resultados entre *linear-probing* e *fine-tuning* entre os modelos e métricas avaliados para a tarefa de STS, mostrando uma grande degradação dos modelos multilíngues e alguns monolíngues. 47
Figura 5.3 – Deltas de acurácia para os diferentes conjuntos de dados na tarefa de classificação 48
Figura 5.4 – Deltas de acurácia para os diferentes conjuntos de dados na tarefa de NLI. . 48
Figura 5.5 – Deltas de acurácia para os diferentes conjuntos de dados na tarefa de *clusterização*. 49

Lista de Tabelas

Tabela 2.1 – Resultado da *tokenização* da frase "É justamente a possibilidade de realizar um sonho que torna a vida interessante"para algoritmos BPE, WordPiece e SentencePiece 18
Tabela 4.1 – Resumo dos modelos utilizados 36
Tabela 4.2 – Tarefas, métricas e conjunto de dados explorados 38
Tabela 5.1 – Resultados da tarefa de classificação (acurácia) sobre regimes de *linear probing* e *fine tuning*. Melhores resultados por conjunto de dados/regimes estão marcados em negrito. 44
Tabela 5.2 – Resultado do *clustering* (AMI) sobre regimes de *linear-probing* e *fine-tuning*. Os melhores resultados por conjunto de dados/regimes estão marcados em negrito. 45
Tabela 5.3 – Resultados do NLI (Acurácia) no ASSIN2 sobre regimes de *linear-probing* e *fine-tuning*. Melhores valores estão em negrito 46
Tabela 5.4 – Resultados do STS (Pearson e Spearman) sobre regimes de *linear-probing* e *fine-tuning*. Os melhores resultados por *dataset/regime estão em negrito.* . . 46
Tabela 5.5 – Tempo médio de execução, em segundos, em cada conjunto de dados. 50
Tabela 5.6 – Número médio de *tokens* por palavra de diferentes modelos ao longo dos conjuntos de dados. Valores em negrito indicam as menores médias por coluna. 51
Tabela 5.7 – Tokenização qualitativa de diferentes palavras e expressões do conjunto Em Ambiente Real, evidenciando a estrutura morfológica de termos em PT-BR. Contrariamente às expectativas, os tokenizadores do Gemma e do ST MiniLM apresentam forte alinhamento morfológico, enquanto o vocabulário monolíngue do BERTimbau falha ao lidar com empréstimos comuns do inglês. 52

Lista de Abreviaturas e Siglas

AMI Adjusted Mutual Information. viii, 26, 27, 38, 45, 47, 50

BCE *Binary Cross-Entropy Loss*. 13

BERT Bidirectional Encoder Representations from Transformers. xii, 1, 20, 31, 32, 34, 36, 37, 48, 53

BoW Bag-of-Words. 19

BPE Byte Pair Encoding. viii, 17, 18, 51

CBOW Continuous Bag of Words. 20

GloVe Global Vectors for Word Representation. 20

LDA Alocação Latente de Dirichlet. 38

LLM Large Language Model. 1, 23

LoRA Low Rank Adaptation. xii, 2, 3, 24, 25, 32, 40–42, 53

MAE *Mean Absolute Error*. 12

mBERT BERT multilíngue. 31–33

MLP *Multilayer perceptron*. vii, 9, 22, 25

MSE *Mean Absolute Error*. 13

NLI Natural Language Inferencing. vii, viii, xiii, 2, 3, 27, 33, 38, 40, 45–48, 50, 53

PEFT Parameter-Efficient Fine-Tuning. xii, 3, 24, 25, 40, 42, 53

PLN Processamento de Linguagem Natural. xii, 1, 2, 16, 19, 23, 35, 54

PP Pergunta de Pesquisa. xiii, 2, 42, 47, 49

ReLU *Rectified Linear Unit*. vii, xii, 11

RNN Recurrent Neural Networks. 21

SGD *Stochastic Gradient Descent*. 14

ST MiniLM Sentence-Transformer MiniLM. 36, 37, 42, 44–46, 49–53

STS *Semantic Textual Similarity*. vii, xiii, 2, 3, 28, 37, 38, 40, 45, 47, 48, 50, 52–54

TD-IDF Term Frequency–Inverse Document Frequency. 19

Lista de Símbolos

$\sum$	Somatório
ϕ	Função de ativação
∇	Vetor gradiente
ϵ	Função de erro
η	Taxa de aprendizado

Sumário

1 Introdução 1
1.1 Justificativa 2
1.2 Objetivos 3
1.3 Organização do Trabalho 3
2 Referencial Teórico 5
2.1 Inteligência Artificial 5
2.2 Aprendizado de Máquina 6
2.3 Redes Neurais 7
2.3.1 Perceptron 7
2.3.2 Perceptron Multicamadas 8
2.4 Funções de ativação 10
2.4.1 *Sigmoid* 10
2.4.2 *Rectified Linear Unit* 11
2.4.3 *Softmax* 11
2.5 Função de erro 12
2.5.1 Erro Médio Absoluto 12
2.5.2 Erro Médio Quadrático 13
2.5.3 Erro de Entropia Cruzada 13
2.5.4 Erro de Entropia Cruzada Binária 13
2.6 Aprendizado Profundo 13
2.7 Processamento de Linguagem Natural 16
2.7.1 Representação textual 16
2.7.1.1 Tokens 17
2.7.1.2 *Embeddings* 18
2.7.1.2.1 Métodos Baseados em Contagem 19
2.7.1.2.2 *Embeddings* estáticos 20
2.7.1.2.3 *Embeddings* contextualizados - BERT 20
2.7.2 Arquitetura baseada em Transformers 21
2.7.3 Modelos de linguagem 23
2.7.3.1 Grandes Modelos de Linguagem 23
2.8 Ajuste fino 24
2.8.1 Parameter-Efficient Fine-Tuning 24
2.8.1.1 Low Rank Adaptation (LoRA) 24
2.9 Tarefas e avaliação 25
2.9.1 Classificação 25
2.9.2 Clusterização 26

2.9.3 Natural Language Inferencing . . . 27
2.9.4 *Semantic Textual Similarity* . . . 28
2.10 Regimes de avaliação . . . 29
3 Revisão Bibliográfica . . . 31
3.1 Trabalhos Relacionados . . . 31
3.2 *Embeddings* Multilíngues e Modelos Fundacionais . . . 33
3.3 Modelos Monolíngues para o Português . . . 34
3.4 Discussão dos trabalhos . . . 34
4 Materiais e Métodos . . . 36
4.1 Modelos . . . 36
4.2 Conjunto de dados . . . 38
4.3 Preparação dos dados . . . 39
4.4 Regimes de avaliação . . . 40
4.4.1 Linear Probing . . . 40
4.4.2 *Fine-tuning* . . . 40
4.5 Análise da qualidade dos *tokens* . . . 41
5 Resultados . . . 42
5.1 Configuração dos experimentos . . . 42
5.2 Resultados obtidos . . . 42
5.2.1 Pergunta de Pesquisa (PP)1: *Linear-Probing vs Fine-tuning* . . . 42
5.2.2 PP2: A dicotomia da adaptação . . . 47
5.2.3 PP3: Qual modelo escolher? . . . 49
5.2.4 Análise do balanço entre plasticidade e rigidez . . . 50
6 Considerações Finais . . . 53
6.1 Conclusão . . . 53
6.2 Trabalhos Futuros . . . 54

Referências . . . 55

Apêndices 61
APÊNDICE A Conjunto Em Ambiente Real . . . 62

1 Introdução

TURING (1950) propôs um desafio chamado de Jogo da Imitação, que mais tarde viria a ser conhecido como Teste de Turing. Nesse experimento um avaliador humano julga a conversa entre um computador e um humano. A máquina passa no teste caso ela consiga convencer o avaliador de que ela é humano a partir da forma como seus textos são elaborados, independente se estão corretos.

Em 2014, 64 anos após a publicação de Turing, pela primeira vez uma máquina conseguiu passar no teste. Na Universidade de Reading, foi realizado um evento com 30 juízes, e consideraram que a máquina passaria no teste se, pelo menos, 30% dos juízes se convencessem de que a máquina era humana. Dez avaliadores aceitaram que uma máquina era real e, por isso, ela passou no teste (ROHR, 2014).

Durante muitos anos, as técnicas de Processamento de Linguagem Natural (PLN) eram baseadas em métodos estáticos e modelos sequenciais que exigiam um forte engenharia de atributos, possuindo representações numéricas de palavras de forma simplória e com baixa eficiência. E, para fazer o processamento de texto, era comum o uso de redes neurais recorrentes, que processam palavra por palavra e possuem dificuldades em capturar contextos distantes, possuindo um uso bem limitado (LI et al., 2021).

Todavia, em 2017, Vaswani et al. (2017) propuseram as arquiteturas de *transformers* para a tradução de texto, e se tornou o estado da arte para tarefas de PLN (SAJUN; ZUALKERNAN; SANKALPA, 2024). Tal avanço permitiu o surgimento de modelos capazes de analisar textos com mais qualidades, como o Bidirectional Encoder Representations from Transformers (BERT) (DEVLIN et al., 2019), e gerar textos, como o GPT (RADFORD et al., 2018).

Cinco anos depois, em 2022, o mundo se surpreendeu com o avanço dos modelos geradores de texto quando a Open Ai lançou o ChatGPT, um *chatbot* com alta capacidade de escrita que se assemelhava muito a um ser humano escrevendo e possuindo capacidades impressionantes em diversas áreas do conhecimento, como jurídico, história, etc (JOHNSON, 2022). Em dois meses de lançamento atingiu 100 milhões de usuários, se tornando o sistema de maior crescimento já registrado (REUTERS, 2023).

O modelo GPT-3 foi o grande responsável pela popularização do ChatGPT e por trazer ao grande público o termo Large Language Model (LLM) ao utilizar 175 bilhões de parâmetros (SAJUN; ZUALKERNAN; SANKALPA, 2024). Mesmo com suas grandes habilidades em diversas tarefas de PLN, como classificação, tradução e geração de texto, estudos como os de Lai et al. (2023) surgiram mostrando como o modelo possui melhores capacidades para lidar com o inglês do que em outros idiomas. Entre os principais motivos está a grande quantidade de textos em língua inglesa utilizados para a realização do seu treinamento.

Mais recentemente, outros modelos surgiram seguindo o mesmo caminho, utilizando grandes conjuntos de dados de treinamento em diferentes línguas, formando *datasets* e modelos multilíngues. Entre esses modelos, pode-se destacar o Gemma (TEAM, 2024) e Qwen (ZHANG et al., 2025), que, assim como o GPT, conseguem obter ótimos resultados para uma vasta gama de desafios para os quais não foram treinados, ou que foram levemente adaptados.

Saindo dessa generalização de línguas, surgiram modelos focados em idiomas únicos. Para o português brasileiro, modelos monolíngues como o BERTimbau (SOUZA; NOGUEIRA; LOTUFO, 2020), embora treinados com arquiteturas baseadas nas primeiras versões dos transformers, utilizaram exclusivamente dados brasileiros. Com o seu sucesso, uma suposição foi muito consolidada na comunidade de PLN para o português: a de que vocabulários especializados e monolíngues são inerentemente superiores para alcançar os melhores resultados possíveis. Entretanto, com o avanço das arquiteturas, essa suposição pode ser questionada, permanecendo em aberto a possibilidade de que modelos globais levemente adaptados consigam superar modelos locais profundamente especializados. Assim, levanta-se uma questão ainda não tão explorada: *quão eficazes são adaptações leves e como é o seu comportamento em diferentes tarefas de downstream? Permanecem estáveis ou sofrem oscilações consideráveis?*.

Dessa forma, este trabalho busca explorar tal questionamento por meio de um estudo comparativo. O desempenho de sete famílias distintas de *sentence embeddings* é investigado em quatro tarefas focadas no português brasileiro: Classificação, *clusterização*, inferência textual (Natural Language Inferencing (NLI)) e similaridade semântica (*Semantic Textual Similarity* (STS)). Todas as tarefas são executadas para os modelos em dois regimes distintos: (i) *linear-probing*, para medir a qualidade das características dos modelos, e (ii) por adaptação leve, a fim de mensurar o impacto do *fine-tuning* supervisionado.

Para guiar a investigação, três Pergunta de Pesquisa (PP) são feitas:

PP1: *Como modelos de embeddings monolíngues e globais se comparam em português sob os regimes de linear probing e LoRA ao longo de tarefas diversas*?

PP2: *A adaptação dos modelos às suas respectivas tarefas sempre resulta em melhorias significativas de desempenho, ou existem trade-offs de desempenho entre diferentes tarefas?*

PP3: *Qual família de embeddings oferece o melhor equilíbrio entre desempenho e robustez nas tarefas avaliadas em português?*

1.1 Justificativa

Apesar dos avanços recentes em modelos de *embeddings* globais e monolíngues, ainda não há um consenso claro sobre como essas abordagens se comportam no português brasileiro quando avaliadas sob diferentes regimes de uso, como a exploração direta das representações e a adaptação supervisionada. Dessa forma, faz-se necessário realizar estudos sobre a compreensão

dos efeitos da especialização dos modelos em tarefas distintas e com diferentes exigências semânticas.

Além disso, com a possibilidade de treinar grandes modelos com recursos limitados ao utilizar técnicas Parameter-Efficient Fine-Tuning (PEFT) como LoRA, pode-se levantar questionamentos sobre possíveis *trade-offs* entre especialização e preservação das propriedades semânticas dos *embeddings*, sobretudo em tarefas sensíveis à geometria do espaço vetorial. A falta dessa análise integrada pode levar a decisões não ótimas na seleção e adaptação de modelos para aplicações reais.

Dessa forma, este trabalho se justifica pela necessidade de uma avaliação comparativa abrangente, focada no português brasileiro, que investigue o desempenho, estabilidade e a eficiência dos modelos, contribuindo para responder às perguntas de pesquisa propostas e oferecendo recursos para a escolha e adaptação de *embeddings* em diferentes cenários de uso.

1.2 Objetivos

O presente trabalho tem como objetivo principal analisar comparativamente o desempenho de modelos de *embeddings* globais e monolíngues para o português brasileiro, avaliando a qualidade do modelo original em um regime de *linear-probing* e sua versão especializada obtida por *fine-tuning* leve em diferentes tarefas e conjuntos de dados.

Para alcançar esse objetivo, os seguintes objetivos específicos foram definidos:

- Avaliar a qualidade das representações geradas por modelos globais e monolíngues em português brasileiro por meio do regime de *linear-probing*;
- Investigar o impacto do *fine-tuning* eficiente de parâmetros, utilizando técnicas de PEFT, no desempenho dos modelos em tarefas *downstream*;
- Comparar o comportamento dos modelos em diferentes tarefas, incluindo classificação, clusterização, NLI e STS;
- Identificar possíveis *trade-offs* entre adaptação e desempenho;
- Analisar o alinhamento dos modelos com a língua portuguesa.

1.3 Organização do Trabalho

Este trabalho está estruturado de forma a introduzir, inicialmente, os conceitos fundamentais para sua compreensão. O Capítulo 2 apresenta uma revisão bibliográfica que contextualiza os principais temas abordados, como inteligência artificial e modelos linguísticos. Os trabalhos relacionados são apresentados no Capítulo 3. Em seguida, o Capítulo 4 descreve detalhadamente

a metodologia empregada na condução dos experimentos, com foco no alcance dos objetivos geral e específicos. Os resultados obtidos são apresentados e discutidos no Capítulo 5. Por fim, o Capítulo 6 reúne as considerações finais do estudo e propõe direções para trabalhos futuros.

2 Referencial Teórico

Para compreender o comportamento e o desempenho de modelos de *embeddings* globais e monolíngues no português brasileiro, é fundamental estabelecer uma base teórica sólida sobre os principais conceitos envolvidos no processamento de linguagem natural. Assim, esta seção apresenta inicialmente os fundamentos de inteligência artificial e aprendizado de máquina, que sustentam o desenvolvimento de modelos modernos, e avança para o estudo de redes neurais profundas e arquiteturas baseadas em Transformers.

Em seguida, são discutidos os mecanismos de representação textual, com ênfase em *tokenização*, *embeddings* e espaços vetoriais semânticos, bem como os paradigmas de pré-treinamento e adaptação de modelos por meio de técnicas de *fine-tuning* e ajuste eficiente de parâmetros. Essa fundamentação permite contextualizar os desafios associados à especialização linguística, à transferência de conhecimento entre línguas e ao equilíbrio entre plasticidade e rigidez dos modelos, aspectos centrais para a análise desenvolvida ao longo deste trabalho.

2.1 Inteligência Artificial

O termo inteligência artificial não possui uma definição única e formal, mas é compreendido como o campo de estudo voltado para o desenvolvimento de sistemas computacionais capazes de realizar tarefas que, para os seres humanos, exigiriam algum grau de inteligência (RUSSELL; NORVIG, 2021).

Para a execução dessas tarefas, tais sistemas contam com os chamados agentes, entidades que percebem o mundo ao seu redor e são capazes de tomar decisões com base nas informações que obtêm a partir de sensores, como radares, sonares, etc. (RUSSELL; NORVIG, 2021).

Inicialmente, os primeiros avanços da inteligência artificial ocorreram na solução de problemas considerados intelectualmente complexos para seres humanos, mas que podiam ser formalizados de maneira matemática. Por outro lado, problemas aparentemente simples e intuitivos para nós se mostraram extremamente desafiadores para sistemas computacionais, uma vez que podem necessitar de conhecimentos externos e contextuais aprendidos no cotidiano. Esse tipo de conhecimento pode ser extremamente complexo de formalizar ou incorporar em um sistema computacional. Como exemplo, há um sistema que reconhece palavras faladas, mesmo com um sotaque (GOODFELLOW; BENGIO; COURVILLE, 2016).

Em 1959, Arthur L. Samuel propôs um agente capaz de jogar damas enquanto aprendia com suas jogadas a fim de melhorar e vencer jogos (SAMUEL, 1959). Já em 1997, o supercomputador da IBM, o Deep Blue II, derrotou Garry Kasparov, na época, atual campeão mundial de xadrez, com uma busca em árvore e a utilização de créditos para a seleção das melhores jogadas

(CAMPBELL; HOANE; HSU, 2002).

Atualmente, a inteligência artificial está presente em diferentes áreas do cotidiano, seja com o reconhecimento de faces para a identificação de pessoas em fotos no Facebook, a condução de veículos autônomos, o reconhecimento de voz para assistentes virtuais (KAPLAN; HAENLEIN, 2019), a geração de textos com o ChatGPT (RADFORD et al., 2018) e diversas outras aplicações.

2.2 Aprendizado de Máquina

Os agentes de inteligência artificial necessitam de informações sobre o ambiente em que estão inseridos. Para isso, muitos dados são inseridos de forma formal, a fim de permitir que o computador consiga raciocinar em uma estratégia conhecida como base de conhecimento (GOODFELLOW; BENGIO; COURVILLE, 2016).

No entanto, um dos maiores desafios para essa abordagem encontra-se na dificuldade de formular as descrições do ambiente de forma precisa. Essa limitação foi superada com o avanço de técnicas que permitem que a própria inteligência artificial receba dados e extraia informações relevantes para o problema, em uma tarefa chamada aprendizado de máquina (GOODFELLOW; BENGIO; COURVILLE, 2016).

O processo de aprendizado de máquina ocorre quando um programa de computador aprende, a partir de uma experiência, padrões que otimizam sua performance em um conjunto de atividades específicas nas quais ele foi proposto a realizar (MITCHELL; MUNSON, 1997).

O aprendizado de máquina pode ser dividido em quatro grandes campos: aprendizado supervisionado, semi-supervisionado, não supervisionado e por reforço. Segundo Burkov (2019), eles podem ser definidos da seguinte forma:

- No **aprendizado supervisionado**, dado um conjunto de dados de entrada X, o objetivo é treinar um algoritmo capaz de aprender a função que melhor relaciona esses dados com os seus respectivos rótulos presentes no conjunto Y, uma vez que há a correspondência de um-para-um, ou seja, um dado de entrada refere-se a uma única saída. Em outras palavras, seria como um aluno tentando realizar uma tarefa enquanto um professor aponta onde está correto ou errado.
- No **aprendizado semi-supervisionado**, o conjunto de dados é dividido em duas partes: os dados rotulados, que funcionam como guia direto do modelo, fornecendo a associação entre entradas e saídas esperadas de forma semelhante ao aprendizado supervisionado, e os dados não rotulados, que não possuem rótulos, mas são aproveitados para revelar padrões, estruturas e distribuições subjacentes no espaço de entrada. Dessa forma, enquanto os dados rotulados oferecem a base de referência, os dados não rotulados ampliam o conhecimento sobre o problema, permitindo ao modelo generalizar melhor e produzir predições mais

robustas e precisas. É como se um aluno aprendesse a resolver equações matemáticas a partir de exemplos do professor e, depois, aplicasse o que entendeu para solucionar problemas novos.

- No **aprendizado não supervisionado**, apenas os dados de entrada são conhecidos. Nesse caso, busca-se encontrar uma função que transforme os dados ou revele estruturas ou padrões úteis, como agrupamentos ou distribuições latentes. Tal tipo de aprendizado se assemelha ao de um entusiasta em programação que decide aprender tudo sem a instrução de terceiros.

- No **aprendizado por reforço**, o agente interage com o ambiente em que está inserido, realizando diferentes ações que resultam em recompensas ou penalidades. Com base nesse *feedback*, o agente aprende a tomar decisões que maximizem suas recompensas ao longo do tempo. No mundo real, tal comportamento pode ser visto na criação de crianças, que recebem palavras de afirmação para comportamentos corretos, mas são corrigidas quando realizam algo errado.

2.3 Redes Neurais

Para o bom funcionamento dos algoritmos de aprendizado de máquina, é fundamental que os dados passem inicialmente por um processo de extração de características e identificação de padrões relevantes. No entanto, essa tarefa costuma ser complexa e demorada, exigindo a atuação de especialistas em domínios específicos, o que eleva significativamente o custo de desenvolvimento. Como consequência, esse processo dificulta a generalização dos algoritmos e faz com que muitos deles sejam projetados para resolver apenas problemas muito específicos (GOODFELLOW; BENGIO; COURVILLE, 2016; LECUN; BENGIO; HINTON, 2015).

A automatização da tarefa de encontrar boas representações para os dados pode ser feita com um algoritmo de aprendizado de uso geral, sendo essa a característica principal do aprendizado profundo, o *deep learning* (LECUN; BENGIO; HINTON, 2015). Em seu funcionamento, ao invés de tentar representar a informação como um todo, suas características são quebradas em partes menores para, no final, entender o todo (GOODFELLOW; BENGIO; COURVILLE, 2016).

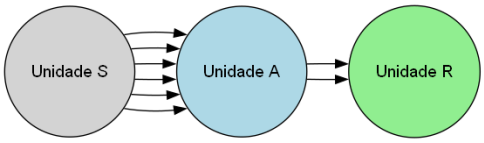
2.3.1 Perceptron

Para realizar a representação por meio de pequenas informações, Rosenblatt (1958) explora a teoria de que o aprendizado em seres vivos ocorre por modificações nas conexões entre os neurônios do cérebro.

Rosenblatt (1958) propõe a simulação de um sistema neural baseado em conexões, o perceptron, um sistema composto por três camadas: unidades S, que detectam estímulos sensoriais,

como os olhos de um animal que vê um predador, unidades A, que recebem os estímulos e os processam de forma não linear, como o cérebro do animal que identificará o que foi visto, processará e decidirá qual ação tomar; e as saídas do sistema, as unidades R, que, no exemplo do animal, seriam a decisão de sair o mais rapidamente possível daquela região. Assim, a representação de todo esse sistema pode ser observada na Figura 2.1

Figura 2.1 – Exemplo de um perceptron proposto por Rosenblatt (1958).



Fonte: Imagem do autor.

Os dados da unidade S vão para a unidade A, onde cada um deles sofrerá uma modificação com base em um peso, como pode ser observado na formulação matemática:

$$a_j = \sum_{i=1}^{N} w_{ij} * s_i, \tag{2.1}$$

onde:

- a_j: Valor da unidade de ativação j;
- N: número total de unidades S conectadas à unidade A;
- $w_{ij} \in \mathbb{R}$: peso da conexão entre s_i e a_j;
- s_i: Valor da i-ésima unidade sensorial.

Sendo que a unidade A somente será considerada ativada quando o valor de a_j for, pelo menos, o limiar de ativação θ, ou seja: $a_j \geq \theta$. Dessa forma, é feita uma função de ativação conhecida como degrau, que resulta somente em valores 0 e 1

Por fim, a saída final do sistema é realizada por meio de uma soma ponderada entre esses valores e os pesos da unidade R.

2.3.2 Perceptron Multicamadas

Mesmo demonstrando o poder de aprendizado do perceptron, Rosenblatt (1958) discute como ele falha ao lidar com abstrações mais complexas, necessitando da formulação de um sistema mais avançado.

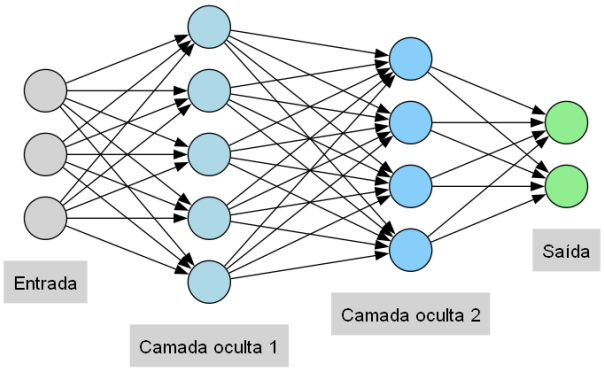
Tal sistema pode ser feito a partir do aumento da quantidade de perceptrons, comumente chamados de neurônios, e separando-os em diferentes camadas, onde um perceptron da camada L está conectado a todos os outros da camada $L + 1$, gerando um perceptron multicamadas, ou em

inglês *Multilayer perceptron* (MLP) (GOODFELLOW; BENGIO; COURVILLE, 2016; ZHANG et al., 2023).

Entre a camada de entrada e a de saída, são adicionadas quantas camadas intermediárias forem consideradas necessárias, sendo conhecidas como camadas ocultas, uma vez que não são diretamente observadas. Esta é uma propriedade somente da camada de entrada e da camada de saída, caracterizando a aprendizagem em profundidade (GOODFELLOW; BENGIO; COURVILLE, 2016; ZHANG et al., 2023).

A Figura 2.2 exibe um exemplo de um MLP com três neurônios na camada de entrada, cinco neurônios na primeira camada oculta, quatro na segunda e dois na camada de saída.

Figura 2.2 – MLP com uma camada de entrada, duas ocultas e uma de saída



Fonte: Imagem do autor

A informação em um MLP é propagada de maneira sequencial, ou seja, a partir da camada L, a informação vai para a $L + 1$ e não retorna para a anterior (GOODFELLOW; BENGIO; COURVILLE, 2016).

Os pesos dos neurônios de uma camada podem ser definidos de forma matricial como w^l_{jk}, onde l é a camada em questão, k é o neurônio da camada anterior e j é o neurônio da camada l (NIELSEN, 2015).

Um viés (também conhecido como *bias*) pode ser adicionado à rede para ter a funcionalidade de um peso adicional. Seu valor é semelhante ao limiar θ do perceptron, mas com o valor oposto. Cada neurônio da camada possui seu próprio *bias*, e ele pode ser representado como b^l_j, onde l indica a camada atual e j o neurônio em questão (NIELSEN, 2015).

Dessa forma, temos a soma ponderada para a saída de um neurônio com respeito a todos os seus valores de entrada, pesos e bias (NIELSEN, 2015), como pode ser visto a seguir:

$$z^l_j = \sum_j w^l_{jk} * x^{l-1}_k + b^l_j, \tag{2.2}$$

onde:

- z^l_j refere-se ao valor de saída do neurônio j da camada l;

- x_k^{l-1} refere-se ao valor de saída da camada anterior ou ao valor de entrada do modelo.

Seguindo tal representação, a rede neural seria uma composição linear de diferentes funções. Dessa forma, para adicionar uma não linearidade aos dados, o valor de saída z^l é modificado por uma função de ativação ϕ não linear (ZHANG et al., 2023). Dessa forma, a saída de um neurônio segue a seguinte equação:

$$a_j^l = \phi(z_j^l). \tag{2.3}$$

Sendo X o conjunto de dados de entrada, W^l a representação matricial de todos os pesos de uma camada, B^l os *biases* da camada l referentes a seus respectivos neurônios, então a Equação 2.4 refere-se à formulação matemática de uma rede neural com duas camadas. Ela pode ser definida por:

$$Z = \phi(W^1(\phi(W^0X + B^0)) + B^1). \tag{2.4}$$

Dessa forma, fica evidente que os valores de um neurônio terão influência no resultado final do processo de propagação dos dados.

2.4 Funções de ativação

Como mencionado anteriormente, os modelos necessitam de uma função de ativação não linear para evitar que uma rede neural seja apenas uma composição linear entre as entradas e saídas, uma vez que uma composição linear é apenas uma função linear que pode ser simplificada.

Além disso, uma característica essencial para as funções de ativação é o fato de serem diferenciáveis, uma vez que o algoritmo de otimização utilizará a derivada das mesmas para encontrar o novo valor para os parâmetros do modelo (ZHANG et al., 2023).

Dessa forma, algumas das principais funções de ativação serão apresentadas a seguir, sendo que, diferentemente do perceptron visto na Seção 2.3.1, podem resultar em valores diferentes de apenas 0 e 1.

2.4.1 *Sigmoid*

A função *sigmoid* tem a característica de "espremer" diferentes valores para o intervalo $[0, 1]$, sendo muito utilizada para a classificação binária (ZHANG et al., 2023). Ela pode ser definida por:

$$sigmoid(x) = \frac{1}{1 + exp(-x)}. \tag{2.5}$$

Já a sua derivada pode ser representada pela equação:

$$\frac{d}{dx} sigmoid(x) = sigmoid(x)(1 - sigmoid(x)). \tag{2.6}$$

2.4.2 *Rectified Linear Unit*

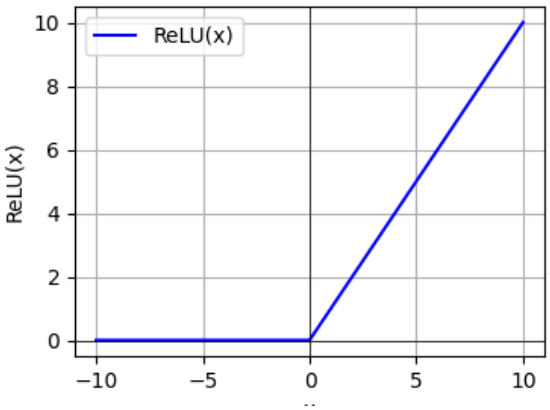
Uma das funções de ativação mais famosas e utilizadas é a Unidade Linear Retificada, ou em inglês *Rectified Linear Unit* (ReLU), onde somente informações positivas são mantidas e qualquer outra é descartada (ZHANG et al., 2023). Ela pode ser definida matematicamente por:

$$\text{ReLU}(x) = \begin{cases} 0 & \text{, se } x < 0 \\ x & \text{, se } x \geq 0 \end{cases}. \tag{2.7}$$

Observando o gráfico da sua função na Figura 2.3, observa-se um ponto de "quina" quando $x = 0$), impedindo a diferenciação da função. Entretanto, para evitar tal problema, sua derivada é definida pela equação:

$$\frac{d}{dx}\text{ReLU}(x) = \begin{cases} 0 & \text{, se } x < 0 \\ 1 & \text{, caso contrário} \end{cases} \tag{2.8}$$

Figura 2.3 – Função ReLU



Fonte: Imagem do autor

2.4.3 *Softmax*

Para problemas de classificação com múltiplas classes, uma forma comum de representar os rótulos é por meio de vetores *one-hot*, ou seja, vetores de N posições (onde N é o número de classes), com todos os valores iguais a zero, exceto na posição correspondente à classe correta, que recebe o valor 1.

Para que a saída de uma rede neural possa ser interpretada como uma distribuição de probabilidade sobre essas classes, é necessário que ela contenha N neurônios e que seus valores resultem em números não-negativos cuja soma total seja igual a 1. No entanto, as saídas brutas da rede nem sempre satisfazem essas propriedades, podendo conter valores negativos e somas arbitrárias (ZHANG et al., 2023).

Para resolver esse problema, utiliza-se a função *softmax*, que transforma os *scores* produzidos pela rede em uma distribuição de probabilidade válida, garantindo que todos os valores estejam no intervalo $[0, 1]$ e que a soma total seja 1 (ZHANG et al., 2023).

A função *softmax* é definida matematicamente por:

$$softmax(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}}, \tag{2.9}$$

onde:

- x_i representa o escore atribuído pelo modelo à classe i;
- K é o número total de classes no problema;
- $\sum_{j=1}^{K} e^{x_j}$ é a soma das exponenciais de todos os valores atribuídos pelo modelo, servindo como fator de normalização.

2.5 Função de erro

Para avaliar a eficiência de um modelo de rede neural supervisionado, verifica-se a discrepância entre os valores previstos e os valores reais esperados (ELHARROUSS et al., 2025).

Cada tipo de tarefa impõe requisitos específicos sobre o modelo. Em tarefas de regressão, a saída esperada é um valor contínuo; já em classificação, espera-se um rótulo discreto. Assim, a escolha da função de erro (ou função de perda) deve ser coerente com a natureza da tarefa. Não é apropriado, por exemplo, empregar uma função de perda voltada à classificação em tarefas generativas, pois, enquanto a primeira visa maximizar a taxa de acertos, a segunda busca produzir saídas com aparência realista (ELHARROUSS et al., 2025).

O cálculo da função de perda considera dois elementos principais: o valor real ou esperado, representado por y, e o valor estimado pelo modelo, denotado por $\hat{y}$.

A seguir está a definição dos principais tipos de erro utilizados em aprendizado de máquina:

2.5.1 Erro Médio Absoluto

Em tarefas de regressão, o Erro Médio Absoluto, ou em inglês *Mean Absolute Error* (MAE), indica o quão distante o valor previsto ficou do valor original na mesma unidade de medida, gerando uma média para todos os N valores calculados (ELHARROUSS et al., 2025). Sendo y_i o valor real e $\hat{y}_i$ é o valor predito para o exemplo i, ele é definido por:

$$L(y, \hat{y}) = \frac{1}{N} \sum_{i=1}^{N} |y_i - \hat{y}_i|. \tag{2.10}$$

2.5.2 Erro Médio Quadrático

Para tarefas de regressão, o Erro Quadrático Médio, ou em inglês *Mean Absolute Error* (MSE), penaliza medidas distantes ao utilizar o quadrado da distância entre os valores previstos e esperados, ou seja, erros crassos sofrerão uma punição maior do que erros pequenos (ELHARROUSS et al., 2025). Sendo y_i o valor real e $\hat{y}_i$ é o valor predito para o exemplo i, o MSE é definido por:

$$L(y, \hat{y}) = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2. \tag{2.11}$$

2.5.3 Erro de Entropia Cruzada

Para tarefas de classificação, o Erro de Entropia Cruzada (*Cross-Entropy Loss*) avalia a diferença entre as distribuições de probabilidade para cada uma das classes do problema, onde os valores previstos somente podem variar entre 0 e 1 (ELHARROUSS et al., 2025). Sendo y_i o valor real e $\hat{y}_i$ é o valor predito para o exemplo i, ele é definido por:

$$L(y, \hat{y}) = -\sum_{i=1}^{N} y_i log(\hat{y}_i). \tag{2.12}$$

2.5.4 Erro de Entropia Cruzada Binária

Para tarefas de classificação com somente duas classes, o Erro de Entropia Cruzada Binária, ou em inglês *Binary Cross-Entropy Loss* (BCE), calcula a distância entre os valores binários previstos e esperados (ELHARROUSS et al., 2025). O BCE é definido por:

$$L(y, \hat{y}) = -(ylog(\hat{y}) + (1 - y)log(1 - \hat{y})). \tag{2.13}$$

2.6 Aprendizado Profundo

Após o resultado de uma predição e com o seu devido erro calculado, pode-se, então, iniciar o processo para a atualização dos parâmetros do modelo, ou seja, modificar os pesos e o *bias* para que a distância entre o valor predito e o valor esperado seja a mais próxima de zero quanto possível (LECUN; BENGIO; HINTON, 2015).

Para a realização do aprendizado, uma técnica muito comum é a aplicação do algoritmo de descida do gradiente. Onde, dada uma função com múltiplos parâmetros, o objetivo é encontrar a combinação que resultará no menor valor possível da função, ou seja, o mínimo local (ALAGöZLü, 2022).

Em um aprendizado supervisionado de uma rede neural, após uma predição, o erro é calculado. Esse valor é uma variável dependente de todos os pesos e *bias* da rede, e encontra-se em algum local no plano multi-dimensional do modelo para os parâmetros atuais. A fim de

encontrar a direção em que a função deverá decrementar o seu valor, o vetor gradiente do erro em função dos pesos pode ser calculado (LECUN; BENGIO; HINTON, 2015).

O vetor gradiente de uma função é a direção na qual os valores da função terão, naquele determinado ponto, a maior taxa de crescimento (STEWART, 2017). Contudo, como o objetivo é encontrar um mínimo local, e não um máximo local, o oposto deste vetor deve ser utilizado (LECUN; BENGIO; HINTON, 2015).

A fim de garantir uma descida suave, o vetor gradiente é multiplicado por um escalar conhecido como taxa de aprendizado (*learning rate*), representado por η. O seu valor define o tamanho do passo para atingir o mínimo local, onde um valor muito pequeno demandará mais tempo, enquanto um valor muito grande pode acelerar o processo, mas pode impedir que o mínimo local seja atingido, fazendo com que a função fique constantemente oscilando próximo àquele ponto (ALAGöZLü, 2022).

Com o erro calculado a partir de todos os dados do conjunto de dados, pode se dar um passo na atualização dos pesos e do *bias* a partir da fórmula geral da descida do gradiente, definida pela equação:

$$\theta = \theta - \eta\nabla(E), \tag{2.14}$$

onde:

- θ refere-se aos pesos ou *bias*
- η refere-se ao *learning rate*
- $\nabla(E)$ refere-se ao vetor gradiente dos pesos ou *bias* em relação ao erro da função

Para que o algoritmo não necessite realizar a predição e cálculo do erro de todos os dados para que só então seja feito um passo na descida do gradiente, é possível realizar um passo a cada predição. Tal processo é conhecido como a Descida de Gradiente Estocástica, ou em inglês *Stochastic Gradient Descent* (SGD), e acelera o processo de convergência dos parâmetros, uma vez que ele está sempre se atualizando, mesmo que de pouco em pouco (LECUN; BENGIO; HINTON, 2015; ALAGöZLü, 2022).

Utilizando ambas as abordagens em um conjunto de dados, este conjunto pode ser dividido em diversos lotes pequenos, conhecidos como mini-lotes, ou em inglês, *mini-batch*. Assim, para cada mini-lote, são feitas suas predições e cálculos dos erros, que serão agregados e utilizados para o passo de descida do gradiente e atualização dos parâmetros, permitindo que, na mesma iteração de dados, avanços sejam feitos de forma procedural e que o próximo mini-lote contribua um pouco mais para o avanço que foi beneficiado. Dessa forma, evita-se ficar constantemente atualizando a rede, que é um processo custoso, ao mesmo tempo que não será necessário esperar pela predição de toda a base para realizar um pequeno passo (ALAGöZLü, 2022).

Para a computação dos vetores gradientes dos pesos e *bias*, é necessário um algoritmo eficiente, uma vez que as redes neurais podem possuir uma grande quantidade de parâmetros a serem ajustados. Dessa forma, foi proposto o algoritmo de retropropagação, *backpropagation* em inglês, em Rumelhart, Hinton e Williams (1986).

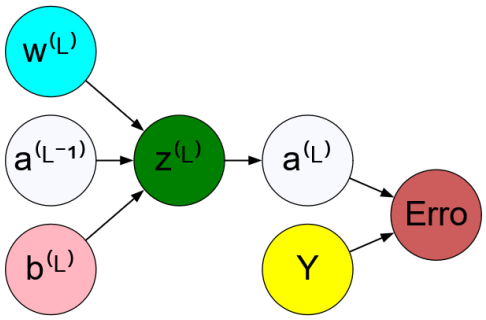
O algoritmo tem o objetivo de calcular o vetor gradiente para cada peso e *bias*, indicando como pequenas variações em seus valores podem modificar o erro final, que é o objetivo a ser minimizado.

A Figura 2.4 exibe um grafo de uma rede neural abstrata com alguns passos e itens necessários para calcular o erro de uma predição, que pode ser realizado pela equação:

$$Erro = \varepsilon(\phi(w^l * a^{l-1} + b^l), Y), \tag{2.15}$$

onde ε se refere a uma função de erro arbitrária, ϕ é uma função de ativação arbitrária, w^l são os pesos da camada atual, a^{l-1} são os resultados provenientes da ativação da camada anterior, b^l são os *bias* da camada atual e Y é o resultado esperado.

Figura 2.4 – Representação da computação do erro de uma rede neural. W indica os pesos da camada, b o bias, a o resultado após ativação, z a soma ponderada antes da ativação e Y o valor real.



Fonte: *Imagem do autor.*

Para calcular o quanto as modificações w^L impactam o erro, será necessário calcular seu vetor gradiente em relação ao erro. Todavia, tal cálculo não pode ser feito de forma direta, uma vez que mudanças em w^L geram modificações em z^L, depois em a^L e, por fim, no erro.

Dessa forma, para contornar tal problema, é necessário aplicar a regra da cadeia para identificar o quanto cada mudança impactará no resultado final. Portanto, seguindo o grafo da Figura 2.4 em seu sentido oposto, é possível montar uma equação que relaciona cada item com aquele que possui uma interferência direta, como pode ser observado pela equação:

$$\frac{\partial \varepsilon}{\partial w^L} = \frac{\partial \varepsilon}{\partial a^L} \frac{\partial a^L}{\partial z^L} \frac{\partial z^L}{\partial w^L}. \tag{2.16}$$

O cálculo do vetor gradiente do *bias* é semelhante ao dos pesos, como pode ser visto na seguinte equação:

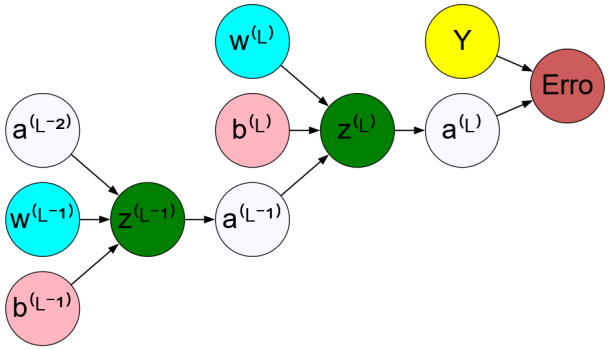
$$\frac{\partial \varepsilon}{\partial b^L} = \frac{\partial \varepsilon}{\partial a^L} \frac{\partial a^L}{\partial z^L} \frac{\partial z^L}{\partial b^L}. \tag{2.17}$$

Para a presença de mais camadas, como na Figura 2.5, o processo é semelhante. Ou seja, basta ir caminhando no grafo seguindo o sentido oposto e adicionando na função os termos diretamente relacionados. Assim, a Equação 2.18 refere-se a w^{L-1}, enquanto a Equação 2.19 refere-se a b^{L-1}, aproveitando os cálculos já realizados das derivadas parciais, evitando operações matemáticas repetitivas e garantindo uma execução viável do algoritmo.

$$\frac{\partial \varepsilon}{\partial w^{L-1}} = \frac{\partial \varepsilon}{\partial a^L}\frac{\partial a^L}{\partial z^L}\frac{\partial z^L}{\partial a^{L-1}}\frac{\partial a^{L-1}}{\partial z^{L-1}}\frac{\partial z^{L-1}}{\partial w^{L-1}}, \tag{2.18}$$

$$\frac{\partial \varepsilon}{\partial b^{L-1}} = \frac{\partial \varepsilon}{\partial a^L}\frac{\partial a^L}{\partial z^L}\frac{\partial z^L}{\partial a^{L-1}}\frac{\partial a^{L-1}}{\partial z^{L-1}}\frac{\partial z^{L-1}}{\partial b^{L-1}}. \tag{2.19}$$

Figura 2.5 – Fluxo de computação de uma rede neural camada a camada até a obtenção do erro



Fonte: *Imagem do autor*.

2.7 Processamento de Linguagem Natural

A comunicação entre humanos e máquinas é muito presente em filmes na cultura POP. Nos filmes do Homem de Ferro, há a presença do JARVIS, uma inteligência artificial capaz de processar e entender a linguagem humana, além de, eventualmente, gerar respostas textuais que possuam sentido e estejam dentro de um contexto. Tal campo de aprendizado de máquina é conhecido como PLN (ALAMMAR; GROOTENDORST, 2024).

2.7.1 Representação textual

Os computadores não entendem textos puros. Dessa forma, é necessário encontrar uma maneira de representá-los numericamente, mas é preciso fazê-lo de forma inteligente, pois palavras semelhantes podem ter contextos diferentes. Um exemplo é a palavra “manga”, que pode ser tanto a fruta quanto uma parte da camisa, necessitando de um contexto maior para entendê-la (ALAMMAR; GROOTENDORST, 2024).

2.7.1.1 Tokens

A primeira etapa para permitir o entendimento é a quebra do texto em pequenas partes, as quais podem ser palavras inteiras ou pedaços de palavras. Tais fragmentos são conhecidos como *tokens*, e o processo de extraí-los é conhecido como *tokenização* (ALAMMAR; GROOTENDORST, 2024).

Para saber o que será considerado como *token*, primeiro cria-se um vocabulário com um tamanho específico. Ele definirá todos os *tokens* que os modelos vão conhecer, assim como caracteres especiais, emojis, instruções de fim de escrita, etc. Para a criação do vocabulário, diferentes algoritmos podem ser utilizados, entre eles o Byte Pair Encoding (BPE), WordPiece e o SentencePiece (ALAMMAR; GROOTENDORST, 2024; ZHANG et al., 2023).

- **BPE**: Começa com um vocabulário inicial formado por todos os caracteres do texto, depois, une os pares mais frequentes, o algoritmo para quando atinge o tamanho definido do vocabulário. Dessa forma, palavras que o algoritmo verifica como mais frequentes são mantidas inteiras, enquanto palavras mais raras são quebradas em mais partes (SENNRICH; HADDOW; BIRCH, 2016).

- **WordPiece**: Similar ao BPE, inicia com um vocabulário composto por caracteres individuais como *tokens*. A partir do corpus de treinamento, o algoritmo avalia candidatos a sub-palavras e adiciona ao vocabulário aquelas que maximizam o ganho de probabilidade do modelo, em vez de apenas considerar a frequência absoluta. Em cada iteração, o *token* AB com maior *score* é selecionado segundo

$$Score(A, B) = \frac{freq(AB) * tamanho(vocabulario)}{freq(A) * freq(B)} \tag{2.20}$$

  e o processo continua até que o tamanho máximo do vocabulário seja atingido. Assim, palavras frequentes tendem a ser representadas como *tokens* únicos, enquanto palavras raras são segmentadas em sub-palavras menores (SONG et al., 2021).

- **SentencePiece**: O algoritmo trabalha com o texto bruto, tratando-o como uma sequência contínua de caracteres. Ele aprende um vocabulário fixo de sub-palavras a partir do corpus de treinamento, usando um modelo como o BPE ou outros. O processo continua até atingir o tamanho máximo do vocabulário (KUDO; RICHARDSON, 2018).

Na Tabela 2.1 é apresentada a *tokenização* da frase "*É justamente a possibilidade de realizar um sonho que torna a vida interessante*" (COELHO, 2001) para os três tipos de *tokenizadores*. Os modelos utilizados estão disponíveis no HugginFace e são '*openai-community/gpt2*', '*google-bert/bert-base-multilingual-cased*' e '*FacebookAI/xlm-roberta-base*'. A partir deles, é possível perceber como cada um quebra palavras e termos de formas diferentes, não seguindo o mesmo padrão e utilizando diferentes caracteres para simbolizar o início ou meio de palavras.

Tabela 2.1 – Resultado da *tokenização* da frase "É justamente a possibilidade de realizar um sonho que torna a vida interessante"para algoritmos BPE, WordPiece e SentencePiece

Byte Pair Encoding - GPT-2
‘Ãī’, ‘Ġjust’, ‘ament’, ‘e’, ‘Ġa’, ‘Ġposs’, ‘ib’, ‘il’, ‘id’, ‘ade’, ‘Ġde’, ‘Ġreal’, ‘iz’, ‘ar’, ‘Ġum’, ‘Ġson’, ‘ho’, ‘Ġque’, ‘Ġtorn’, ‘a’, ‘Ġa’, ‘Ġv’, ‘ida’, ‘Ġint’, ‘e’, ‘ress’, ‘ante’
WordPiece - BERT-base-multilingual-cased
’É’, ‘just’, ‘##amente’, ‘a’, ‘possibilidade’, ‘de’, ‘realizar’, ‘um’, ‘son’, ‘##ho’, ‘que’, ‘torna’, ‘a’, ‘vida’, ‘interessante’
SentencePiece - XLM-RoBERTa-base
’_É’, ‘_just’, ‘amente’, ‘_a’, ‘_possibilidade’, ‘_de’, ‘_realizar’, ‘_um’, ‘_sonho’, ‘_que’, ‘_torna’, ‘_a’, ‘_vida’, ‘_interessante’

Fonte: *Tabela do autor*.

2.7.1.2 *Embeddings*

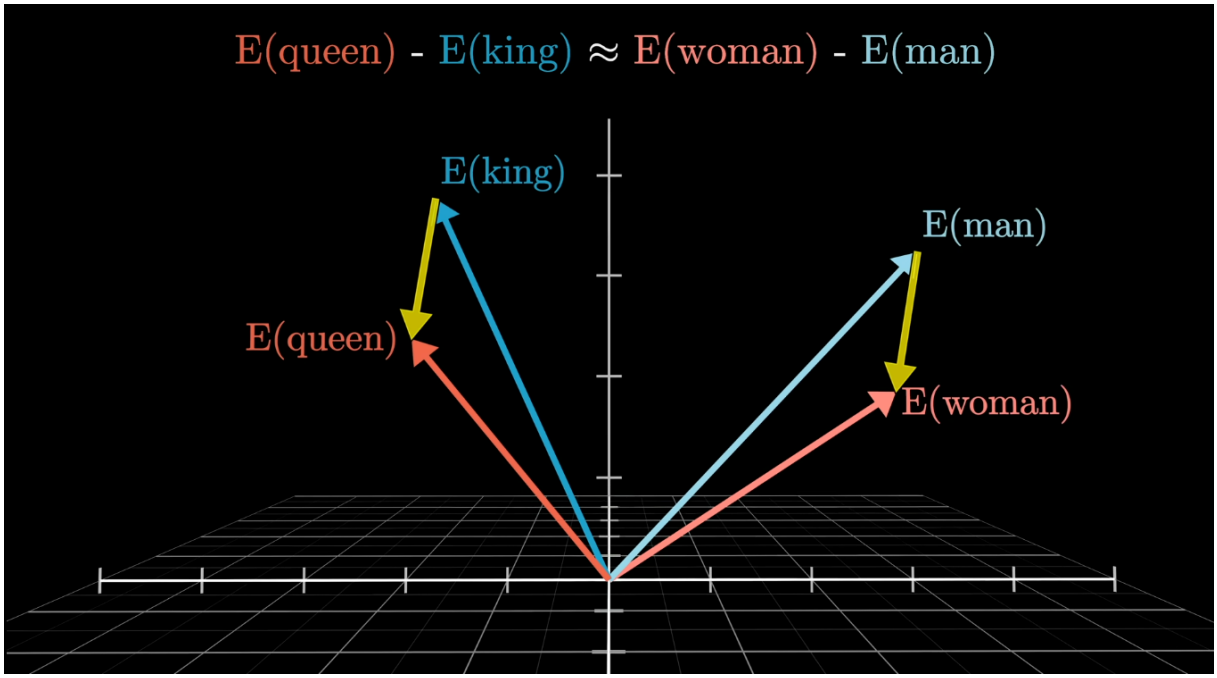
A partir da criação do vocabulário, cria-se uma matriz de valores numéricos para representar os *tokens*. A tabela possui um tamanho de (tamanho do vocabulário $\times$ número de características), dessa forma, cada linha representa uma palavra. Esses valores representativos são conhecidos como *embeddings*, e podem ser treinados por modelos específicos a fim de se obter as melhores representações possíveis, podendo carregar informações semânticas e de contexto, ou seja, palavras similares terão valores próximos (ALAMMAR; GROOTENDORST, 2024).

Como os *embeddings* estão representados matricialmente, cada valor numérico pode ser entendido como uma coordenada em um plano N dimensional, gerando um espaço de *embeddings*. Logo, operações matemáticas podem ser aplicadas para mostrar o quão próximas algumas palavras podem estar (SANDERSON, 2024).

A Figura 2.6 mostra um exemplo das operações matemáticas possíveis no espaço de *embeddings*. Como rei e homem referem-se a figuras masculinas, e mulher e rainha referem-se a figuras femininas, a distância entre o *embedding* de rei, $E(king)$, e rainha, $E(queen)$ e entre homem, $E(man)$, e mulher, $E(woman)$, é próxima. Assim, também é possível aproximar-se de rainha ao fazer a expressão $E(queen) \approx E(king) + E(woman) - E(man)$, uma vez que os o espaço latente dos dados carregam os valores semânticos dos termos. (SANDERSON, 2024).

Para a geração dos *embeddings*, diferentes algoritmos são utilizados. Existem métodos que se baseiam na contagem das palavras, mas ignoram a ordem das mesmas nas sentenças. Outros são estáticos, mas quantificam as posições. Por fim, há métodos que definem contexto para os valores, diferenciando homônimos perfeitos.

Figura 2.6 – Espaço de *embeddings* arbitrário para as palavras *king* (rei), *queen* (rainha), *man* (homem) e *woman* (mulher), mostrando que as diferenças vetoriais dos termos masculinos em relação aos femininos são semelhantes.



Fonte: Sanderson (2024).

2.7.1.2.1 Métodos Baseados em Contagem

Os métodos de geração de *embeddings* baseados em contagem foram uma das primeiras abordagens utilizadas para representar texto de forma numérica em tarefas de PLN, onde constroem vetores a partir da frequência de ocorrência das palavras em um corpus. O objetivo é capturar o significado de um texto a partir da quantidade e distribuição dos termos que o compõem (ZHANG et al., 2023).

O Bag-of-Words (BoW) é uma forma bem simples desse tipo de representação. Ele considera apenas a frequência das palavras em um texto, ignorando a ordem em que aparecem. Inicia-se construindo um vetor com a quantidade de palavras distintas do *dataset*, segurando o valor que cada uma aparece no texto. Embora seja rápido e eficiente, o BoW não captura informações semânticas nem contexto, podendo ter palavras com valores extremamente altos e outras próximas a zero BoW (QADER; AMEEN; AHMED, 2019).

Tentando evoluir o BoW, o Term Frequency–Inverse Document Frequency (TD-IDF) busca atribuir mais informações às palavras ao combinar duas métricas: a frequência do termo no documento e a frequência inversa dos termo nos documentos do corpus. A sua ideia é a de que palavras muito frequentes em um documento, mas raras no corpus como um todo, tendem a ser mais relevantes para caracterizar o conteúdo. Dessa forma, diminui-se o peso de termos muito repetidos, como artigos e preposições, ao mesmo tempo que destaca palavras mais características, gerando mais informações para diferentes tarefas (JONES, 1972).

2.7.1.2.2 *Embeddings* estáticos

Diferente dos métodos baseados em contagem, os *embeddings* estáticos utilizam modelos preditivos ou estatísticos para capturar relações semânticas entre palavras, posicionando termos semelhantes de forma mais próxima no espaço vetorial. São chamados de estáticos por atribuir um único vetor a um termo, independente do contexto em que aparece na frase (ZHANG et al., 2023).

Proposto por Mikolov et al. (2013), o Word2Vec é um modelo extremamente conhecido. Ele utiliza redes neurais rasas para aprender representações distribuídas de palavras a partir do contexto. Existem duas arquiteturas principais: Continuous Bag of Words (CBOW), que prediz uma palavra com base nas que estão ao redor, e o *skip-gram*, que prevê o contexto a partir de uma palavra central. Ao treinar os modelos para realizar essa tarefa de predição, os vetores aprendem propriedades semânticas e sintáticas.

Já o Global Vectors for Word Representation (GloVe), proposto por (PENNINGTON; SOCHER; MANNING, 2014), se inicia com a construção de uma matriz global que registra quantas vezes cada palavra aparece no contexto de outra em todo o corpus. A partir dessa matriz, o modelo aprende vetores ao otimizar uma função objetivo que busca fazer com que o produto interno entre dois vetores de palavras reflita o logaritmo da frequência com que elas coocorrem. A ideia central é a de que não são apenas as coocorrências absolutas que importam, mas principalmente as razões entre probabilidades de coocorrência, pois elas capturam relações semânticas mais profundas, pois "gelo"e "vapor"possuem uma forte relação com "água".

2.7.1.2.3 *Embeddings* contextualizados - BERT

Os *embeddings* contextualizados geram representações dinâmicas, ou seja, a mesma palavra pode assumir vetores diferentes dependendo da frase em que se encontra. Essa característica permite capturar ambiguidades linguísticas e variações semânticas de maneiras mais precisas, evitando ambiguidades e lidando com homônimos perfeitos (DEVLIN et al., 2019).

O Bidirectional Encoder Representations from Transformers (BERT), proposto por Devlin et al. (2019), é um dos principais exemplos de *embeddings* baseados em contexto. Ele utiliza a arquitetura transformer (explorada mais a frente na Seção 2.7.2), mais especificamente, apenas o bloco *encoder*, utilizando do mecanismo de atenção para analisar simultaneamente todas as palavras de uma sentença. Disso surge uma de suas principais características, a análise bidirecional do texto (da esquerda para a direita e da direita para a esquerda), o que permite o entendimento do contexto completo da palavra (DEVLIN et al., 2019).

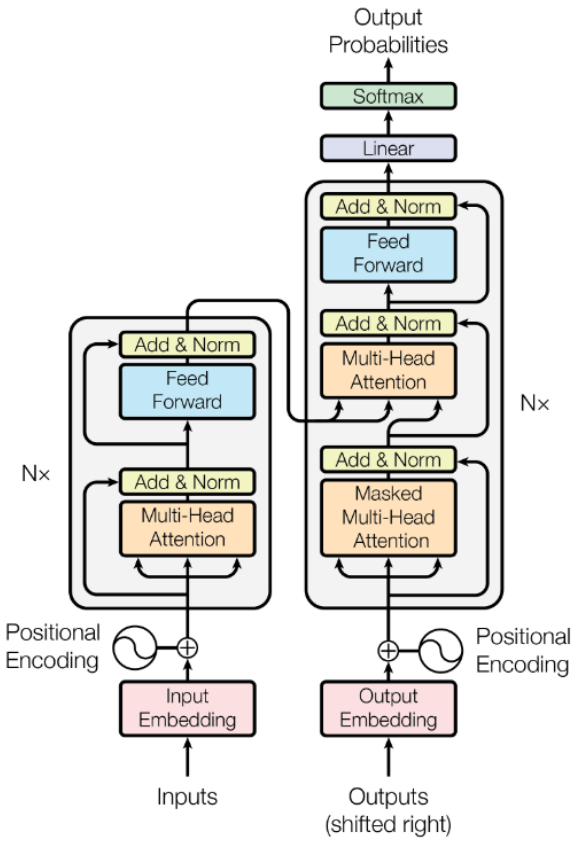
O seu pré-treinamento é feito com a predição de palavras ocultas no meio do texto (*Masked Language Modelin*) e prevendo se uma sentença vem logo após a outra (*Next Sentence Prediction*). Como resultado, o BERT produz *embeddings* ricos de informações semânticas e sintáticas. Esses vetores podem ser utilizados diretamente como representações contextualizadas

ou servir como base para fine-tuning em tarefas específicas, adaptando o modelo a diferentes domínios com pequenas quantidades de dados rotulados (DEVLIN et al., 2019).

2.7.2 Arquitetura baseada em Transformers

Com os *embeddings* devidamente obtidos, é possível alimentar os algoritmos de inteligência artificial. Uma tática comum para trabalhar com sequências era o uso de Recurrent Neural Networks (RNN), que mantêm um certo contexto dos dados. Contudo, essa janela é curta e seu uso pode levar a instabilidades de treinamento e ao desaparecimento do gradiente (ZHANG et al., 2023). Visando solucionar esses problemas, Vaswani et al. (2017) propuseram a arquitetura Transformers, apresentada pela Figura 2.7, utilizando mecanismos de atenção para entender o contexto das palavras no texto.

Figura 2.7 – Arquitetura do modelo de um transformer.



Fonte: Vaswani et al. (2017).

O modelo proposto não mantém a ordem das palavras de forma intrínseca, como ocorre em RNNs. Dessa forma, para codificar essa informação, (VASWANI et al., 2017) propõem uma codificação posicional com os *embeddings* recebidos, somando duas funções senoidais a eles,

definidas por:

$$PE_{(pos,i)} = \begin{cases} \sin\left(\dfrac{pos}{10000^{i/d_{\text{modelo}}}}\right), & \text{se } i \text{ é par} \\ \cos\left(\dfrac{pos}{10000^{(i-1)/d_{\text{modelo}}}}\right), & \text{se } i \text{ é ímpar} \end{cases}, \tag{2.21}$$

onde as posições pares dos *embeddings* utilizam a função seno e as ímpares utilizam a função cosseno. Nela, pos indica a posição do *token* na sequência, i é o índice da dimensão do *embedding*, d_{modelo} é a dimensão do *embedding* do modelo e $10000^{\frac{i}{d_{modelo}}}$ controla a frequência da senoide.

De posse das representações das palavras, agora é necessário entender a relação das mesmas e manipular as representações de forma a garantir uma melhor representação do texto. Isso é feitio a partir do mecanismo de atenção, que tem o objetivo de alterar o *embedding* original, movendo-o para uma posição no espaço em que fará mais sentido. Para realizar essa tarefa, inicia-se calculando o grau de similaridade entre os *tokens* da sequência por dois vetores de pesos treináveis, *Query* e *Key*.

Com esses valores calculados, gera-se a matriz de atenção calculada pelo produto escalar entre cada par de *Query* e *Key*, onde as colunas representam as *queries* e as linhas representam as *keys*. Se os valores são altos, isso significa que há uma relação entre as palavras, por outro lado, se os valores são baixos, pode ser que não exista tal relação. Mas como tais resultados podem ser quaisquer valores reais, aplica-se a função *softmax* para gerar a distribuição de probabilidade para cada coluna. Para ajudar na estabilidade numérica, pode-se adicionar uma divisão pela raiz quadrada da dimensão do espaço *key-query*, ou seja, $\sqrt{d_k}$ (VASWANI et al., 2017).

Esse calculo permite que o modelo entenda quais palavras são relevantes para as outras. Agora, é necessário alterar geometricamente o valor do *embedding* para melhorar na representação das informações. Para isso, é usada a matriz de *Value*, sendo adicionada após o *softmax*, conforme a seguinte equação apresentada por Vaswani et al. (2017), onde Q indica as *queries*, K as *keys* e V os *values*:

$$\text{Atenção}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V. \tag{2.22}$$

Em seguida, os dados de cada *embedding* passam por redes *Multilayer perceptron* (MLP) antes de alimentar a camada de atenção seguinte, passando por somas residuais a fim de adicionar novas informações relevantes às representações dos *tokens*. As camadas do MLP realizam transformações não lineares independentes em cada *token*, permitindo que o modelo refine as representações no espaço de *embeddings* (VASWANI et al., 2017).

Na parte final do modelo, para prever a próxima palavra de uma sentença, utiliza-se a representação do *token* final da sequência e a multiplica por uma matriz conhecida como *unembedding*. A sua função é mapear a última coluna de volta para a probabilidade dos *tokens*, onde pode-se utilizar o *softmax* para definir a próxima palavra (VASWANI et al., 2017).

A arquitetura descrita é conhecida como *encoder-decoder*, pois o *encoder* gera representações contextuais, enquanto o *decoder* as utiliza para prever o próximo *token*, sendo muito útil

para transformar um texto em outro, como ocorre em tradutores (VASWANI et al., 2017). Por outro lado, há outras arquiteturas diferentes, como os *decoder-only* e *encoder-only*.

Arquiteturas *encoder-only* capturam relações entre todos os *tokens*, utilizando atenção completa. Como não há a geração sequencial, os modelos geram representações ricas das informações contextuais. Por isso, são adequados para tarefas de classificação, análise de sentimentos, extração de informações e recuperação semântica, onde o objetivo é entender a entrada e não gerar novos *tokens* (DEVLIN et al., 2019).

Já os *decoder-only* geram um *token* por vez com base nos *tokens* anteriores, isso por meio da atenção mascarada, que impede o acesso a informações futuras. Cada novo *token* gerado é adicionado novamente à entrada para a geração do seguinte, permitindo uma geração contínua de textos (RADFORD et al., 2018).

2.7.3 Modelos de linguagem

A fim de executar tais funções e para compreender o contexto das palavras, modelos de linguagem são muito utilizados com o objetivo de prever a próxima palavra de uma sequência a partir das probabilidades dos padrões aprendidos durante o treinamento. Nesse caso, uma palavra x na n-ésima posição de um texto é dada pela probabilidade da combinação conjunta de todas as outras palavras do texto, ou seja, $x_n \sim P(x_n \mid x_1, x_2, \ldots, x_{n-1})$ (ZHANG et al., 2023).

A partir disso, tais modelos conseguem predizer a próxima palavra de uma frase, possibilitando a geração de textos coerentes, ao mesmo tempo em que as máquinas conseguem aprender a estrutura da linguagem, permitindo aplicações complexas como traduções, resumos, resoluções de questões, etc (ZHANG et al., 2023; ALAMMAR; GROOTENDORST, 2024).

2.7.3.1 Grandes Modelos de Linguagem

Os primeiros modelos de linguagem utilizados em tarefas de PLN eram, em geral, modelos estatísticos ou neurais de menor escala, especialistas em tarefas específicas e com limitações de generalização. Com o aumento da quantidade de dados disponíveis e do poder de processamento, tornou-se possível a ampliação desses modelos para que utilizassem milhões ou bilhões de parâmetros, visando ao aumento do desempenho em tarefas de linguagem natural (ALAMMAR; GROOTENDORST, 2024).

A partir desse significativo aumento de escala, consolidou-se o termo LLM (Grandes Modelos de Linguagem). Tais modelos passam por um processo de pré-treinamento em grandes volumes de texto, o qual é responsável por sua capacidade de compreender a linguagem natural e realizar tarefas para as quais não foram originalmente desenhados, permitindo que atuem tanto na geração de texto quanto em tarefas como classificação ou geração de representações vetoriais de palavras (*embeddings*) (ALAMMAR; GROOTENDORST, 2024).

2.8 Ajuste fino

Quando um modelo é treinado, ele é otimizado levando em consideração a tarefa-alvo e os desafios associados aos conjuntos de dados utilizados durante o treinamento. Considerando um modelo arbitrário para a classificação binária de notícias falsas, esse modelo é calibrado para aprender padrões e representações relevantes a partir dos dados, identificando características que distinguem notícias verdadeiras de falsas (VRBANčIč; PODGORELEC, 2020).

Supondo agora que se deseja estimar o nível de falsidade de uma notícia, e não apenas uma decisão binária, grande parte do conhecimento previamente aprendido pode ser reutilizado, uma vez que as tarefas compartilham similaridades semânticas e estruturais. Nesse contexto, remove-se a camada final de classificação e a substitui por uma nova camada adequada à nova tarefa. Esse procedimento caracteriza a transferência de aprendizado, permitindo aproveitar representações previamente aprendidas e reduzir o custo computacional e a necessidade de grandes volumes de dados para o novo treinamento (VRBANčIč; PODGORELEC, 2020).

Além disso, levando em conta as especificidades de cada conjunto de dados ou desafio, é possível refinar esse modelo pré-treinado por meio de uma técnica chamada *fine-tuning* (ou ajuste fino). Esse processo consiste em adaptar os pesos do modelo de forma mais precisa ao novo domínio, garantindo melhores resultados em tarefas especializadas (VRBANčIč; PODGORELEC, 2020).

O refinamento de modelos não é monotonicamente positivo. Ao se especializar em um novo desafio, o algoritmo não faz distinção do que era importante para a tarefa inicial, permitindo uma atualização indiscriminada dos parâmetros devido a uma alta plasticidade dos mesmos. Tais modificações podem levar a uma degradação extrema dos resultados em um fenômeno conhecido como "esquecimento catastrófico"(KIRKPATRICK et al., 2017).

2.8.1 Parameter-Efficient Fine-Tuning

O *fine-tuning* de uma rede completa pode ser muito custoso, dependendo da quantidade de parâmetros da rede. Enquanto uma rede com alguns milhões de parâmetros pode ser treinada em ambientes mais simples, modelos com bilhões podem necessitar de *hardwares* mais robustos e caros e, ainda assim, levam um certo tempo para serem treinados. Nesse contexto, Houlsby et al. (2019) propõem o Parameter-Efficient Fine-Tuning (PEFT), uma abordagem que evita o treinamento da rede por completo. Em vez disso, pequenos módulos são adicionados à arquitetura do modelo pré-treinado, mantendo os pesos originais intactos.

2.8.1.1 LoRA

Uma técnica de PEFT é o LoRA, que trabalha com a decomposição de matrizes. Quando um peso W_0 é atualizado, há uma variação ΔW para indicar a magnitude da modificação. Dessa

forma, o novo valor dos pesos W_1 pode ser entendido como a soma dos pesos atuais, W_0, mais a variação dos valores, ΔW, ou seja, $W_1 = W_0 + \Delta W$ (HU et al., 2021).

Porém, o tamanho de ΔW pode ser muito grande, aumentando a quantidade de memória necessária. Por exemplo, caso as dimensões de ΔW sejam (200, 200), 40.000 parâmetros são necessários. Decompondo essa matriz em duas, A e B, com dimensões (2, 200) e (200, 2), somente 800 parâmetros são necessários e poucas modificações foram realizadas na fórmula original (HU et al., 2021), como pode ser visto segundo a fórmula:

$$W_1 = W_0 + \Delta W = W_0 + BA. \tag{2.23}$$

Para o uso do LoRA, outras informações são necessárias, como a definição do *rank* das matrizes decompostas, que é definido como o menor valor entre os números de linhas e colunas, sendo 2 no exemplo acima. Ademais, o resultado de 'BA' pode ser multiplicado por um valor alpha (α) para determinar o tanto de mudanças adicionada aos pesos (HU et al., 2021).

Para realizar o treinamento utilizando bibliotecas como a PEFT proposta por HugginFace (n.d.), uma série de outras parâmetros também são passados. Conhecidos como 'target modules', eles definem os pesos que serão alterados pelos modelos. Os campos *q_proj*, *k_proj* e *v_proj* permitem o treinamento da *Query*, *Key* e *Value*, respectivamente. Já o *o_proj* reorganiza a saída das múltiplas cabeças de atenção de volta à dimensão original. Por fim, *gate_proj*, *up_proj* e *down_proj* permitem a camada MLP controlar o fluxo, expandir dimensões e reduzir dimensões, respectivamente.

2.9 Tarefas e avaliação

Os algoritmos utilizados para os diferentes cenários possuem características próprias e apresentam métricas adequadas para quantificar o desempenho.

2.9.1 Classificação

As tarefas de classificação fazem parte dos problemas de aprendizado supervisionado. Nela, o modelo aprende que há N diferentes classes e que precisa atribuir uma classe a cada instância de entrada (BURKOV, 2019), como, por exemplo, definir se um texto é ou não um discurso de ódio.

Um algoritmo muito conhecido de classificação é a regressão logística. Ela funciona de forma binária, atribuindo valor 0 para uma classe e 1 para a outra. A predição final pode ser um valor numérico como 0,7, dessa forma, para atribuir a sua classe, utiliza-se um limiar, sendo 0,5 um valor muito comum. Como $0,7 > 0,5$, logo a classe da predição será 1 (BURKOV, 2019).

Caso deseja-se classificar em mais de duas classes, outra abordagem precisa ser seguida. Uma muito comum é a de *one-hot-encoding*, onde é criado um vetor com tamanho N e cada posição representa uma classe. Assim, aplica-se a regressão logística para cada uma dessas posições, a fim de definir a classe final da instância de entrada (BURKOV, 2019).

Entre as métricas utilizadas para definir a eficiência de um classificador, encontra-se a acurácia. Sendo definida pela Equação 2.24. Nela, divide-se a quantidade de itens corretamente classificados pela quantidade total de itens no conjunto de dados (BURKOV, 2019). Ela é definida por:

$$\text{Acurácia} = \frac{N_{\text{corretos}}}{N_{\text{total}}}. \tag{2.24}$$

2.9.2 Clusterização

A tarefa de *clusterização* consiste em particionar um conjunto de dados em grupos, de forma semelhante à classificação, porém sem conhecimento prévio das classes. Assim, um *clustering* corresponde à partição completa dos dados em *clusters*, nos quais os elementos de um mesmo grupo tendem a apresentar maior similaridade entre si do que em relação aos demais (VINH; EPPS; BAILEY, 2009).

Um algoritmo conhecido de *clusterização* é o de K-médias. Inicialmente, define-se um valor K, muitas vezes desconhecidos, de quantos *clusters* serão procurados. Tal valor é um hiper-parâmetro e pode ser otimizado; em seguida, inicia-se k centroides, escolhidos de forma aleatória, e atribui cada ponto de treinamento ao centroide mais próximo. Após essa etapa, os centroides são recalculados como a média dos pontos pertencentes a cada *cluster*. Esse processo se repete até que os centroides não mudem significativamente e representem bem os dados como se observa na Figura 2.8 (CHONG et al., 2021).

Uma métrica amplamente utilizada para avaliar a similaridade entre dois agrupamentos é a Adjusted Mutual Information (AMI) (VINH; EPPS; BAILEY, 2009). Supondo U e V como dois *clusterings* distintos de um mesmo conjunto de dados. A AMI é definida como:

$$\text{AMI}(U, V) = \frac{\text{MI}(U, V) - \mathbb{E}[\text{MI}(U, V)]}{\max H(U), H(V) - \mathbb{E}[\text{MI}(U, V)]} \tag{2.25}$$

em que:

$\text{MI}(U, V)$ é a *Mutual Information* (Informação Mútua) entre os agrupamentos;

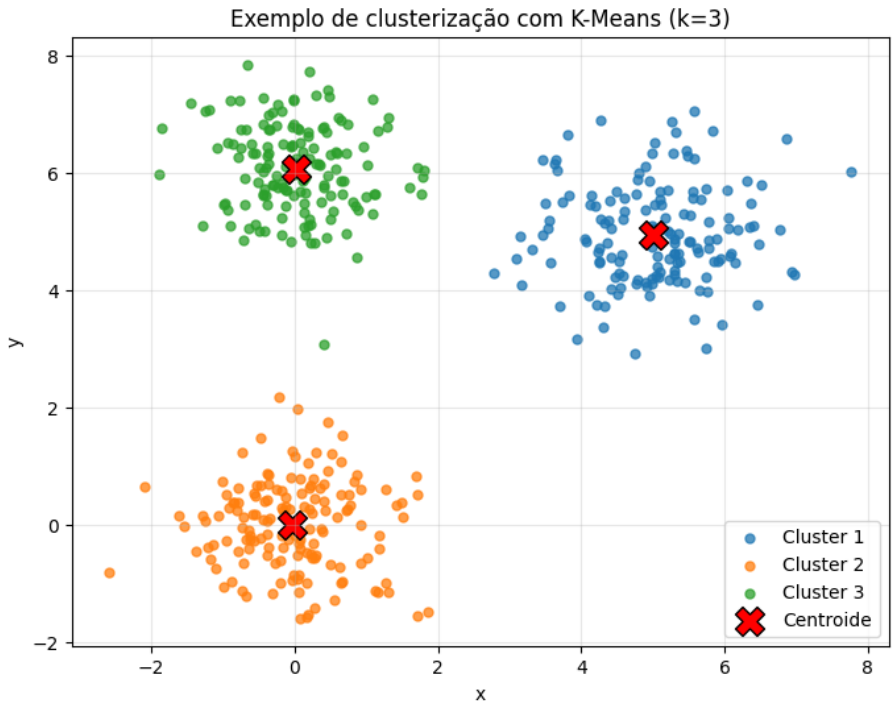
$\mathbb{E}[\text{MI}(U, V)]$ é o valor esperado da Informação Mútua ao acaso;

$H(U)$ e $H(V)$ são as entropias associadas aos agrupamentos U e V.

A Informação Mútua é definida como:

$$\text{MI}(U, V) = \sum_i \sum_j p(i, j) \log \left(\frac{p(i, j)}{p(i)p(j)} \right), \tag{2.26}$$

Figura 2.8 – Exemplo de clusterização via K-médias com $k = 3$ e exibição dos centroides.



Fonte: *Imagem do autor*

em que $p(i)$ e $p(j)$ representam as probabilidades de um elemento pertencer ao cluster i de U e ao cluster j de V, respectivamente, enquanto $p(i, j)$ representa a probabilidade conjunta.

O AMI mede o quanto dois agrupamentos compartilham informação em comum, ajustando esse valor para descontar a similaridade que ocorreria apenas por acaso, dessa forma, o resultado fica no intervalo $[0, 1]$. AMI=1 indica agrupamentos idênticos, enquanto valores próximos de 0 indicam similaridade equivalente ao acaso, permitindo comparações consistentes mesmo quando os agrupamentos possuem diferentes números de *clusters* ou distribuições de tamanhos (VINH; EPPS; BAILEY, 2009).

2.9.3 Natural Language Inferencing

O problema de Natural Language Inferencing consiste em determinar se uma hipótese h pode ser inferida a partir de uma premissa p. A relação entre as duas sentenças pode ser de implicação, contradição ou neutra. Em suma, o problema volta a ser uma tarefa de classificação com três classes possíveis, podendo utilizar a acurácia para medir a qualidade dos resultados (HARSHA; SWAROOP; CHANDAVARKAR, 2021).

A implicação ocorre quando a hipótese pode ser deduzida da premissa. Em um simples exemplo, se a premissa é a de que "o livro está sobre a mesa"a hipótese é de que "o livro está em um móvel". Por outro lado, seria uma contradição se a hipótese fosse de que "o livro está no chão", pois há uma oposição em relação à premissa. Por fim, seria neutro se a hipótese fosse "é importante

ler livros", pois não há uma implicação ou contradição nessa sentença (AZWARSHARIQ, 2023).

A tarefa de NLI é fundamental em diversas aplicações de Processamento de Linguagem Natural, como sistemas de perguntas e respostas, verificação automática de fatos e sumarização, pois exige que o modelo capture relações semânticas, conhecimento implícito e, em alguns casos, raciocínio lógico (HARSHA; SWAROOP; CHANDAVARKAR, 2021).

2.9.4 *Semantic Textual Similarity*

A tarefa de *Semantic Textual Similarity* (STS) tem como objetivo medir a similaridade semântica entre duas unidades de texto. Para isso, ela avalia o quão próximos semanticamente são os significados de duas sentenças, gerando como resultado um número real que indica a pontuação de semelhança (BOUDAA; BOUDAA; HADDADI, 2024).

Como resultado dessa tarefa, a saída é um valor contínuo indicando o quão semelhantes os textos são. Assim, os coeficientes de correlação de Pearson e Spearman são utilizados para medir a concordância entre a previsão feita e os resultados previamente anotados (HE; DUMDUMAYA; QUIMNO, 2024).

O coeficiente de Pearson mede a força e a direção entre duas variáveis lineares, no caso, a pontuação prevista e a anotada. Os resultados variam entre -1 (forte relação linear negativa - enquanto um cresce, o outro decresce), 0 (sem relação linear) e 1 (forte relação linear positiva - os dois crescem ou diminuem juntos) (HE; DUMDUMAYA; QUIMNO, 2024).

O seu calculo é feito a partir do grau de interdependência linear de duas variáveis aleatórias, normalizado pelo produto de seus desvios-padrão. Formalmente, seja $X = (x_1, x_2, \ldots, x_n)$ o vetor de pontuações previstas pelo modelo e $Y = (y_1, y_2, \ldots, y_n)$ o vetor de pontuações anotadas manualmente. O coeficiente r de Pearson é definido como:

$$r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

em que $\bar{x}$ e $\bar{y}$ representam as médias amostrais de X e Y, respectivamente. O numerador corresponde à covariância entre as variáveis, enquanto o denominador realiza a normalização pelos desvios-padrão, garantindo que o resultado esteja no intervalo ([-1, 1]). Dessa forma, o coeficiente expressa o grau de associação linear entre as pontuações previstas e as anotadas (WINTER; GOSLING; POTTER, 2016).

Já o coeficiente de Spearman mede a força e a direção de uma relação monotônica entre duas variáveis (à medida que uma variável aumenta, de forma consistente, a outra também aumenta ou diminui), sendo mais adequado para situações em que os dados não seguem uma distribuição normal ou linear (HE; DUMDUMAYA; QUIMNO, 2024).

Ele é calculado com base em *ranks* das observações, e não diretamente nos valores originais. Inicialmente, substitui-se os valores de X e Y por suas respectivas posições ordenadas

$R(X)$ e $R(Y)$. Em seguida, calcula-se o coeficiente de Pearson sobre esses postos. Quando não há empates, o coeficiente de Spearman (ρ) pode ser expresso pela fórmula simplificada:

$$\rho = 1 - \frac{6 \sum_{i=1}^{n} d_i^2}{n(n^2 - 1)}$$

em que (d_i) representa a diferença entre os postos da i-ésima observação (($d_i = R(x_i) - R(y_i)$)) e n é o número de pares avaliados (WINTER; GOSLING; POTTER, 2016).

2.10 Regimes de avaliação

Os regimes de avaliação de modelos referem-se às diferentes estratégias utilizadas para medir o desempenho, a capacidade de generalização e a qualidade das representações aprendidas por um modelo de aprendizado de máquina. Em vez de apenas analisar métricas tradicionais como acurácia em um conjunto de teste, esses regimes buscam verificar se o modelo compreendeu padrões gerais do problema, se consegue transferir o que aprendeu para novos desafios, se as representações do conhecimento fazem sentido, etc (GENG et al., 2025).

Um regime bem conhecido é o *Zero-Shot*, onde o modelo, sem nenhum treinamento extra, é avaliado em tarefas ou classes que não foram vistas durante o seu treinamento. O objetivo é verificar a capacidade de generalização para cenários completamente novos (XIAN et al., 2020). Seguindo esse pensamento, Li et al. (2025) mostra que esse tipo de avaliação tornou-se especialmente relevante com o surgimento de modelos pré-treinados em larga escala, como modelos de linguagem e modelos multimodais, que aprendem representações amplas a partir de grandes volumes de dados. Nessa configuração, o modelo recebe apenas uma pequena informação sobre o desafio que está trabalhando e precisar inferir, corretamente, a saída com base no seu conheicmento previamente adquirido.

Do ponto de vista metodológico, o zero-shot pode ocorrer de diferentes formas. Em tarefas de classificação de texto, o modelo pode utilizar *embeddings* textuais para relacionar as características ao que foi aprendido. Em modelos de geração de texto, pode-se simplesmente fornecer uma instrução, sem qualquer atualização de pesos. O aspecto fundamental é que não há adaptação supervisionada adicional para aquela tarefa específica. Assim, o desempenho obtido reflete diretamente a capacidade do modelo de reutilizar conhecimento previamente internalizado e estruturar inferências em novos contextos (XIAN et al., 2020; LI et al., 2025).

Por outro lado, o *Linear Probing* é um regime de avaliação voltado especificamente para analisar a qualidade das representações internas aprendidas por um modelo. Nesse cenário, todos os parâmetros do modelo pré-treinado permanecem congelados, e apenas uma camada linear, normalmente uma regressão logística ou uma camada densa de uma rede neural, é treinada sobre as representações extraídas. O objetivo é verificar se as informações relevantes para determinada

tarefa já estão organizadas de forma linearmente separável no espaço vetorial aprendido pelo modelo (ALAIN; BENGIO, 2018).

Se a camada linear obtém bom desempenho, isso indica que o modelo já codificou as características discriminativas necessárias, mesmo sem ter sido treinado diretamente para aquela tarefa. Caso contrário, entende-se que as representações não capturam adequadamente os atributos necessários ou que a tarefa exige transformações não lineares adicionais. Dessa forma, o *linear probing* funciona como uma ferramenta que permite investigar em quais camadas da rede certas propriedades se fazem presentes e qual o grau de entendimento obtido pelas representações (ALAIN; BENGIO, 2018).

3 Revisão Bibliográfica

Este capítulo apresenta, na Seção 3.1, os principais artigos utilizados como base para o estudo realizado, onde são feitas comparações sobre modelos monolíngues e multilíngues em diferentes cenários, e exploram questões sobre os *tokenizadores* utilizados. Já a Seção 3.2 apresenta modelos multilíngues existentes e relevantes e suas características. Por outro lado, a Seção 3.3 apresenta os modelos monolíngues focados no português brasileiro. Por fim, a Seção 3.4 apresenta as lacunas existentes na literatura e explica a direção seguida pelo trabalho.

3.1 Trabalhos Relacionados

Para verificar como modelos multilíngues lidam com múltiplas representações específicas dos idiomas, Wu e Dredze (2020) analisam a qualidade das representações das 104 línguas cobertas pelo BERT multilíngue (mBERT) e as comparam com *baselines* monolíngues e bilíngues. Seus resultados revelam que o BERT multilíngue (mBERT) não apresenta desempenho uniforme entre as línguas: ele enfrenta dificuldades significativas em idiomas com poucos dados de pré-treinamento, enquanto línguas com maior disponibilidade de dados são melhor atendidas por modelos monolíngues especializados. Curiosamente, o mBERT alcança seu melhor desempenho relativo em línguas de médio recurso, nas quais a escassez de dados é reduzida, mas os modelos monolíngues já não oferecem uma vantagem clara.

Buscando avaliar o impacto de modelos monolíngues, multilíngues e estratégias cross-lingual na detecção de discurso de ódio em línguas indianas de baixo recurso, Ghosh e Senapati (2025) exploram modelos baseados no BERT. São usados conjuntos de dados existentes para hindi, marathi e bangla, além de proporem um novo corpora para assamês e bodo, permitindo análises em cenários multilíngues, monolíngues e de transferência entre línguas. Os resultados mostram que modelos multilíngues apresentam melhor desempenho em línguas com menos recursos, ou sem modelos dedicados (como o bodo), enquanto modelos monolíngues se destacam com a presença de dados suficientes para o pré-treinamento. Por fim, os experimentos cross-lingual verificam que a transferência entre línguas da mesma família pode reduzir a necessidade de grandes volumes de dados anotados, sendo ideal para cenários de baixos recursos.

Analisando a classificação de comentários de código como funcionais (descrevem comportamento) ou não funcionais (meta-informações) em cenários monolíngues, multilíngues e *cross-lingual*, Kostić, Batanović e Nikolić (2023) exploram modelos neurais como ELECTRA, BERTić, mBERT e XLM-RoBERTa. Usando 10 mil comentários de código em inglês e sérvio, os resultados também mostraram que os modelos monolíngues, como o BERTić para o sérvio, foram mais eficazes quando há dados suficientes e representativos, enquanto o mBERT apresenta melhor desempenho para comentários em inglês quando treinado de forma multilíngue.

Comparando diferentes abordagens de treinamento entre modelos multilíngues e monolíngues, Chen et al. (2024) perceberam que, treinar um modelo com LoRA em múltiplas línguas tende a gerar melhor desempenho médio e maior robustez linguística. Agora, fazendo o *fine-tuning* completo, os resultados são mistos, com multilíngue superando os modelos monolíngues em alguns cenários específicos. Um achado importante é que treinar modelos multilíngues com menos dados por língua, mas cobrindo múltiplas línguas, apresenta resultados competitivos e é mais robusto em idiomas não vistos, garantindo uma melhor generalização. Por outro lado, usar apenas um modelo em inglês para responder em diferentes línguas funciona razoavelmente bem para línguas próximas, mas gera muitos erros para línguas distantes, como o russo e o chinês.

Investigando alternativas mais eficientes e sustentáveis aos grandes modelos multilíngues, Singh, Clercq e Lefever (2023) propõem a geração de modelos monolíngues a partir da destilação de conhecimento de modelos multilíngues (como mBERT e XLM-RoBERTa) sobre grandes volumes de texto monolíngue. Com seis línguas exploradas, variando o nível de recursos entre elas, foi possível perceber que, para tarefas de *downstream* sintáticas e semânticas, os modelos destilados frequentemente igualavam ou superavam seus professores. Ademais, foi mostrado que adaptar o vocabulário filtrando somente *tokens* relevantes para a palavra alvo ajuda na eficiência sem degradar o desempenho, mostrando que modelos monolíngues são uma alternativa para línguas com poucos recursos.

Buscando lidar com a sub-representação do finlandês no mBERT, Virtanen et al. (2019) treinaram um modelo BERT-base do zero e relataram melhorias substanciais em todas as tarefas avaliadas quando comparado ao mBERT. Uma limitação semelhante é observada para o português brasileiro, para o qual Souza, Nogueira e Lotufo (2020) introduziram o BERTimbau, explorando o brWaC, o maior corpus público disponível em português. Seu modelo monolíngue superou consistentemente o mBERT, alimentando ainda mais a discussão sobre os trade-offs entre arquiteturas monolíngues e multilíngues.

Expandindo essa perspectiva, Rust et al. (2021) compara modelos de linguagem multilíngues, como o mBERT, e seus equivalentes monolíngues. Nove línguas são testadas ao longo de cinco tarefas (reconhecimento de entidades nomeadas, análise de sentimento, resposta de perguntas, *parsing* de dependências e POS tagging). São treinados modelo monolíngues e variantes do mBERT com tokenizadores monolíngues e multilíngues, mantendo o volume de dados constante. Os resultados mostram que a diferença de desempenho entre modelos monolíngues e multilíngues não é universal, pois ela depende fortemente da língua e da tarefa. E o estudo demonstra que tokenizadores monolíngues bem adaptados têm impacto tão relevantes quanto o tamanho dos dados de pré-treinamento, sendo capazes de reduzir significativamente a lacuna de desempenho do mBERT em tarefas monolíngues, especialmente em QA e parsing, indicando que modelos multilíngues não são intrinsecamente inferiores quando adequadamente adaptados

Explorando ainda mais o processo de *tokenização*, Petrov et al. (2023) analisaram diversos *tokenkinzadores* e modelos (como GPT, BERT, BLOOm, etc) em múltiplas línguas. Utilizando o

inglês como referência (aproximadamente 1,0 *token* por palavra), o português gera, em média, 1,54 tokens a mais, enquanto outras línguas com uma representatividade muito menor no conjunto de treinamento, como o Sinhala e Santoli gerando de 12 a 13 vezes mais, e o Shan com quase 18 vezes mais *tokens*. Essa fragmentação excessiva diminui a quantidade efetiva de informação passada para os modelos, podendo aumentar os custos associados a requisições a modelos pagos por *tokens* e degrada o desempenho em tarefas de *downstream*.

3.2 *Embeddings* Multilíngues e Modelos Fundacionais

Modelos multilíngues apresentam uma capacidade para representações eficazes de informações em múltiplos idiomas. Um dos principais modelos é o mBERT (DEVLIN et al., 2019), pré-treinado com arquivos da Wikipédia em 104 idiomas diferentes. Já o XLM-R (CONNEAU et al., 2020) treina com 100 línguas, utilizando dados monolíngues sem alinhamento explícito entre os idiomas, sendo pré-treinado de forma não supervisionada para prever palavras que foram mascaradas dos conjuntos de dados.

O modelo LaBSE (CONNEAU et al., 2020) foca em otimizar a similaridade semântica de sentenças em múltiplas línguas, sendo treinado com dados monolíngues e com dados bilíngues para pares de tradução em aproximadamente 6 bilhões de pares de sentenças, cobrindo um total de 109 línguas. Já o paraphrase-multilingual-MiniLM-L12-v2 (REIMERS; GUREVYCH, 2019a) utiliza de arquiteturas siamesas (utiliza a média de dois modelos) para auxiliar nas tarefas de similaridade semântica, sendo treinado, principalmente, com dados de NLI.

Mais recentemente, modelos maiores, com altas capacidades e treinados sobre enormes conjuntos de dados, conhecidos como modelos fundacionais, vieram tomando espaço nos cenários de processamento de linguagem natural. Entre esses modelos, o Gemma (TEAM, 2024) se destaca por utilizar tecnologias similares ao modelo mais famoso da Google, o Gemini, enquanto utiliza uma quantidade significativamente menor de parâmetros. Já o Qwen (ZHANG et al., 2025) é a família de modelos criada pela gigante Alibaba, com uma ampla gama de variações, desde modelos pequenos a extremamente robustos.

Por fim, Ahuja, Vaddamanu e Patra (2025) descobriram que o inglês não é, necessariamente, o modelo mais eficiente para raciocínio em modelos de linguagem. Para isso foram avaliados três modelos diferentes (DeepSeek R1, Qwen 2.5 e Qwen 3) em quatro conjuntos de dados de matemática diferentes e com etapa de raciocínio em sete diferentes línguas. Os resultados mostraram que a quantidade de *tokens* pode cair de 20% a 40% sem perda significativa de métricas. Os resultados sugerem que os modelos multilíngues conseguem manter um bom desempenho enquanto reduzem custo computacional.

3.3 Modelos Monolíngues para o Português

Focando na especialização dos modelos, ocorreu o desenvolvimento de modelos focados em línguas específicas não inglesas, já que essa é uma das principais línguas com conteúdo para treinamento disponível. Entre um dos principais e mais famosos modelos existentes para o português brasileiro é o BERTimbau (SOUZA; NOGUEIRA; LOTUFO, 2020), um modelo BERT pré-treinado do zero em um grande corpus de português brasileiro (brWaC), com 2,6 bilhões de tokens provindos de 3,5 milhões de documentos, treinado para prever palavras ocultas (Mask Language Modeling) e identificar qual a frase seguinte (Next Sentence Prediction).

Após a especialização introduzida pelo BERTImbau, outros modelos começaram a surgir apresentando características e especificidades. O Albertina (SANTOS et al., 2024) é uma família de modelos *encoder-only* baseada no DeBERTa, treinada com foco em diferentes variantes do português; o Serafim (GOMES et al., 2024) é uma família de *sentece-encoder* treinada para produzir embeddings especializados em português para tarefas de classificação, agrupamento e recuperação de informação; e, mais recentemente, o Tucano (CORRÊA et al., 2025), uma família de modelos *decoder-only* focados em português, treinados no grande corpus GigaVerbo com o objetivo de obter forte desempenho em geração de texto em português.

Grandes modelos com bilhões de parâmetros também foram desenvolvidos para a geração de texto em português brasileiro. O Canarim (Maicon Domingues, 2023) possui uma alta capacidade de realizar tarefas onde alguns exemplos foram dados, isso combinando com a sua eficiência ao ser usado em modelos quantizados. O BODE (GARCIA et al., 2024) é um *fine-tuning* do LLaMa 2 utilizando o *dataset* Alpaca traduzido para o português brasileiro, possuindo boas capacidades para tarefas de detecção de sentimentos. Por fim, um dos modelos mais conhecidos é o Sabiá (PIRES et al., 2023), também construído em cima de um modelo LLaMa, apresentando resultados comparáveis ao GPT-4o para questões da OAB e diversas áreas do ENADE, com uma janela de contexto superior aos outros modelos, atingindo 32 mil *tokens*.

3.4 Discussão dos trabalhos

A literatura comparativa entre modelos monolíngues e multilíngues mostram que, em cenários de baixo recurso, especialmente quando não existem modelos monolíngues especializados ou quando o volume de dados disponíveis é pequeno, os modelos multilíngues possuem melhores resultados. Isso ocorre porque eles se beneficiam do compartilhamento de representações entre línguas, principalmente quando pertencem à mesma família linguística. Já os modelos monolíngues tendem a superar os multilíngues quando há grande volume de dados de pré-treinamento ou especialização na língua em questão, pois conseguem utilizar um vocabulário mais adequado, dedicar toda a capacidade do modelo Àquela língua e aprender características da língua com mais profundidade.

Outro ponto central na literatura é o papel da *tokenização* e da representatividade linguística no pré-treinamento. A fragmentação excessiva de *tokens* em línguas pouco representadas prejudica a eficiência e o desempenho em tarefas *downstream*, elevando custos e reduzindo a qualidade das representações. *Tokenizadores* bem adaptados à língua podem reduzir significativamente a lacuna entre modelos multilíngues e monolíngues. No caso do português brasileiro, modelos como BERTimbau e outras famílias especializadas evidenciam ganhos consistentes quando há treinamento dedicado em grandes corpora específicos.

Diante disso, este trabalho busca propor uma avaliação comparativa abrangente entre múltiplas famílias de modelos, investigadas em quatro tarefas distintas e sob dois regimes complementares, *linear probing* e *fine-tuning* eficiente com LoRA. Ao analisar simultaneamente desempenho, estabilidade e efeitos da adaptação, busca-se compreender o equilíbrio entre plasticidade e rigidez em *embeddings* globais e monolíngues, oferecendo evidências mais sólidas para decisões práticas em aplicações de PLN para o português brasileiro.

4 Materiais e Métodos

Neste capítulo, será definida a experimentação realizada no trabalho. Na Seção 4.1, define-se os modelos utilizados e avaliados sobre o conjunto de dados escolhidos na Seção 4.2. A preparação dos mesmos é definida na Seção 4.3. A forma como os modelos serão avaliados é definida na Seção 4.4 e, por fim, a forma como é analisada a qualidade dos *tokens* gerados por cada modelo é apresentada na Seção 4.5. O código fonte pode ser encontrando no repositório do projeto no GitHub.

4.1 Modelos

A fim de verificar as capacidades de modelos multilíngues e monolíngues, diferentes modelos foram selecionados para que os comportamentos pudessem sem entendidos. Assim, sete famílias de modelos distintas foram escolhidas e nove modelos foram investigados. A Tabela 4.1 resume as principais características dos modelos utilizados.

Tabela 4.1 – Resumo dos modelos utilizados

Família	Arquitetura	Tamanho do vocabulário	Parâmetros
Albertina PT-BR	Encoder	128k	139M 887M
BERTimbau	Encoder	29k	110M 335M
Gemma	Decoder	256k	2B
Qwen	Decoder	152k	596M
Serafim	Encoder	29k	335M
ST MiniLM	Encoder	250k	118M
Tucano	Decoder	32k	650M

Fonte: *Tabela do autor*.

Albertina (RODRIGUES et al., 2023; SANTOS et al., 2024) é uma família de modelos *encoder-only*, focados na língua portuguesa e baseados na arquitetura do DeBERTa, onde deram continuidade ao treinamento desse modelo, agora utilizando conjuntos de dados monolíngues em português. O Albertina 900M 'PORTULAN/albertina-900m-portuguese-ptbr-encoder' foi treinado na base de dados brWaC, composto por textos extraídos da internet, enquanto o Albertina 100M ('PORTULAN/albertina-100m-portuguese-ptbr-encoder') utilizou o resultado da filtragem para português do conjunto OSCAR, uma coleção massiva de textos extraídos da internet em múltiplas línguas.

O **BERTimbau** (SOUZA; NOGUEIRA; LOTUFO, 2020) é um modelo monolíngue, focado no português brasileiro, baseado na arquitetura do BERT. Para permitir o funcionamento de

ambas as suas versões, *base* e *large* (‘neuralmind/bert-base-portuguese-cased’ e ‘neuralmind/bert-large-portuguese-cased’), com 110M e 335M de parâmetros, respectivamente, o BERTimbau conta com um tokenizador próprio, construído para um vocabulário em português. O modelo segue um paradigma de pré-treinamento alinhado com *fine-tuning*, feito a partir da base de dados brWaC e complementado com artigos em português da Wikipédia.

O **Gemma** (TEAM, 2024) é uma família de modelos *decoder-only* e *open-source* criados pelo Google. O seu conjunto de dados de treinamento é multilíngue e muito variado, possuindo 6 trilhões de tokens, mas com um grande foco no inglês. Mesmo possuindo 2 bilhões de parâmetros, os autores consideram o modelo como de baixo custo computacional. Neste trabalho, foi utilizada a segunda versão do modelo (‘google/gemma-2-2b-it’).

O **Qwen3 Embedding**, chamado apenas de **Qwen** para fins didáticos, (ZHANG et al., 2025) é uma série de modelos construídos sobre o Qwen3, projetados para tarefas de recuperação de informação, STS e sistemas multilíngues. O seu treinamento consiste em etapas de pré-treinamento com dados fracos artificiais, seguidos por dados de alta qualidade para um *fine-tuning* supervisionado. Para este trabalho, foi selecionado o modelo Qwen3 Embedding de 600 milhões de parâmetros (‘Qwen/Qwen3-Embedding-0.6B’).

O **Serafim PT-*** (GOMES et al., 2024) é uma família de *sentece encoders* focados no português (europeu e brasileiro) e desenvolvidos para tarefas de classificação, *clustering* e recuperação de informação. Os modelos seguem a arquitetura do sBERT, que utiliza os *encoders* do Albertina e BERTimbau para gerar o modelo alvo. Foram utilizadas múltiplas bases de dados para treinar os modelos nas duas variações das línguas lusófonas. Entre as diferentes versões, a escolhida foi a intermediária de 335M de parâmetros (‘PORTULAN/serafim-335m-portuguese-pt-sentence-encoder’).

O Sentence-Transformer MiniLM (ST MiniLM) (REIMERS; GUREVYCH, 2019b), é um modelo treinado por *sentence-transformers*, também conhecidos como sBERT, uma adaptação da arquitetura BERT para operar como uma arquitetura siamesa, com o modelo treinado em cima do BERT ou RoBERTa. Tal técnica permite uma codificação mais eficiente de sentenças, reduzindo o custo computacional para tarefas de STS, *clustering* e recuperação de informação. O modelo escolhido foi o ‘sentence-transformers/paraphrase-multilingual-MiniLM-L12-v2’.

Por fim, o **Tucano** (CORRÊA et al., 2025) é uma família de modelos *decoder-only* que segue uma arquitetura de *Transformer* autoregressiva. Os modelos foram treinados do zero utilizando grandes volumes de dados textuais extraídos da internet, além de contarem com corpora curados para uma maior qualidade. O modelo escolhido é o intermediário com 335 milhões de parâmetros (‘TucanoBR/Tucano-630m’).

4.2 Conjunto de dados

Para investigar a adaptação dos modelos em diferentes cenários, seis conjuntos de dados foram escolhidos para quatro tarefas diferentes. A Tabela 4.2 resume as principais características dos dados explorados.

Tabela 4.2 – Tarefas, métricas e conjunto de dados explorados

Tarefa	**Métrica**	**Conjunto de dados**	**Instâncias de teste**
Classificação	Acurácia	AG News PT	7600
		TuPyE	8734
		HateBR	1400
Clustering	AMI	AG News PT	7600
		TuPyE	8734
		HateBR	1400
NLI	Acurácia	Assin2	2448
STS	Pearson & Spearman	ASSIN	4000
		Assin2	2448
Análise de tokens	-	Em Ambiente Real	14

Fonte: *Tabela do autor*

O **AG News PT** (MARITACA-AI, 2023) (‘maritaca-ai/ag_news_pt’) é uma tradução para o português brasileiro do conjunto de dados AG News (ZHANG, 2015). O *dataset* extraiu, de mais de 2000 fontes, mais de 1 milhão de notícias de quatro grandes categorias: (1) mundo (*world*); (2) esportes (*sports*), (3) negócios (*business*) e (4) ciência e tecnologia (*Sci/Tech*). O conjunto de treino possui 120 mil itens, 30 mil instâncias para cada classe, e o subconjunto de teste possui 7600 itens, 1900 por classe.

O **TuPyE** (Silly-Machine, 2024) integra e aumenta a quantidade de dados de outros estudos para detecção de discurso de ódio em português brasileiro, gerando um conjunto de dados com 43.668 documentos oriundos do Instagram e X (antigo Twitter). O *dataset* possui uma versão com múltiplas classes, com 13 diferentes categorias de ódio/agressão (como racismo, misoginia, xenofobia, etc.) e uma versão binária com as classes ’ódio’ e ’agressivo’, sendo esta a utilizada.

O **HateBR** (VARGAS et al., 2022) possui 7.000 comentários de publicações no Instagram de figuras políticas. Os dados foram anotados manualmente como ‘ofensivos’ e ‘não ofensivos’, e verificados por três especialistas a fim de obter uma maior concordância e diminuir a subjetividade. O dataset é balanceado com 3.500 itens para cada classe.

O **ASSIN** (FONSECA et al., 2016) é um corpus em português voltado às tarefas de STS e inferência textual, composto por 10.000 pares de sentenças extraídos de notícias do Google News do Brasil e de Portugal. A construção do corpus iniciou-se com a coleta de notícias que descreviam o mesmo evento. Em seguida, modelos de Alocação Latente de Dirichlet (LDA), treinados em textos jornalísticos, foram utilizados para identificar sentenças semanticamente

relacionadas entre esses documentos a partir de limiares de similaridade. Os pares selecionados passaram por revisão manual e foram anotados por pelo menos quatro avaliadores, que atribuíram uma pontuação de similaridade (1 a 5) e um rótulo de inferência (*none*, *entailment* ou *paraphrase*). O corpus é balanceado entre pt-BR e pt-PT, com 2.500 pares para treino, 500 para validação e 2.000 para teste em cada variante.

Já o **ASSIN2** (REAL; FONSECA; OLIVEIRA, 2020) é composto por frases mais simples, focadas no português brasileiro. O conjunto de dados apresenta 6.500 pares de frases de treinamento, 500 para validação e 3.000 para teste, aproximadamente. Todos os dados foram manualmente anotados, com valores variando de 1 a 5 para similaridade e as classes de *entailment* como *entailment* ou *none*.

Por fim, o conjunto "Em Ambiente Real" (Apêndice A) foi criado pelo autor para permitir uma análise dos *tokens* gerados pelos modelos em contextos diversos. Esse pequeno *dataset* é composto por 14 sentenças, variando de regionalismos e gírias até trechos da legislação, do cinema e da literatura brasileira.

4.3 Preparação dos dados

Para a maior parte dos dados, não foram necessárias preparações extras significativas, uma vez que o *dataset* original já estava bem formatado. Assim, foi necessário realizar somente a troca do nome de algumas colunas a fim de adequar-se ao *pipeline* proposto de treinamento e avaliação.

Entre o conjunto de dados que precisou de algumas modificações, está o TuPyE. Como já mencionado anteriormente, a versão escolhida foi a binária, com os rótulos '*aggressive*' e '*hate*'. Dessa forma, os dados foram transformados em rótulos numéricos que variam de 1 a 3 de acordo com todas as combinações de '*hate*' e '*aggressive*' presentes no *dataset*, definido pela seguinte equação:

$$\mathrm{y} = \begin{cases} 0 & \text{, se '}\textit{aggressive}\text{' = 0 e '}\textit{hate}\text{' = 0} \\ 1 & \text{, se '}\textit{aggressive}\text{' = 1 e '}\textit{hate}\text{' = 0} \\ 2 & \text{, caso contrário} \end{cases} . \tag{4.1}$$

O HateBR não possui uma distribuição pré-definida de dados de treino e de teste. Logo, foi feita uma divisão aleatória estratificada em relação aos rótulos, de forma que os dados de treinamento possuíssem 80% do conjunto e, consequentemente, o teste ficasse com 20%. Para garantir a reprodutibilidade, foi utilizado como semente aleatória o valor padrão 42.

4.4 Regimes de avaliação

Para avaliar a capacidade dos modelos em diferentes conjuntos de dados e tarefas, duas abordagens são adotadas. A primeira é uma avaliação com *linear probing*, a fim de verificar a qualidade dos *embeddings*, e a segunda é um treinamento de baixo custo feito nesses modelos para adaptá-los à tarefa desejada.

4.4.1 Linear Probing

A fim de se adequar à natureza de cada desafio, diferentes formas de avaliar foram propostas, adicionando as modificações necessárias nas camadas finais a fim de capturar corretamente o essencial de cada modelo.

Para a tarefa de classificação, extraem-se os *embeddings* dos dados de treino e teste. A partir disso, um modelo de regressão logística é treinado com os dados de treinamento por um número máximo de 5.000 iterações e, em seguida, o modelo é avaliado com os dados de teste.

Algo similar ocorre para a tarefa de NLI. O modelo gera os *embeddings* para as premissas e hipóteses, que são combinados em um vetor de características que inclui suas representações, diferença absoluta e seu produto elemento a elemento. Sobre essas informações, treina-se um classificador de regressão logística, assim como na classificação.

Para a tarefa de *clusterização*, somente os dados de teste foram utilizados para a extração dos *embeddings*, sendo utilizados para alimentar o algoritmo de K-médias. O número de grupos foi definido como o número de classes de cada conjunto de dados.

Por fim, para STS, calculou-se a similaridade do cosseno entre os *embeddings*, a fim de verificar quão sintaticamente alinhados os textos estão. Por fim, aplicam-se as métricas de Pearson e Spearman em cima desses resultados.

4.4.2 *Fine-tuning*

Para verificar a plasticidade do modelo em se adaptar aos diferentes desafios propostos, uma solução comum é o *fine-tuning*. Entretanto, como os modelos de linguagem possuem muitos parâmetros e devido à quantidade de experimentos realizados, o treinamento do modelo por completo se tornaria inviável.

Assim, foi realizado o PEFT utilizando o LoRA. O seu uso justifica-se por oferecer um bom equilíbrio entre eficiência computacional, baixo custo de memória e desempenho competitivo, sem alterar a arquitetura do modelo, mantendo uma boa estabilidade no treinamento e possuindo uma alta facilidade de implementação.

O treinamento foi feito para as tarefas de classificação, STS e NLI. Para a clusterização, foi utilizado o mesmo modelo treinado para classificação, uma vez que são tarefas relativamente próximas.

Para o treinamento, foi utilizado um conjunto consistente de hiperparâmetros, a fim de garantir uma comparação justa entre todos. Os modelos foram treinados por 10 épocas, utilizando uma taxa de aprendizado de 3×10^{-5}, *weight decay* de 0,01, uma razão de *warm-up* de 0,06 e treinamento em precisão mista (*fp16*). O método LoRA foi configurado com *rank* igual a 8, *lora_alpha* igual a 16 e *dropout* de 0,05. Os módulos-alvo variam de acordo com as características de cada modelo: para os modelos Qwen, Gemma e Tucano, foram atualizados os módulos *query*, *key*, *value* e *dense*, enquanto para os demais modelos foram atualizados os módulos *q_proj*, *k_proj*, *v_proj*, *o_proj*, *gate_proj*, *up_proj* e *down_proj*.

4.5 Análise da qualidade dos *tokens*

Somente a análise das métricas pode não ser suficiente para uma análise completa dos modelos. Por isso, uma análise quantitativa e qualitativa dos diferentes *tokens* gerados pelos *tokenizadores* de cada modelo é feita.

É comum assumir que, para línguas morfologicamente ricas, como o português brasileiro, *tokenizadores* especializados e específicos da língua são mais eficientes do que alternativas de propósito geral. Nesse contexto, eficiência refere-se principalmente à quantidade de *tokens* gerados, uma vez que sequências menores permitem incluir mais palavras como entrada do modelo, ampliando o contexto disponível (RUST et al., 2021). Com base nessa premissa, foi realizada uma comparação entre os *tokens* produzidos pelos diferentes modelos avaliados, verificando-os tanto de forma quantitativa quanto qualitativa.

A análise quantitativa buscou verificar, para cada *dataset* e modelo, a quantidade média de *tokens* por palavra que são gerados, onde valores mais altos indicariam uma maior fragmentação dos vocábulos. Já a análise qualitativa baseou-se na análise dos *tokens* gerados para as instâncias do conjunto "Em Ambiente Real", observando a forma como as palavras são quebradas e verificando se as divisões estão próximas da morfologia original ou de alguma outra estrutura da língua portuguesa, como as sílabas.

É importante ressaltar que nem todos os *tokens* foram analisados em sua completude. Isso se deve à grande quantidade de informações geradas. Assim, essa análise restringe-se àqueles que, subjetivamente, se demonstraram mais diferentes e desafiadores para os diversos modelos. Ademais, uma maior atenção foi dada para *tokens* provenientes de línguas estrangeiras e para palavras com acentos ou de maior tamanho.

5 Resultados

Neste capítulo, são apresentados os resultados obtidos após a execução do treinamento do modelo proposto. Na Seção 5.1, são apresentados o ambiente onde os experimentos foram executados, bem como os hiperparâmetros obtidos. Na Seção 5.2, os resultados serão explorados, analisando a forma como as classificações foram feitas e levantando hipóteses do que pode ter causado os erros obtidos.

5.1 Configuração dos experimentos

Os experimentos foram executados nas máquinas do CSI Lab, utilizando duas máquinas equipadas com 128 GB de RAM DDR4, GPU NVIDIA GeForce RTX 3090 com 24 GB de memória VRAM e um processador Intel(R) Core(TM) i9-10900F. A linguagem de programação escolhida foi o Python, devido a ampla quantidade de bibliotecas implementadas para a criação de redes neurais. Assim, também foram utilizadas as bibliotecas Transformers (4.52.4) para treinamento, assim como o PyTorch (2.7.0). O *fine-tuning* com LoRA utilizou a biblioteca PEFT (0.18.0). As métricas foram computadas utilizando a biblioteca SciKit-Learn (1.6.1).

5.2 Resultados obtidos

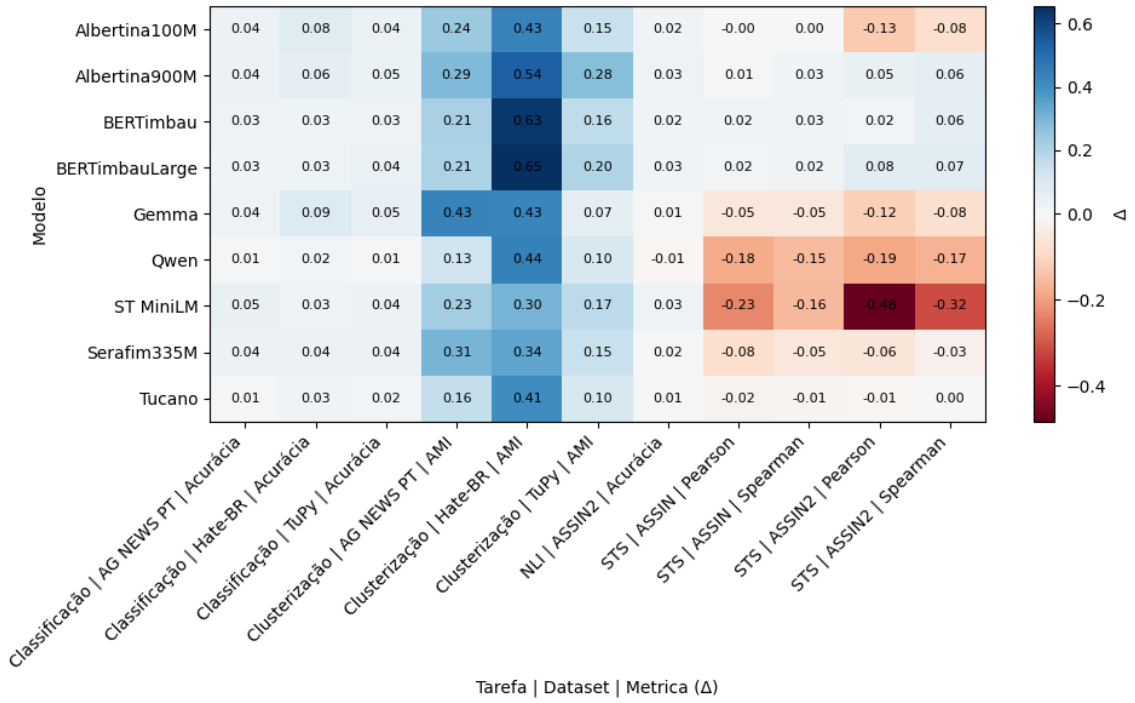
As análises dos resultados obtidos seguirão as perguntas de pesquisa definidas no Capítulo 1, de forma a respondê-las, mas não se restringindo somente a elas, também serão explorados outros temas pertinentes que complementam as discussões feitas.

5.2.1 PP1: *Linear-Probing vs Fine-tuning*

Entre todas as tarefas realizadas, os modelos monolíngues (Albertina, BERTimbau, Serafim e Tucano) apresentaram resultados mais consistentes e estáveis do que os modelos multilíngues (Gemma, Qwen e Sentence-Transformer MiniLM (ST MiniLM)), como pode ser visto na Figura 5.1. Mesmo que os modelos multilíngues se apresentem competitivos em regimes de *linear probing*, eles demonstram uma performance de baixa qualidade quando passam por *fine-tuning*, apresentando melhoras significativas em seus resultados, mas isso não é suficiente para que estejam entre os melhores modelos.

A Tabela 5.1 exibe os resultados da tarefa de classificação nos três diferentes *datasets*. Em suma, o BERTimbauLarge manteve-se o modelo mais estável, apresentando resultados competitivos em ambos os cenários de avaliação. Ou seja, mesmo quando não obteve os melhores resultados, ainda estava muito próximo.

Figura 5.1 – Mapa de calor dos Δ's de variação da avaliação *linear-probing* e *fine-tuning*



Fonte: *Imagem do autor*

No contexto de *linear probing*, os resultados mostram que os modelos apresentam desempenhos muito próximos entre si. No AG News PT, o BERTimbau Large obteve a melhor pontuação (0,8982), superando, por pouco, o Qwen (0,8976), com o Albertina 900M logo em seguida (0,897). No HateBR, o BERTimbau Large novamente aparece como o melhor modelo (0,902), mas com diferença muito pequena em relação ao BERTimbau base (0,900) e ao Serafim (0,898). Já no TuPy, o BERTimbau Large alcançou 0,789, sendo seguido de perto pelo Serafim e pelo BERTimbau, ambos com 0,788. Esses resultados indicam que, ao avaliar apenas a qualidade das representações, os modelos têm desempenhos bastante semelhantes.

Quando os modelos passam pelo *fine-tuning*, as diferenças ficam mais evidentes. No AG News PT, o Albertina 900M apresenta o melhor resultado (0,937), superando com folga o BERTimbau Large (0,929). No HateBR, o Serafim obteve a maior pontuação (0,936), embora muito próximo do BERTimbau Large (0,934). Por fim, no TuPy, o BERTimbau Large alcançou o melhor desempenho (0,830), com o Serafim novamente logo atrás (0,828). De forma geral, o *fine-tuning* permite que cada modelo explore melhor suas capacidades, resultando em ganhos mais claros de desempenho em comparação ao *linear probing*.

A Tabela 5.2 exibe os resultados da tarefa de *clusterização* nos três diferentes conjuntos de dados. De forma concisa, não há um modelo que apresente um maior domínio dos resultados.

No cenário inicial, observa-se que o Serafim domina claramente os resultados no HateBR (0,318) e no TuPy (0,045). No entanto, esse bom desempenho não se repete no AG News, onde

Tabela 5.1 – Resultados da tarefa de classificação (acurácia) sobre regimes de *linear probing* e *fine tuning*. Melhores resultados por conjunto de dados/regimes estão marcados em negrito.

Dataset	Modelo	Linear-Probing	Fine-tuning	Δ
AG NEWS PT	Albertina100M	0,872	0,908	0,036
	Albertina900M	0,897	**0,937**	0,040
	BERTimbau	0,887	0,922	0,035
	BERTimbauLarge	**0,898**	0,929	0,030
	Gemma	0,887	0,922	0,035
	ST MiniLM	0,871	0,921	**0,051**
	Qwen	0,898	0,909	0,011
	Serafim335M	0,887	0,927	0,040
	Tucano	0,892	0,906	0,014
Hate-BR	Albertina100M	0,802	0,880	0,078
	Albertina900M	0,848	0,909	0,061
	BERTimbau	0,900	0,929	0,029
	BERTimbauLarge	**0,902**	0,934	0,032
	Gemma	0,824	0,916	**0,091**
	ST MiniLM	0,844	0,879	0,034
	Qwen	0,873	0,890	0,017
	Serafim335M	0,898	**0,936**	0,039
	Tucano	0,878	0,908	0,030
TuPy	Albertina100M	0,761	0,802	0,042
	Albertina900M	0,774	0,819	0,045
	BERTimbau	0,788	0,823	0,034
	BERTimbauLarge	**0,789**	**0,830**	0,041
	Gemma	0,763	0,818	**0,055**
	ST MiniLM	0,761	0,800	0,039
	Qwen	0,781	0,791	0,010
	Serafim335M	0,788	0,828	0,040
	Tucano	0,784	0,800	0,016

Fonte: *Tabela do autor*

o modelo apresenta um resultado muito inferior, enquanto o Qwen se destaca de forma clara, alcançando a melhor pontuação (0,588). Isso mostra que o desempenho dos modelos varia bastante de acordo com a tarefa e o conjunto de dados avaliado.

Quando é aplicado o *fine-tuning*, os resultados tornam-se mais equilibrados entre os modelos. O Serafim continua sendo o melhor no HateBR (0,662), mantendo sua força nessa tarefa. Por outro lado, o Albertina passa a liderar no AG News (0,802) e também no TuPy (0,281), indicando que o ajuste fino permite que diferentes modelos se destaquem em diferentes cenários.

De forma geral, o *fine-tuning* beneficia todos os modelos avaliados. Ainda assim, os modelos monolíngues apresentam vantagens mais claras, com um delta médio maior de desempenho (Δ 0,256 contra Δ 0,03) e também uma performance média superior (0,498 contra 0,430). Esses

Tabela 5.2 – Resultado do *clustering* (AMI) sobre regimes de *linear-probing* e *fine-tuning*. Os melhores resultados por conjunto de dados/regimes estão marcados em negrito.

Dataset	Modelo	Linear-Probing	Fine-tuning	Δ
AG NEWS PT	Albertina100M	0,478	0,721	0,243
	Albertina900M	0,514	**0,802**	0,289
	BERTimbau	0,521	0,733	0,212
	BERTimbauLarge	0,547	0,754	0,207
	Gemma	0,277	0,712	**0,435**
	ST MiniLM	0,526	0,753	0,226
	Qwen	**0,588**	0,716	0,128
	Serafim335M	0,446	0,752	0,306
	Tucano	0,542	0,697	0,156
Hate-BR	Albertina100M	0,016	0,450	0,435
	Albertina900M	0,022	0,559	0,536
	BERTimbau	0,005	0,636	0,631
	BERTimbauLarge	0,000	0,654	**0,654**
	Gemma	0,004	0,430	0,426
	ST MiniLM	0,152	0,457	0,305
	Qwen	0,007	0,448	0,441
	Serafim335M	**0,318**	**0,662**	0,343
	Tucano	0,020	0,426	0,406
TuPy	Albertina100M	0,002	0,148	0,147
	Albertina900M	0,002	**0,281**	**0,279**
	BERTimbau	0,012	0,177	0,164
	BERTimbauLarge	0,008	0,207	0,199
	Gemma	0,001	0,066	0,065
	ST MiniLM	0,007	0,177	0,171
	Qwen	0,007	0,112	0,105
	Serafim335M	**0,045**	0,191	0,146
	Tucano	0,002	0,105	0,103

Fonte: *Tabela do autor*

resultados sugerem que, para essas tarefas, modelos monolíngues conseguem aproveitar melhor o processo de *fine-tuning* do que os modelos multilíngues.

A Tabela 5.3 exibe os resultados da tarefa de NLI no *dataset* ASSIN2. Nesse desafio, o Serafim 335M domina em ambos os regimes de avaliação (0,839 no *linear-probing* e *0,855* no *fine-tuning*). Por outro lado, o Qwen apresentou uma pequena degradação em seus resultados, atingindo um delta Δ negativo de 0,011.

A Tabela 5.4 exibe os resultados obtidos para a tarefa de STS. É perceptível como o Serafim335M obteve resultados superiores a qualquer outro modelo em ambos os *datasets* e regimes de avaliação, embora tenha ocorrido uma degradação considerável em seus resultados, como observado pelo delta (Δ).

Entretanto, esse fenômeno de diminuição de resultados não é exclusivo do Serafim. Como

Tabela 5.3 – Resultados do NLI (Acurácia) no ASSIN2 sobre regimes de *linear-probing* e *fine-tuning*. Melhores valores estão em negrito

Dataset	Modelo	*Linear-Probing*	*Fine-tuning*	Δ
ASSIN2	Albertina100M	0,763	0,785	0,023
	Albertina900M	0,773	0,805	0,032
	BERTimbau	0,808	0,829	0,021
	BERTimbauLarge	0,772	0,800	0,029
	Gemma	0,724	0,732	0,008
	ST MiniLM	0,790	0,822	**0,032**
	Qwen	0,815	0,804	-0,011
	Serafim335M	**0,839**	**0,855**	0,015
	Tucano	0,769	0,779	0,011

Fonte: *Tabela do autor*

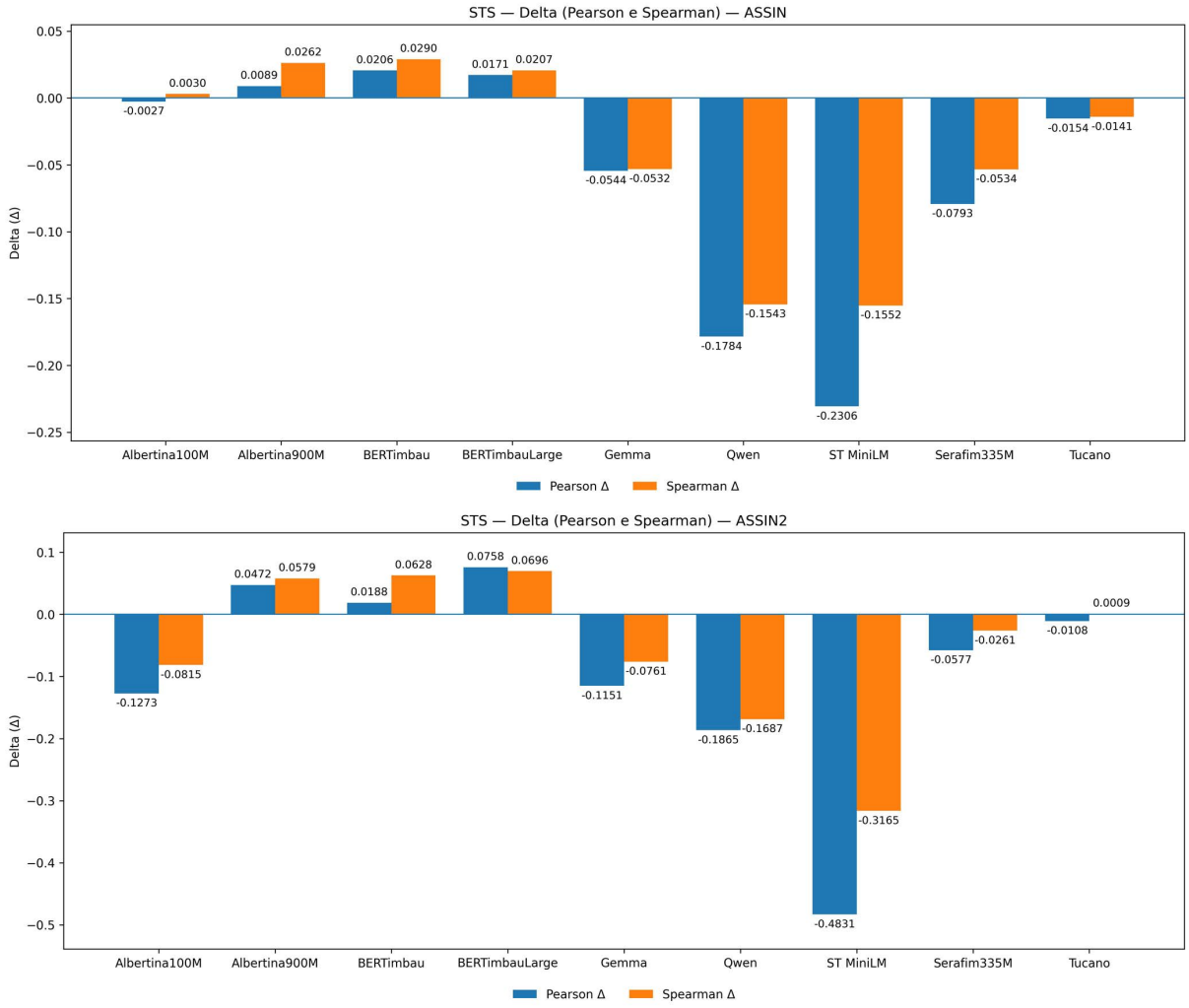
Tabela 5.4 – Resultados do STS (Pearson e Spearman) sobre regimes de *linear-probing* e *fine-tuning*. Os melhores resultados por *dataset/regime estão em negrito.*

Dataset	Modelo	STS Pearson			STS Spearman		
		Linear-probing	Fine-Tuning	Δ	Linear-probing	Fine-Tuning	Δ
ASSIN	Albertina100M	0,584	0,581	-0,003	0,592	0,595	0,003
	Albertina900M	0,599	0,608	0,009	0,608	0,634	0,026
	BERTimbau	0,614	0,635	**0,021**	0,619	0,648	**0,029**
	BERTimbauLarge	0,582	0,599	0,017	0,597	0,618	0,021
	Gemma	0,395	0,340	-0,054	0,454	0,401	-0,053
	ST MiniLM	0,678	0,447	-0,231	0,679	0,524	-0,155
	Qwen	0,745	0,567	-0,178	0,739	0,585	-0,154
	Serafim335M	**0,809**	**0,730**	-0,079	**0,801**	**0,747**	-0,053
	Tucano	0,532	0,517	-0,015	0,541	0,527	-0,014
ASSIN 2	Albertina100M	0,614	0,487	-0,127	0,587	0,506	-0,082
	Albertina900M	0,601	0,648	0,047	0,565	0,623	0,058
	BERTimbau	0,672	0,690	0,019	0,614	0,677	0,063
	BERTimbauLarge	0,681	0,757	**0,076**	0,619	0,689	**0,070**
	Gemma	0,469	0,354	-0,115	0,466	0,390	-0,076
	ST MiniLM	0,772	0,289	-0,483	0,715	0,398	-0,317
	Qwen	0,805	0,618	-0,186	0,744	0,575	-0,169
	Serafim335M	**0,860**	**0,802**	-0,058	**0,832**	**0,806**	-0,026
	Tucano	0,561	0,550	-0,011	0,509	0,510	0,001

Fonte: *Tabela do autor*

observado na Figura 5.2, tal padrão ocorre para todos os modelos multilíngues (Qwen, Gemma e ST MiniLM), sendo estes os que apresentaram as maiores quedas de resultados, além de alguns monolíngues (Albertina100M, Serafim335M e Tucano).

Figura 5.2 – Delta de resultados entre *linear-probing* e *fine-tuning* entre os modelos e métricas avaliados para a tarefa de STS, mostrando uma grande degradação dos modelos multilíngues e alguns monolíngues.



Fonte: *Imagem do autor*

5.2.2 PP2: A dicotomia da adaptação

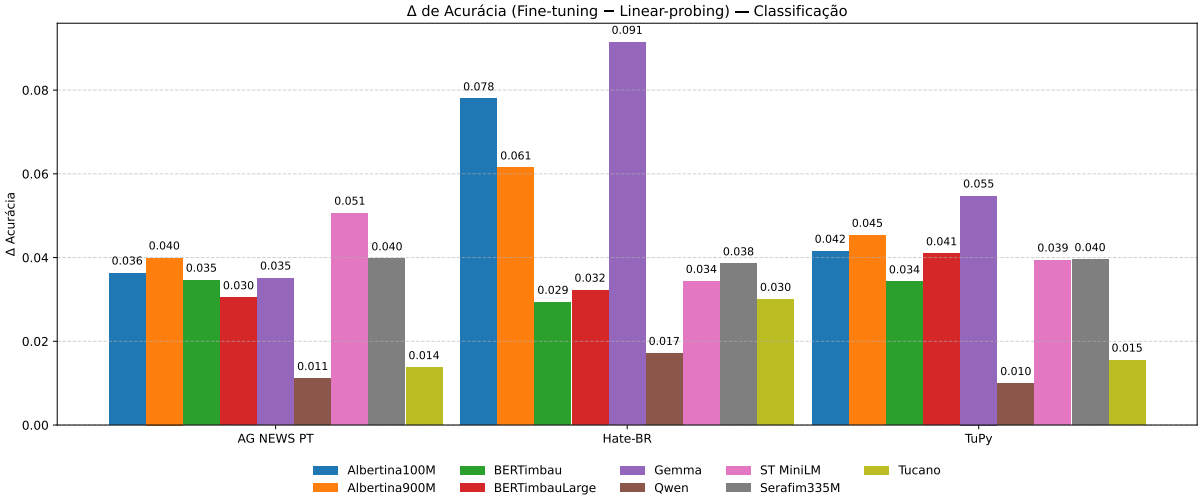
A Figura 5.3 mostra como todos os modelos apresentaram melhorias nos resultados depois do processo de *fine-tuning* para a tarefa de classificação. Mesmo possuindo melhorias marginais, em muitos casos, ainda demonstram a consistência dos modelos.

A tarefa de NLI segue o mesmo caminho e apresenta baixas melhorias, com um único caso de degradação dos resultados, como pode ser evidenciado pela Figura 5.4.

Já a tarefa de *clusterização* apresenta deltas extremamente significativos, que demonstram a importância da especialização dos modelos para essas tarefas. Como pode ser observado na Figura 5.5, o BERTimbauLarge teve um crescimento significativo de seu AMI no *dataset* HateBR, atingindo $\Delta = 0{,}654$.

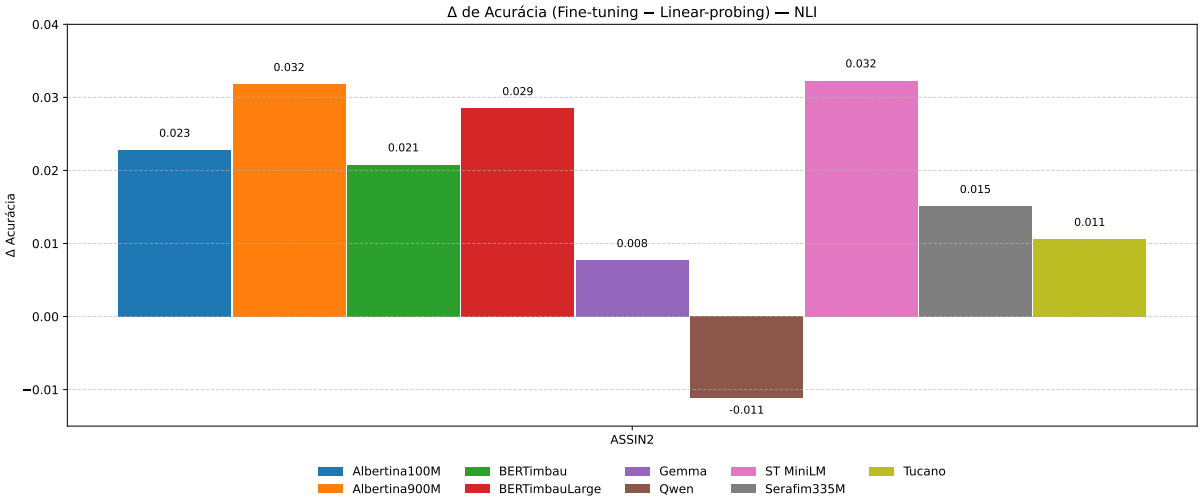
Em contraste a todos os efeitos positivos da especialização, a tarefa de STS já demonstra

Figura 5.3 – Deltas de acurácia para os diferentes conjuntos de dados na tarefa de classificação



Fonte: *Imagem do autor*

Figura 5.4 – Deltas de acurácia para os diferentes conjuntos de dados na tarefa de NLI.



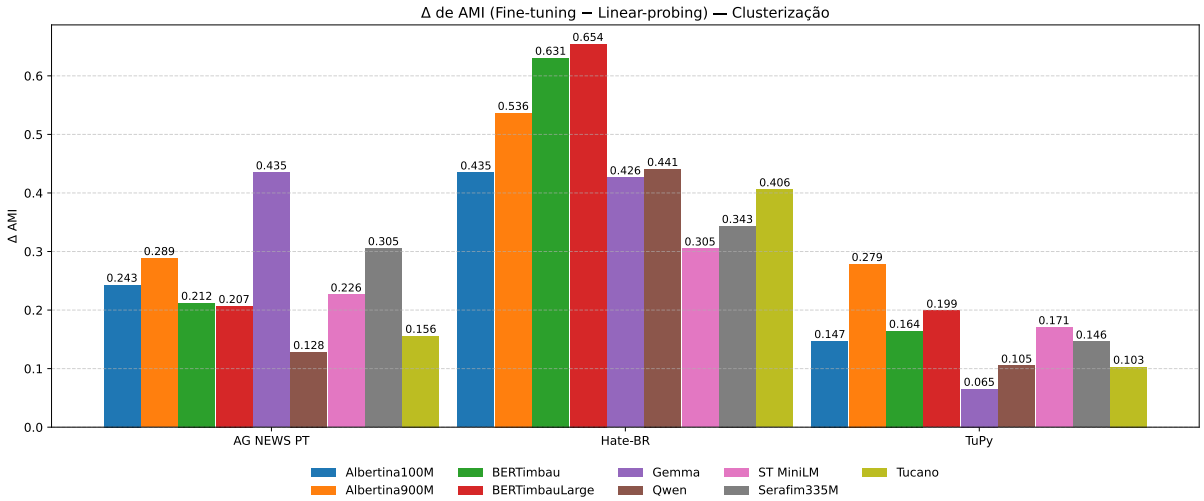
Fonte: *Imagem do autor*

efeitos totalmente contrários. Como evidenciado pela Tabela 5.4 e Figura 5.2, as correlações de Pearson e Spearman possuem uma degradação notória na grande maioria dos resultados.

Em contraste, o Albertina900M e ambas as variantes do BERTimbau não apenas obtêm melhorias nas tarefas de classificação, como também apresentam valores de Δ positivos para as tarefas de STS em ambos os conjuntos de dados avaliados. Notavelmente, esses modelos são os únicos a demonstrar ganhos de desempenho consistentes em todos os cenários experimentais.

A hipótese levantada para tal diferença se dá na origem dos modelos, por serem baseados em BERT, se beneficiam de um espaço de *embeddings* mais plástico, permitindo especializações em diferentes tarefas sem perdas significativas em suas representações semânticas gerais. Tal característica pode ser referida como "plasticidade de adaptação", que surge de um espaço

Figura 5.5 – Deltas de acurácia para os diferentes conjuntos de dados na tarefa de *clusterização*.



Fonte: *Imagem do autor*

geométrico de *embeddings* menos restritivo do que o dos modelos mais rígidos, que necessitam aprender diversas línguas.

Vale destacar que, apesar de pertencerem à mesma família de modelos, Albertina100M e Albertina900M diferem em diversos aspectos. Em particular, eles utilizam tokenizadores distintos (DebertaTokenizerFast e DebertaV2TokenizerFast, respectivamente) e são treinados em corpora diferentes para se adequar a seus diferentes tamanhos (OSCAR e BrWaC, respectivamente) (SANTOS et al., 2024). Essas diferenças arquiteturais e relacionadas aos dados podem explicar as discrepâncias observadas no desempenho de ambos os modelos.

5.2.3 PP3: Qual modelo escolher?

A escolha de um modelo necessita levar em conta fatores que vão além dos resultados das métricas. Também é importante analisar a quantidade de parâmetros dos modelos e como isso se traduz em tempo de execução. Um modelo arbitrário pode possuir ótimas métricas, mas se sua execução for muito demorada, o seu uso pode não fazer sentido em um sistema que necessita de velocidade, por exemplo.

A fim de verificar essa situação, foi calculado o tempo médio (em segundos) de cinco execuções da avaliação dos modelos com o conjunto de testes, com os resultados visíveis na Tabela 5.5. Todos foram avaliados com o ‘max_length‘ de 512 e ‘batch_size‘ 4, limitado superiormente pelo Gemma, o modelo que necessita da maior quantidade de memória para executar.

Para a tarefa de classificação, o BERTimbauLarge apresenta um tempo de execução maior, mas alcança resultados significativamente superiores aos do ST MiniLM, que obtém a maior parte dos melhores resultados. Uma alternativa é o Serafim, que possui um tamanho de modelo comparável e níveis de desempenho semelhantes. Embora o Serafim exija um tempo de execução

Tabela 5.5 – Tempo médio de execução, em segundos, em cada conjunto de dados.

Modelo	Classificação (segundos)			Clusterização (segundos)			NLI (segundos)	STS (segundos)	
	AG News PT	HateBR	TuPy	AG News PT	HateBR	TuPy	ASSIN2	ASSIN	ASSIN2
Albertina100m	1190,527	28,024	430,625	33,371	7,391	38,395	71,356	31,394	18,826
Albertina900m	1800,243	59,084	488,531	90,571	14,028	76,620	125,725	63,767	34,786
BERTimbau	795,823	12,967	**233,455**	15,831	3,865	18,483	**29,450**	**12,688**	8,069
BERTimbauLarge	1039,507	22,827	322,232	31,081	5,985	31,059	51,820	22,369	13,877
Gemma	3026,848	79,020	697,281	141,704	18,039	105,376	146,326	85,862	40,190
Qwen	1501,652	54,261	498,322	61,007	12,210	67,097	131,940	57,971	35,070
Serafim335m	982,286	22,415	325,077	30,706	6,257	32,020	52,619	23,319	14,027
ST MiniLM	**629,193**	**12,589**	288,648	**13,017**	**2,826**	**14,678**	30,563	13,206	**8,047**
Tucano	1412,694	28,883	352,823	43,048	7,917	40,038	65,169	27,637	16,835

Fonte: *Tabela do autor*

ligeiramente maior nos conjuntos HateBR e TuPy, isso é parcialmente compensado por seu tempo de execução substancialmente menor no AG News PT.

Quando a eficiência computacional é uma restrição rigorosa, o BERTimbau surge como um forte candidato, pois obtém resultados muito próximos aos do Serafim e do BERTimbau Large, ao mesmo tempo em que demanda um tempo de execução significativamente menor. Embora seja mais lento que o ST MiniLM, essa diferença não é proibitiva na prática.

Para a tarefa de *clusterização*, o BERTimbauLarge oferece um *trade-off* favorável entre desempenho e tempo de execução. Seus resultados ficam próximos aos dos modelos com melhor desempenho, ao mesmo tempo em que mantém um tempo de execução menor do que aqueles que apresentam maiores pontuações de AMI.

Para as tarefas de STS e NLI, o Serafim335M se destaca como a escolha mais adequada. Apesar de seu tempo de execução significativamente maior, ele entrega de forma consistente resultados substancialmente superiores. O *fine-tuning* pode ser explorado nesse contexto; no entanto, ele pode levar ao esquecimento catastrófico na tarefa de STS, o que sugere que uma configuração de *linear probing* pode ser uma alternativa mais estável e confiável.

Em geral, a melhor estratégia seria a seleção de modelos para tarefas específicas, escolhendo o modelo com a arquitetura mais apropriada para cada desafio. Porém, quando as restrições de tempo são críticas e somente um modelo pode ser escolhido, BERTimbau oferece um bom balanceamento entre eficiência computacional e performance preditiva entre as tarefas.

5.2.4 Análise do balanço entre plasticidade e rigidez

A análise quantitativa dos *tokens* mostra que a hipótese de que *tokenizadores* em português são sempre mais eficientes não se sustenta como uma regra simples. Não foi possível identificar uma correlação direta entre o fato de um modelo ser monolíngue e a geração de uma menor quantidade de *tokens*, nem entre essa característica e um melhor desempenho em tarefas posteriores. Isso fica evidente, por exemplo, no caso do Tucano, um decodificador monolíngue que produziu a menor quantidade de *tokens*, mas que não apresentou desempenhos expressi-

vos nas tarefas avaliadas. Em contraste, o Albertina900M, que se destaca pela geração de um grande número de *tokens*, obteve excelentes resultados em diferentes tarefas. Esses achados refutam a hipótese simplista de que o foco monolíngue, por si só, garante uma representação mais compacta ou mais eficaz, evidenciando uma complexidade maior, aqui denominada "Dilema da Tokenização".

Tabela 5.6 – Número médio de *tokens* por palavra de diferentes modelos ao longo dos conjuntos de dados. Valores em negrito indicam as menores médias por coluna.

Modelo	AG News PT	TuPy	ASSIN	ASSIN2	HateBR	Em Ambiente Real	Média
Albertina100M	2,098	2,186	2,143	2,046	2,418	2,265	2,193
Albertina900M	1,841	1,884	1,855	1,707	2,007	1,914	1,868
BERTimbau	**1,400**	1,644	1,363	1,207	1,626	1,468	1,451
BERTimbauLarge	**1,400**	1,644	1,363	1,207	1,626	1,468	1,451
Gemma	1,416	1,491	1,458	1,187	1,536	1,429	1,420
QWen	1,686	1,749	1,730	1,529	1,874	1,725	1,716
Serafim335M	**1,400**	1,644	1,363	1,207	1,626	1,468	1,451
ST MiniLM	1,460	1,536	1,434	1,308	1,624	1,461	1,471
Tucano	1,311	**1,484**	**1,271**	**1,162**	**1,525**	**1,326**	**1,347**

Fonte: *Tabela do autor*

Uma análise qualitativa do comportamento dos tokenizadores mostrados na Tabela 5.7 de alguns termos do conjunto 'Em Ambiente Real' (Apêndice A) contribui para esclarecer esse dilema ao evidenciar seus diferentes compromissos estratégicos. Observa-se que tokenizadores do tipo WordPiece (BERTimbau, Serafim) tendem a se alinhar de forma mais próxima à morfologia do português, segmentando, por exemplo, *preditiva* em *pred* + *itiva*. No entanto, esses *tokenizadores* apresentam dificuldades com empréstimos linguísticos, fragmentando termos comuns como *fake news* em subunidades com pouco conteúdo semântico.

Em contraste, *tokenizadores* baseados em SentencePiece (Gemma, Tucano) mostram-se mais eficientes em termos de quantidade de *tokens* e mais robustos a empréstimos lexicais, embora suas segmentações sejam, em geral, menos fiéis à estrutura morfológica da língua. Por fim, *tokenizadores* BPE (Albertina100M e Qwen) tendem a segmentar excessivamente palavras em português, o que aumenta a contagem de *tokens* e reduz a clareza linguística.

O dilema da *tokenização* oferece uma explicação plausível para a dicotomia de adaptação observada nos principais resultados. A partir dele, percebe-se que os diferentes comportamentos estão ligados às propriedades inerentes da geometria dos espaços de *embeddings* dos modelos.

Esse comportamento é evidenciado pelo bom desempenho de codificadores monolíngues, como BERTimbau e Albertina900M, que estão associados a uma maior "plasticidade de adaptação". Seus vocabulários especializados e restritos a uma única língua tendem a resultar em uma geometria de *embeddings* mais maleável, uma vez que o modelo pode se concentrar em aprender características específicas de uma única língua, em vez de lidar com restrições interlinguísticas. Dessa forma, levanta-se a hipótese de que essa flexibilidade permita ao processo de *fine-tuning*

Tabela 5.7 – Tokenização qualitativa de diferentes palavras e expressões do conjunto Em Ambiente Real, evidenciando a estrutura morfológica de termos em PT-BR. Contrariamente às expectativas, os tokenizadores do Gemma e do ST MiniLM apresentam forte alinhamento morfológico, enquanto o vocabulário monolíngue do BERTimbau falha ao lidar com empréstimos comuns do inglês.

Modelo	***Tokenização***				
	’*pulsava*”	’*preditiva*”	’*saudosa*”	’*fundamento*”	’*fake news*”
Morfologia	’puls ’a’, ’va’	’predit’, ’iva’	’saud’, ’osa’	’fund’, ’a’, ’mento’	-
Albertina100M	’Ġpuls’, ’ava’	’Ġpred’, ’it’, ’iva’	’Ġsa’, ’ud’, ’osa’	’Ġfundament’, ’o’	’Ġfake’, ’Ġnews’
Albertina900M	’_pul’, ’s’, ’ava’	’_pre’, ’di’, ’tiva’	’_sau’, ’d’, ’osa’	’_fund’, ’amento’	’_fake’, ’_news’
BERTimbau	’pul’, ’##sa’, ’##va’	’pred’, ’##itiva’	’saud’, ’##osa’	’funda’, ’##mento’, ’o’	’fa’, ’##ke’, ’ne’, ’##ws’
BERTimbauLarge	’pul’, ’##sa’, ’##va’	’pred’, ’##itiva’	’saud’, ’##osa’	’funda’, ’##mento’, ’o’	’fa’, ’##ke’, ’ne’, ’##ws’
Gemma	’_puls’, ’ava’	’_predi’, ’tiva’	’_sau’, ’dos’, ’a’	’_fundamento’	’_fake’, ’_news’
Qwen	’Ġpuls’, ’ava’	’Ġpred’, ’it’, ’iva’	’Ġsa’, ’ud’, ’osa’	’Ġfund’, ’amento’, ’Ġo’	’Ġfake’, ’Ġnews’
Serafim335M	’pul’, ’##sa’, ’##va’	’pred’, ’##itiva’	’saud’, ’##osa’	’funda’, ’##mento’, ’o’	’fa’, ’##ke’, ’ne’, ’##ws’
ST MiniLM	’_pulsa’, ’va’	’_pred’, ’i’, ’tiva’	’_sau’, ’dos’, ’a’	’_fundamento’	’_fake’, ’_news’
Tucano	’_puls’, ’ava’	’_pred’, ’itiva’	’_saud’, ’osa’	’_fundamento’	’_f’, ’ake’, ’_new’, ’s.’

Fonte: *Tabela do autor*

reorganizar o espaço de maneira mais eficaz para tarefas centradas no português, explicando a melhoria consistente observada em STS.

Por outro lado, o comportamento de modelos globais reflete uma maior “rigidez de adaptação”. Seus vocabulários multilíngues e objetivos de pré-treinamento são explicitamente projetados para promover um forte alinhamento interlinguístico, o que tende a gerar uma geometria de *embeddings* altamente estruturada e restrita, otimizada para a consistência entre línguas. Embora essa rigidez seja vantajosa para cenários de *linear probing*, ela parece limitar a adaptação específica à tarefa. Sugere-se que o *fine-tuning* leva os modelos ao esquecimento catastrófico ao tentar modificar um espaço geométrico que permite pouca ou nenhuma modificação.

Por fim, esses resultados corroboram com os encontrados por Ahuja, Vaddamanu e Patra (2025), evidenciando que o multilinguismo influência não somente no impacto vetorial, mas também na forma como os modelos se comportam. Dessa forma, diferenças linguísticas podem refletir trajetórias cognitivas distintas dentro do modelo.

6 Considerações Finais

Neste capítulo, na Seção 6.1 será feita um resumo geral do trabalho evidenciando os resultados e os objetivos alcançados, bem como os resultados obtidos. Já a Seção 6.2 exibe as limitações encontradas durante o desenvolvimento do trabalho ao passo que define os caminhos a serem traçados para a continuidade da pesquisa.

6.1 Conclusão

Este trabalho teve como objetivo principal analisar comparativamente o desempenho de modelos de *embeddings* globais e monolíngues para o português brasileiro, considerando tanto a qualidade das representações originais, avaliadas por meio do regime de *linear probing*, quanto o impacto da adaptação supervisionada via *fine-tuning* por meio do PEFT LoRA, em diferentes tarefas. A partir de uma ampla experimentação envolvendo tarefas de classificação, clusterização, NLI e STS, foi possível investigar de forma sistemática os trade-offs entre generalização, especialização e estabilidade dos modelos.

Avaliando a qualidade das representações geradas pelos modelos globais e monolíngues em português brasileiro por meio do regime de *linear-probing*, foi possível perceber que os modelos apresentaram resultados muito próximos na maioria das tarefas. Modelos globais como o Qwen e ST MiniLM se demonstraram muito competitivos, mostrando que representações aprendidas em larga escala podem transmitir conhecimento de forma eficaz para o português brasileiro. Por outro lado, modelos monolíngues, como o BERTimbau e Serafim, mantiveram resultados consistentemente altos, sugerindo que a especialização linguística continua sendo um fator relevante para a qualidade das representações.

A investigação do impacto do *fine-tuning* eficiente de parâmetros utilizando técnicas de PEFT no desempenho dos modelos em tarefas downstream evidenciou os claros ganhos de desempenho obtidos em tarefas de classificação, NLI, e, sobretudo, na *clusterização*, que apresentou melhorias significativas. Por outro lado, os resultados também revelaram que o processo de adaptação pode ser maléfico, como observado na tarefa de STS, onde diversos modelos, especialmente os multilíngues, apresentaram perdas significativas de desempenho após o *fine-tuning*, caracterizando um efeito de esquecimento catastrófico.

Comparando o comportamento dos modelos em diferentes tarefas, a identificação de padrões de estabilidade ou degradação permitiu evidenciar a capacidade dos modelos baseados no BERT, especialmente BERTimbau e Albertina, de apresentar melhorias consistentes ou estabilidade simultânea em múltiplas tarefas, até mesmo no STS. Enquanto isso, modelos globais apresentaram um maior *trade-off* entre adaptação e desempenho, com ganhos expressivos em

algumas tarefas, mas perdas significativas naquelas que dependiam fortemente da preservação da geometria semântica do espaço de *embeddings*.

As análises quantitativas e qualitativas dos *tokens* gerados refutam hipóteses simplistas de que *tokenizadores* monolíngues são sempre mais eficientes ou superiores, estando mais alinhados com a língua portuguesa. Em vez disso, observou-se um "dilema da tokenização", onde estratégias diferentes apresentam resultados distintos em relação à eficiência, ao alinhamento morfológico e à robustez a termos estrangeiros. Tal análise contribuiu para explicar parte das diferenças observadas no comportamento dos modelos durante o processo de adaptação.

Em suma, os resultados obtidos permitem concluir que o objetivo principal do trabalho foi alcançado. A análise comparativa conduzida evidenciou que modelos globais modernos são altamente competitivos em português brasileiro em regimes de *linear-probing*, mas podem apresentar maior rigidez de adaptação, levando a instabilidades após o treinamento. Por outro lado, modelos monolíngues apresentam maior plasticidade de adaptação, equilibrando melhor os ganhos de desempenho e a preservação das representações semânticas. Esses resultados contribuem para um maior entendimento do papel da especialização linguística e fornecem análises práticas para a escolha e adaptação de modelos em aplicações reais de PLN para o português brasileiro.

6.2 Trabalhos Futuros

Os conjuntos de dados utilizados neste trabalho possuem um grande foco em postagens de redes sociais e notícias. Assim, a exploração de variações de texto, como jurídicos, científicos, literários, etc., pode ajudar a entender mais sobre a generalização dos modelos.

O uso de métricas puras ajuda na comparação dos modelos, mas não explica o porquê das modificações nos resultados, como a queda dos resultados no STS. Assim, com a criação de um conjunto de testes específicos, pode-se investigar como o modelo muda após o processo de adaptação, ou seja, se passa a preferir certos significados de palavras ambíguas (polissemia), se altera a ordem das palavras ou tipos de frases, e se muda a forma como expressa uma avaliação, opinião ou atitude.

Por fim, é possível tentar medir a "plasticidade de adaptação" dos modelos a partir de uma métrica que identifique o quanto o modelo muda quando é forçado a se especializar e quais características linguísticas se tornam mais ou menos importantes após o *fine-tuning*.

Referências

AHUJA, S.; VADDAMANU, P.; PATRA, B. Efficientxlang: Towards improving token efficiency through cross-lingual reasoning. In: *Findings of the Association for Computational Linguistics: EMNLP 2025*. [S.l.: s.n.], 2025. p. 15612–15624.

ALAGöZLü, M. Stochastic gradient descent variants and applications. 02 2022.

ALAIN, G.; BENGIO, Y. *Understanding intermediate layers using linear classifier probes*. 2018. Disponível em: <https://arxiv.org/abs/1610.01644>.

ALAMMAR, J.; GROOTENDORST, M. *Hands-on large language models*. Sebastopol, CA: O'Reilly Media, 2024.

AZWARSHARIQ. Understanding natural language inference: From models to containers. *Medium*, 07 2023. Acessado em: 14/02/2026. Disponível em: <https://medium.com/red-buffer/title-understanding-natural-language-inference-from-models-to-containers-26c1e777d72f>.

BOUDAA, S.; BOUDAA, T.; HADDADI, A. E. Semantic textual similarity: Overview and comparative study between arabic and english. *Comput. Sist.*, Instituto Politecnico Nacional/Centro de Investigacion en Computacion, v. 28, n. 3, set. 2024.

BURKOV, A. *The hundred-page machine learning book*. [S.l.]: True Positive, 2019.

CAMPBELL, M.; HOANE, A.; HSU, F. hsiung. Deep blue. *Artificial Intelligence*, v. 134, n. 1, p. 57–83, 2002. ISSN 0004-3702. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0004370201001291>.

CHEN, P.; JI, S.; BOGOYCHEV, N.; KUTUZOV, A.; HADDOW, B.; HEAFIELD, K. Monolingual or multilingual instruction tuning: Which makes a better alpaca. In: GRAHAM, Y.; PURVER, M. (Ed.). *Findings of the Association for Computational Linguistics: EACL 2024*. St. Julian's, Malta: Association for Computational Linguistics, 2024. p. 1347–1356. Disponível em: <https://aclanthology.org/2024.findings-eacl.90/>.

CHONG, B. et al. K-means clustering algorithm: a brief review. *Academic Journal of Computing & Information Science*, Francis Academic Press, v. 4, n. 5, p. 37–40, 2021.

COELHO, P. *O Alquimista*. [S.l.]: Pergaminho, 2001. 34 p. ISBN 9789727110797.

CONNEAU, A.; KHANDELWAL, K.; GOYAL, N.; CHAUDHARY, V.; WENZEK, G.; GUZMÁN, F.; GRAVE, E.; OTT, M.; ZETTLEMOYER, L.; STOYANOV, V. Unsupervised cross-lingual representation learning at scale. In: *Proceedings of the 58th annual meeting of the association for computational linguistics*. [S.l.: s.n.], 2020. p. 8440–8451.

CORRÊA, N. K.; SEN, A.; FALK, S.; FATIMAH, S. Tucano: Advancing Neural Text Generation for Portuguese. *Patterns*, Elsevier, 2025. ISSN 2666-3899. Disponível em: <https://doi.org/10.1016/j.patter.2025.101325>.

DEVLIN, J.; CHANG, M.-W.; LEE, K.; TOUTANOVA, K. 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)*. [S.l.: s.n.], 2019. p. 4171–4186.

ELHARROUSS, O.; MAHMOOD, Y.; BECHQITO, Y.; SERHANI, M. A.; BADIDI, E.; RIFFI, J.; TAIRI, H. *Loss Functions in Deep Learning: A Comprehensive Review*. 2025. Disponível em: <https://arxiv.org/abs/2504.04242>.

FONSECA, E.; SANTOS, L.; CRISCUOLO, M.; ALUISIO, S. Assin: Avaliacao de similaridade semantica e inferencia textual. In: *Computational Processing of the Portuguese Language-12th International Conference, Tomar, Portugal*. [S.l.: s.n.], 2016. p. 13–15.

GARCIA, G. L.; PAIOLA, P. H.; MORELLI, L. H.; CANDIDO, G.; JúNIOR, A. C.; JODAS, D. S.; AFONSO, L. C. S.; GUILHERME, I. R.; PENTEADO, B. E.; PAPA, J. P. *Introducing Bode: A Fine-Tuned Large Language Model for Portuguese Prompt-Based Task*. 2024.

GENG, S.; HSIEH, C.-Y.; RAMANUJAN, V.; WALLINGFORD, M.; LI, C.-L.; KOH, P. W.; KRISHNA, R. *The Unmet Promise of Synthetic Training Images: Using Retrieved Real Images Performs Better*. 2025. Disponível em: <https://arxiv.org/abs/2406.05184>.

GHOSH, K.; SENAPATI, A. Hate speech detection in low-resourced indian languages: An analysis of transformer-based monolingual and multilingual models with cross-lingual experiments. *Natural Language Processing*, v. 31, n. 2, p. 393–414, 2025.

GOMES, L.; BRANCO, A.; SILVA, J.; RODRIGUES, J.; SANTOS, R. *Open Sentence Embeddings for Portuguese with the Serafim PT* encoders family*. 2024. Disponível em: <https://arxiv.org/abs/2407.19527>.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep Learning*. [S.l.]: MIT Press, 2016. <http://www.deeplearningbook.org>.

HARSHA, S. S.; SWAROOP, K. K.; CHANDAVARKAR, B. R. Natural language inference: Detecting contradiction and entailment in multilingual text. In: VENUGOPAL, K. R.; SHENOY, P. D.; BUYYA, R.; PATNAIK, L. M.; IYENGAR, S. S. (Ed.). *Data Science and Computational Intelligence*. Cham: Springer International Publishing, 2021. p. 314–327. ISBN 978-3-030-91244-4.

HE, Z.; DUMDUMAYA, C. E.; QUIMNO, V. Measurement of semantic text similarity. *Journal of Theoretical and Applied Information Technology*, v. 102, n. 5, 2024.

HOULSBY, N.; GIURGIU, A.; JASTRZEBSKI, S.; MORRONE, B.; LAROUSSILHE, Q. de; GESMUNDO, A.; ATTARIYAN, M.; GELLY, S. *Parameter-Efficient Transfer Learning for NLP*. 2019. Disponível em: <https://arxiv.org/abs/1902.00751>.

HU, E. J.; SHEN, Y.; WALLIS, P.; ALLEN-ZHU, Z.; LI, Y.; WANG, S.; WANG, L.; CHEN, W. *LoRA: Low-Rank Adaptation of Large Language Models*. 2021. Disponível em: <https://arxiv.org/abs/2106.09685>.

HUGGINFACE. *LoRA*. Hugging Face, n.d. Disponível em: <https://huggingface.co/docs/peft/package_reference/lora>.

JOHNSON, A. Tudo o que você precisa saber sobre o chatgpt da openai. *Forbes*, 12 2022. Acessado em: 13/02/2026. Disponível em: <https://forbes.com.br/forbes-tech/2022/12/tudo-o-que-voce-precisa-saber-sobre-o-chatgpt-da-openai/>.

JONES, K. S. A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*, v. 28, n. 1, p. 11–21, 01 1972. ISSN 0022-0418. Disponível em: <https://doi.org/10.1108/eb026526>.

KAPLAN, A.; HAENLEIN, M. Siri, siri, in my hand: Who's the fairest in the land? on the interpretations, illustrations, and implications of artificial intelligence. *Business Horizons*, v. 62, n. 1, p. 15–25, 2019. ISSN 0007-6813. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0007681318301393>.

KIRKPATRICK, J.; PASCANU, R.; RABINOWITZ, N.; VENESS, J.; DESJARDINS, G.; RUSU, A. A.; MILAN, K.; QUAN, J.; RAMALHO, T.; GRABSKA-BARWINSKA, A. et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, National Academy of Sciences, v. 114, n. 13, p. 3521–3526, 2017.

KOSTIć, M.; BATANOVIć, V.; NIKOLIć, B. Monolingual, multilingual and cross-lingual code comment classification. *Engineering Applications of Artificial Intelligence*, v. 124, p. 106485, 2023. ISSN 0952-1976. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0952197623006693>.

KUDO, T.; RICHARDSON, J. *SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing*. 2018. Disponível em: <https://arxiv.org/abs/1808.06226>.

LAI, V. D.; NGO, N. T.; VEYSEH, A. P. B.; MAN, H.; DERNONCOURT, F.; BUI, T.; NGUYEN, T. H. *ChatGPT Beyond English: Towards a Comprehensive Evaluation of Large Language Models in Multilingual Learning*. 2023. Disponível em: <https://arxiv.org/abs/2304.05613>.

LECUN, Y.; BENGIO, Y.; HINTON, G. Deep learning. *Nature*, v. 521, n. 7553, p. 436–444, 2015. ISSN 1476-4687. Disponível em: <https://doi.org/10.1038/nature14539>.

LI, Q.; PENG, H.; LI, J.; XIA, C.; YANG, R.; SUN, L.; YU, P. S.; HE, L. *A Survey on Text Classification: From Shallow to Deep Learning*. 2021. Disponível em: <https://arxiv.org/abs/2008.00364>.

LI, S.; WONG, K. W.; WANG, G.; DUONG, T.-T. A systematic review of multi-modal large language models on domain-specific applications. *Artif. Intell. Rev.*, Springer Science and Business Media LLC, v. 58, n. 12, out. 2025.

Maicon Domingues. *canarim-7b (Revision 08fdd2b)*. Hugging Face, 2023. Disponível em: <https://huggingface.co/dominguesm/canarim-7b>.

MARITACA-AI. *ag_news_pt*. Hugging Face, 2023. Disponível em: <https://huggingface.co/datasets/maritaca-ai/ag_news_pt>.

MIKOLOV, T.; CHEN, K.; CORRADO, G.; DEAN, J. *Efficient Estimation of Word Representations in Vector Space*. 2013. Disponível em: <https://arxiv.org/abs/1301.3781>.

MITCHELL, T. M.; MUNSON, E. M. *Machine learning*. [S.l.]: WCB/McGraw-Hill, 1997.

NIELSEN, M. *Neural Networks and Deep Learning*. [S.l.]: Determination Press, 2015.

PENNINGTON, J.; SOCHER, R.; MANNING, C. GloVe: Global vectors for word representation. In: MOSCHITTI, A.; PANG, B.; DAELEMANS, W. (Ed.). *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, 2014. p. 1532–1543. Disponível em: <https://aclanthology.org/D14-1162/>.

PETROV, A.; MALFA, E. L.; TORR, P.; BIBI, A. Language model tokenizers introduce unfairness between languages. In: OH, A.; NAUMANN, T.; GLOBERSON, A.; SAENKO, K.; HARDT, M.; LEVINE, S. (Ed.). *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2023. v. 36, p. 36963–36990. Disponível em: <https://proceedings.neurips.cc/paper_files/paper/2023/file/74bb24dca8334adce292883b4b651eda-Paper-Conference.pdf>.

PIRES, R.; ABONIZIO, H.; ALMEIDA, T. S.; NOGUEIRA, R. Sabiá: Portuguese large language models. In: ____. *Intelligent Systems*. Springer Nature Switzerland, 2023. p. 226–240. ISBN 9783031453922. Disponível em: <http://dx.doi.org/10.1007/978-3-031-45392-2_15>.

QADER, W. A.; AMEEN, M. M.; AHMED, B. I. An overview of bag of words;importance, implementation, applications, and challenges. In: *2019 International Engineering Conference (IEC)*. [S.l.: s.n.], 2019. p. 200–204.

RADFORD, A.; NARASIMHAN, K.; SALIMANS, T.; SUTSKEVER, I. et al. Improving language understanding by generative pre-training. San Francisco, CA, USA, 2018.

REAL, L.; FONSECA, E.; OLIVEIRA, H. G. The assin 2 shared task: a quick overview. In: SPRINGER. *International Conference on Computational Processing of the Portuguese Language*. [S.l.], 2020. p. 406–412.

REIMERS, N.; GUREVYCH, I. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084*, 2019.

REIMERS, N.; GUREVYCH, I. Sentence-bert: Sentence embeddings using siamese bert-networks. In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2019. Disponível em: <http://arxiv.org/abs/1908.10084>.

REUTERS. Chatgpt tem recorde de crescimento da base de usuários. *Forbes*, 02 2023. Acessado em: 13/02/2026. Disponível em: <https://forbes.com.br/forbes-tech/2023/02/chatgpt-tem-recorde-de-crescimento-da-base-de-usuarios/>.

RODRIGUES, J.; GOMES, L.; SILVA, J.; BRANCO, A.; SANTOS, R.; CARDOSO, H. L.; OSóRIO, T. Advancing neural encoding of portuguese with transformer albertina pt-*. In: ____. *Progress in Artificial Intelligence*. Springer Nature Switzerland, 2023. p. 441–453. ISBN 9783031490088. Disponível em: <http://dx.doi.org/10.1007/978-3-031-49008-8_35>.

ROHR, A. Computador convence juízes de que é garoto de 13 anos em 'teste de turing'. *G1*, 06 2014. Acessado em: 13/02/2026. Disponível em: <https://g1.globo.com/tecnologia/noticia/2014/06/computador-convence-juizes-que-e-garoto-de-13-anos-em-teste-de-turing.html>.

ROSENBLATT, F. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, American Psychological Association, v. 65, n. 6, p. 386, 1958.

RUMELHART, D. E.; HINTON, G. E.; WILLIAMS, R. J. Learning representations by back-propagating errors. *Nature*, v. 323, n. 6088, p. 533–536, Oct 1986. ISSN 1476-4687. Disponível em: <https://doi.org/10.1038/323533a0>.

RUSSELL, S. J.; NORVIG, P. *Artificial Intelligence: A modern approach*. [S.l.]: Pearson, 2021.

RUST, P.; PFEIFFER, J.; VULIĆ, I.; RUDER, S.; GUREVYCH, I. How good is your tokenizer? on the monolingual performance of multilingual language models. In: *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. [S.l.: s.n.], 2021. p. 3118–3135.

SAJUN, A. R.; ZUALKERNAN, I.; SANKALPA, D. A historical survey of advances in transformer architectures. *Applied Sciences*, v. 14, n. 10, 2024. ISSN 2076-3417. Disponível em: <https://www.mdpi.com/2076-3417/14/10/4316>.

SAMUEL, A. L. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, v. 3, n. 3, p. 210–229, 1959.

SANDERSON, G. *Mas o que é um GPT? Introdução visual aos Transformadores | Aprendizagem profunda, capítulo 5*. 2024. Disponível em: <https://youtu.be/wjZofJX0v4M?si=lV_5yFiSxa29LVku>.

SANTOS, R.; RODRIGUES, J.; GOMES, L.; SILVA, J.; BRANCO, A.; CARDOSO, H. L.; OSóRIO, T. F.; LEITE, B. *Fostering the Ecosystem of Open Neural Encoders for Portuguese with Albertina PT* Family*. 2024. Disponível em: <https://arxiv.org/abs/2403.01897>.

SENNRICH, R.; HADDOW, B.; BIRCH, A. *Neural Machine Translation of Rare Words with Subword Units*. 2016. Disponível em: <https://arxiv.org/abs/1508.07909>.

Silly-Machine. *TuPyE-Dataset*. Hugging Face, 2024. Disponível em: <https://huggingface.co/datasets/Silly-Machine/TuPyE-Dataset>.

SINGH, P.; CLERCQ, O. D.; LEFEVER, E. Distilling monolingual models from large multilingual transformers. *Electronics*, v. 12, n. 4, 2023. ISSN 2079-9292. Disponível em: <https://www.mdpi.com/2079-9292/12/4/1022>.

SONG, X.; SALCIANU, A.; SONG, Y.; DOPSON, D.; ZHOU, D. *Fast WordPiece Tokenization*. 2021. Disponível em: <https://arxiv.org/abs/2012.15524>.

SOUZA, F.; NOGUEIRA, R.; LOTUFO, R. Bertimbau: pretrained bert models for brazilian portuguese. In: *Intelligent Systems: 9th Brazilian Conference, BRACIS 2020, Rio Grande, Brazil, October 20–23, 2020, Proceedings, Part I*. [S.l.]: Springer, 2020. p. 403–417.

STEWART, J. *Cálculo, Volume 2*. 8. ed. São Paulo: Cengage Learning, 2017. ISBN 9788522129971.

TEAM, G. Gemma. Kaggle, 2024. Disponível em: <https://www.kaggle.com/m/3301>.

TURING, A. M. I.—computing machinery and intelligence. *Mind*, LIX, n. 236, p. 433–460, 10 1950. ISSN 0026-4423. Disponível em: <https://doi.org/10.1093/mind/LIX.236.433>.

VARGAS, F.; CARVALHO, I.; GÓES, F. Rodrigues de; PARDO, T.; BENEVENUTO, F. HateBR: A large expert annotated corpus of Brazilian Instagram comments for offensive language and hate speech detection. In: *Proceedings of the 13th Conference on Language Resources and Evaluation (LREC 2022)*. Marseille, France: European Language Resources Association, 2022. p. 7174–7183. Disponível em: <https://aclanthology.org/2022.lrec-1.777>.

VASWANI, A.; SHAZEER, N.; PARMAR, N.; USZKOREIT, J.; JONES, L.; GOMEZ, A. N.; KAISER, Ł.; POLOSUKHIN, I. Attention is all you need. *Advances in neural information processing systems*, v. 30, 2017.

VINH, N. X.; EPPS, J.; BAILEY, J. Information theoretic measures for clusterings comparison: is a correction for chance necessary? In: *Proceedings of the 26th Annual International Conference on Machine Learning*. New York, NY, USA: Association for Computing Machinery, 2009. (ICML '09), p. 1073–1080. ISBN 9781605585161. Disponível em: <https://doi.org/10.1145/1553374.1553511>.

VIRTANEN, A.; KANERVA, J.; ILO, R.; LUOMA, J.; LUOTOLAHTI, J.; SALAKOSKI, T.; GINTER, F.; PYYSALO, S. *Multilingual is not enough: BERT for Finnish*. 2019. Disponível em: <https://arxiv.org/abs/1912.07076>.

VRBANčIč, G.; PODGORELEC, V. Transfer learning with adaptive fine-tuning. *IEEE Access*, v. 8, p. 196197–196211, 2020.

WINTER, J. C. D.; GOSLING, S. D.; POTTER, J. Comparing the pearson and spearman correlation coefficients across distributions and sample sizes: A tutorial using simulations and empirical data. *Psychological methods*, American Psychological Association, v. 21, n. 3, p. 273, 2016.

WU, S.; DREDZE, M. *Are All Languages Created Equal in Multilingual BERT?* 2020. Disponível em: <https://arxiv.org/abs/2005.09093>.

XIAN, Y.; LAMPERT, C. H.; SCHIELE, B.; AKATA, Z. *Zero-Shot Learning – A Comprehensive Evaluation of the Good, the Bad and the Ugly*. 2020. Disponível em: <https://arxiv.org/abs/1707.00600>.

ZHANG, A.; LIPTON, Z. C.; LI, M.; SMOLA, A. J. *Dive into Deep Learning*. [S.l.]: Cambridge University Press, 2023. <https://D2L.ai>.

ZHANG, X. *AG's News Topic Classification Dataset*. Hugging Face, 2015. Disponível em: <https://huggingface.co/datasets/sh0416/ag_news>.

ZHANG, Y.; LI, M.; LONG, D.; ZHANG, X.; LIN, H.; YANG, B.; XIE, P.; YANG, A.; LIU, D.; LIN, J.; HUANG, F.; ZHOU, J. Qwen3 embedding: Advancing text embedding and reranking through foundation models. *arXiv preprint arXiv:2506.05176*, 2025.

Apêndices

APÊNDICE A – Conjunto Em Ambiente Real

O conjunto "*Em Ambiente Real*" contém frases, em português brasileiro, de diferentes contextos, sendo utilizadas para corroborar na análise dos resultados dos modelos. Tais sentenças e suas respectivas áreas são descritas abaixo:

- Palavras comuns com acentos:
  - "O coração pulsava com uma emoção indescritível."
- Contrações, gírias e coloquialismo:
  - "Cê tá ligado que a gente vai praquele trampo novo, né?"
  - "Uai, cê bobo, sô! Uai é uai, uai!"
- Termos de domínios específicos:
  - "A manutenção preditiva do rolamento de esferas evitou a parada da linha de produção."
  - "Esse político é um esquerdopata, só vive de lacração e fake news."
  - "O fundo de investimento anunciou um desinvestimento estratégico para aumentar a liquidez."
- Literatura brasileira:
  - "É justamente a possibilidade de realizar um sonho que torna a vida interessante."
  - "Ao verme que primeiro roeu as frias carnes do meu cadáver dedico como saudosa estas memórias póstumas"
  - "Se a única coisa de que o homem terá certeza é a morte; a única certeza do brasileiro é o carnaval no próximo ano."
- Cinema brasileiro:
  - "Eu posso até te ajudar, aliás, eu vou te ajudar! Eu quero te ajudar! Mas agora você tem que me ajudar a te ajudar.",
  - "Parecia até uma mensagem de Deus: se liga aí, mané, a honestidade não compensa.",
- Legislação brasileira:

- "A disciplina do uso da internet no Brasil tem como fundamento o respeito à liberdade de expressão."
- "Esta Lei dispõe sobre o tratamento de dados pessoais, inclusive nos meios digitais."
- "atividades de investigação e repressão de infrações penais"

Declaração

Durante a preparação deste documento, eu, Matheus Peixoto Ribeiro Vieira, estudante de graduação do curso de Ciência da Computação, declaro o uso de IAGen ChatGPT, versão 4 e 5.2, para ajustes em código, revisão de conceitos teóricos, correção de gramática e ortografia e sanar dúvidas provenitente da leitura de artigos.

Após o uso desta ferramenta/modelo/serviço, revisei e editei o conteúdo em conformidade com os princípios éticos para uso de IAGen (Resolução CONPEP 144) e com os acordos estabelecidos com a pessoa orientadora da pesquisa. Dessa forma, assumo total responsabilidade pelo conteúdo da publicação.

A IAGen foi utilizada nas seguintes etapas:

- Etapa 1: Auxiliar na geração de imagem. Gerar comandos para a criação de imagens utilizando o Graphviz; Correção de scripts de imagens; Alterações pontuais. O modelo utilizado foi o **ChatGPT 4o**. O *prompt* utilizado foi:
  - Gere o código para a criação de uma imagem com Graphviz com três círculos, o primeiro com cor cinza e texto "Unidade S", com seis setas ligadas a um círculo azul com texto "Unidade A", que está conectado, por duas setas, a um círculo verde com texto "Undiade R".
  - Aumente o tamanho da fonte
  - Faça com que a imagem esteja na horizontal
  - Gere o código para a criação de uma imagem de uma mlp com quatro camadas, a primeira com três neurônios, a segunda com cinco, a terceira com quatro e a última com duas. A camada de entrada deve ser cinza, as camadas ocultas devem ser azuis e a de saída deve ser verde. Também preciso que você gere labels para descrever cada uma das camadas.
- Etapa 2: Auxílio em código. Geração de código, explicação de funções, correção de erros. O modelo utilizado foi o **ChatGPT 5.2**. O *prompt* utilizado foi:
  - Me explique o que está acontecendo nesse código [...]
  - Para que serve essa função [...]
  - Gere um código para realizar o treinamento do modelo utilizando o PyTorch
  - Gere um código para realizar o salvamento de um modelo treinado pela biblioteca do HugginFace juntamente com o LoRA. Quando eu carrego novamente o modelo, os resultados são iguais aos que estavam ocorrendo antes do treinamento.

  - Dada essa tabela de resultados gere um código para criar um mapa de calor com os valores dos deltas. As linhas devem ser os modelos e as colunas os deltas de acordo com dataset-tarefa-metrica
  - Gere um código para exibir um gráfico de barras com os valores de delta para cada tarefa e dataset dessa tabela de resultados

- Etapa 3: Auxílio em geração de texto. Resumo de textos, correção de ortografia, melhora de fluidez. O modelo utilizado foi o **ChatGPT 5.2**. O *prompt* utilizado foi:

  ... Dado esse texto, corrija a ortografia dele para ficar de acordo com a norma formal
  - Resuma o seguinte texto para ficar com três parágrafos
  - Me ajude a melhorar a fluidez desse texto para ter melhor conexão entre os parágrafos e seguir uma linha de raciocínio